MSIN0143 Group Coursework : A2

Word count: 1934

1 Introduction

The US grocery retail industry is facing declining gross margins due to changes in consumer preferences and increased competition (1). Thus, many US grocery retailers need to adapt their strategies and operations to better serve the rapidly evolving needs of their clients.

This report analyses Dunnhumby's grocery data to gain insights into consumer preferences and spending habits and ultimately proposes a set of informed strategies to help US grocery retailers increase sales in four food categories: oral hygiene, bag snacks, frozen pizza, and cold cereals. The data is publicly available on Dunnhumby's website, providing product sales and promotion information recorded weekly (2).

2 Data Preparation

2.1 Imports

In [1]:
```
#Installing and importing all the required libriaries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import matplotlib.dates as mdates
import datetime
!pip install -U seaborn
%matplotlib inline
import seaborn as sns
import warnings
from IPython.display import Image

from sklearn.svm import SVR
from sklearn.pipeline import make_pipeline
```

```
from sklearn.preprocessing import StandardScaler
warnings.filterwarnings('ignore')
!pip install xgboost
!pip install scikit-learn
import xgboost as xgb
from xgboost import plot_importance, plot_tree
from sklearn.metrics import mean_squared_error as mse
from sklearn.metrics import mean_absolute_percentage_error as mape

!pip install lightgbm
from lightgbm import LGBMRegressor
from sklearn.model_selection import GridSearchCV
```

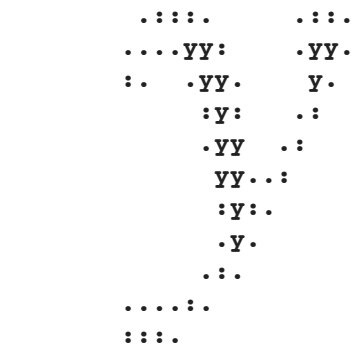
```
/bin/bash: /opt/anaconda/envs/Python3/lib/libtinfo.so.6: no version information available (required by /bin/bash)
```


- Project files and data should be stored in /project. This is shared among everyone in the project.
- Personal files and configuration should be stored in /home/faculty.
- Files outside /project and /home/faculty will be lost when this server is terminated.
- Create custom environments to setup your servers reproducibly.

```
bash: /opt/anaconda/envs/Python3/lib/libtinfo.so.6: no version information available (required by bash)
Requirement already satisfied: seaborn in /opt/anaconda/envs/Python3/lib/python3.9/site-packages (0.12.1)
Requirement already satisfied: numpy>=1.17 in /opt/anaconda/envs/Python3/lib/python3.9/site-packages (from seaborn)
 (1.21.5)
Requirement already satisfied: matplotlib!=3.6.1,>=3.1 in /opt/anaconda/envs/Python3/lib/python3.9/site-packages (from
seaborn) (3.5.3)
Requirement already satisfied: pandas>=0.25 in /opt/anaconda/envs/Python3/lib/python3.9/site-packages (from seaborn)
 (1.5.1)
Requirement already satisfied: cycler>=0.10 in /opt/anaconda/envs/Python3/lib/python3.9/site-packages (from matplotli
b!=3.6.1,>=3.1->seaborn) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in /opt/anaconda/envs/Python3/lib/python3.9/site-packages (from matpl
otlib!=3.6.1,>=3.1->seaborn) (4.25.0)
Requirement already satisfied: python-dateutil>=2.7 in /opt/anaconda/envs/Python3/lib/python3.9/site-packages (from ma
tplotlib!=3.6.1,>=3.1->seaborn) (2.8.2)
Requirement already satisfied: pyparsing>=2.2.1 in /opt/anaconda/envs/Python3/lib/python3.9/site-packages (from matplo
tlib!=3.6.1,>=3.1->seaborn) (3.0.9)
Requirement already satisfied: kiwisolver>=1.0.1 in /opt/anaconda/envs/Python3/lib/python3.9/site-packages (from matpl
otlib!=3.6.1,>=3.1->seaborn) (1.4.2)
Requirement already satisfied: packaging>=20.0 in /opt/anaconda/envs/Python3/lib/python3.9/site-packages (from matplot
lib!=3.6.1,>=3.1->seaborn) (21.3)
Requirement already satisfied: pillow>=6.2.0 in /opt/anaconda/envs/Python3/lib/python3.9/site-packages (from matplotli
b!=3.6.1,>=3.1->seaborn) (9.2.0)
Requirement already satisfied: pytz>=2020.1 in /opt/anaconda/envs/Python3/lib/python3.9/site-packages (from pandas>=0.
25->seaborn) (2022.1)
Requirement already satisfied: six>=1.5 in /opt/anaconda/envs/Python3/lib/python3.9/site-packages (from python-dateuti
```

```
l>=2.7->matplotlib!=3.6.1,>=3.1->seaborn) (1.16.0)
/bin/bash: /opt/anaconda/envs/Python3/lib/libtinfo.so.6: no version information available (required by /bin/bash)

         .:::.     .::.
        ....yy:    .yy.
        :.  .yy.    y.
             :y:   .:
             .yy  .:
              yy..:
              :y:.
              .y.
             .:.
        ....:.
        :::.

• Project files and data should be stored in /project. This is shared among everyone
  in the project.
• Personal files and configuration should be stored in /home/faculty.
• Files outside /project and /home/faculty will be lost when this server is terminated.
• Create custom environments to setup your servers reproducibly.

bash: /opt/anaconda/envs/Python3/lib/libtinfo.so.6: no version information available (required by bash)
Requirement already satisfied: xgboost in /opt/anaconda/envs/Python3/lib/python3.9/site-packages (1.7.2)
Requirement already satisfied: numpy in /opt/anaconda/envs/Python3/lib/python3.9/site-packages (from xgboost) (1.21.5)
Requirement already satisfied: scipy in /opt/anaconda/envs/Python3/lib/python3.9/site-packages (from xgboost) (1.7.3)
/bin/bash: /opt/anaconda/envs/Python3/lib/libtinfo.so.6: no version information available (required by /bin/bash)

         .:::.     .::.
        ....yy:    .yy.
        :.  .yy.    y.
             :y:   .:
             .yy  .:
              yy..:
              :y:.
              .y.
             .:.
        ....:.
        :::.

• Project files and data should be stored in /project. This is shared among everyone
  in the project.
• Personal files and configuration should be stored in /home/faculty.
• Files outside /project and /home/faculty will be lost when this server is terminated.
• Create custom environments to setup your servers reproducibly.
```

```
bash: /opt/anaconda/envs/Python3/lib/libtinfo.so.6: no version information available (required by bash)
Requirement already satisfied: scikit-learn in /opt/anaconda/envs/Python3/lib/python3.9/site-packages (1.0.2)
Requirement already satisfied: joblib>=0.11 in /opt/anaconda/envs/Python3/lib/python3.9/site-packages (from scikit-lea
rn) (1.1.1)
Requirement already satisfied: scipy>=1.1.0 in /opt/anaconda/envs/Python3/lib/python3.9/site-packages (from scikit-lea
rn) (1.7.3)
Requirement already satisfied: numpy>=1.14.6 in /opt/anaconda/envs/Python3/lib/python3.9/site-packages (from scikit-le
arn) (1.21.5)
Requirement already satisfied: threadpoolctl>=2.0.0 in /opt/anaconda/envs/Python3/lib/python3.9/site-packages (from sc
ikit-learn) (2.2.0)
/bin/bash: /opt/anaconda/envs/Python3/lib/libtinfo.so.6: no version information available (required by /bin/bash)
```



- Project files and data should be stored in /project. This is shared among everyone in the project.
- Personal files and configuration should be stored in /home/faculty.
- Files outside /project and /home/faculty will be lost when this server is terminated.
- Create custom environments to setup your servers reproducibly.

```
bash: /opt/anaconda/envs/Python3/lib/libtinfo.so.6: no version information available (required by bash)
Requirement already satisfied: lightgbm in /opt/anaconda/envs/Python3/lib/python3.9/site-packages (3.3.3)
Requirement already satisfied: numpy in /opt/anaconda/envs/Python3/lib/python3.9/site-packages (from lightgbm) (1.21.
5)
Requirement already satisfied: scipy in /opt/anaconda/envs/Python3/lib/python3.9/site-packages (from lightgbm) (1.7.3)
Requirement already satisfied: scikit-learn!=0.22.0 in /opt/anaconda/envs/Python3/lib/python3.9/site-packages (from li
ghtgbm) (1.0.2)
Requirement already satisfied: wheel in /opt/anaconda/envs/Python3/lib/python3.9/site-packages (from lightgbm) (0.37.
1)
Requirement already satisfied: joblib>=0.11 in /opt/anaconda/envs/Python3/lib/python3.9/site-packages (from scikit-lea
rn!=0.22.0->lightgbm) (1.1.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in /opt/anaconda/envs/Python3/lib/python3.9/site-packages (from sc
ikit-learn!=0.22.0->lightgbm) (2.2.0)
```

2.2 Data merging

The Dunnhumby dataset, "The Breakfast At the Frat," consists of three individual files: a transaction file, a store file and a product file. These three files were merged by performing a left join to create a single dataset for easy reporting and data analysis.

Transaction file

The transaction file includes data regarding weekly grocery transactions at the product level from stores in 4 states (Kentucky, Ohio, Texas and Indiana). It contains five products from each of the three major brands in four categories: frozen pizza, pretzels, cold cereal and mouthwash.

In [2]:
```
#read transaction CSV file into dataframe using Pandas
transaction=pd.read_csv('dunnhumby_transaction.csv')
transaction.head()
```

Out[2]:

	WEEK_END_DATE	STORE_NUM	UPC	UNITS	VISITS	HHS	SPEND	PRICE	BASE_PRICE	FEATURE	DISPLAY	TPR_ONLY
0	14-Jan-09	367.0	1.111009e+09	13.0	13.0	13.0	18.07	1.39	1.57	0.0	0.0	1.0
1	14-Jan-09	367.0	1.111009e+09	20.0	18.0	18.0	27.80	1.39	1.39	0.0	0.0	0.0
2	14-Jan-09	367.0	1.111010e+09	14.0	14.0	14.0	19.32	1.38	1.38	0.0	0.0	0.0
3	14-Jan-09	367.0	1.111035e+09	4.0	3.0	3.0	14.00	3.50	4.49	0.0	0.0	1.0
4	14-Jan-09	367.0	1.111038e+09	3.0	3.0	3.0	7.50	2.50	2.50	0.0	0.0	0.0

In [3]:
```
#check info by using the dataframe.info() method
""" Specifically, we printed out info about the number of columns, column labels,
column datatype, memory usage, range index, and the number of cells in each column
(non-null values)."""
transaction.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 682321 entries, 0 to 682320
Data columns (total 12 columns):
 #   Column         Non-Null Count   Dtype  
---  ------         --------------   -----  
 0   WEEK_END_DATE  524950 non-null  object 
 1   STORE_NUM      524950 non-null  float64
 2   UPC            524950 non-null  float64
 3   UNITS          524950 non-null  float64
 4   VISITS         524950 non-null  float64
 5   HHS            524950 non-null  float64
 6   SPEND          524950 non-null  float64
 7   PRICE          524927 non-null  float64
 8   BASE_PRICE     524765 non-null  float64
 9   FEATURE        524950 non-null  float64
 10  DISPLAY        524950 non-null  float64
 11  TPR_ONLY       524950 non-null  float64
dtypes: float64(11), object(1)
memory usage: 62.5+ MB
```

Store file

The store file provides detailed store information, including the city where the store is located, the city's stage, store parking space quantity etc.

In [4]:
```
#read store CSV file into dataframe using Pandas
store=pd.read_csv('dunnhumby_store.csv')
store.head()
```

Out[4]:

	STORE_ID	STORE_NAME	ADDRESS_CITY_NAME	ADDRESS_STATE_PROV_CODE	MSA_CODE	SEG_VALUE_NAME	PARKING_SPACE_QTY	SAI
0	389	SILVERLAKE	ERLANGER	KY	17140	MAINSTREAM	408.0	
1	2277	ANDERSON TOWNE CTR	CINCINNATI	OH	17140	UPSCALE	NaN	
2	4259	WARSAW AVENUE	CINCINNATI	OH	17140	VALUE	NaN	
3	6379	KINGWOOD	KINGWOOD	TX	26420	MAINSTREAM	NaN	
4	6431	AT WARD ROAD	BAYTOWN	TX	26420	VALUE	350.0	

In [5]:
```
#check info by using the dataframe.info() method
store.info() #we can see that last column has null values so we have to drop it
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 79 entries, 0 to 78
Data columns (total 10 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   STORE_ID                79 non-null     int64  
 1   STORE_NAME              79 non-null     object 
 2   ADDRESS_CITY_NAME       79 non-null     object 
 3   ADDRESS_STATE_PROV_CODE  79 non-null     object 
 4   MSA_CODE                79 non-null     int64  
 5   SEG_VALUE_NAME          79 non-null     object 
 6   PARKING_SPACE_QTY       27 non-null     float64
 7   SALES_AREA_SIZE_NUM     79 non-null     int64  
 8   AVG_WEEKLY_BASKETS      79 non-null     int64  
 9   Unnamed: 9              0 non-null      float64
dtypes: float64(2), int64(4), object(4)
memory usage: 6.3+ KB
```

The last column, "Unamed:9", contains all null values. This is because the store CSV file had an empty column. Thus, we dropped it.

In [6]:
```
# we used iloc[:, :-1] to select all rows and all columns except the last column
store = store.iloc[: , :-1]
store.head()
```

Out[6]:

	STORE_ID	STORE_NAME	ADDRESS_CITY_NAME	ADDRESS_STATE_PROV_CODE	MSA_CODE	SEG_VALUE_NAME	PARKING_SPACE_QTY	SAI
0	389	SILVERLAKE	ERLANGER	KY	17140	MAINSTREAM	408.0	
1	2277	ANDERSON TOWNE CTR	CINCINNATI	OH	17140	UPSCALE	NaN	
2	4259	WARSAW AVENUE	CINCINNATI	OH	17140	VALUE	NaN	
3	6379	KINGWOOD	KINGWOOD	TX	26420	MAINSTREAM	NaN	
4	6431	AT WARD ROAD	BAYTOWN	TX	26420	VALUE	350.0	

Product file

The product file provides detailed product information for each "UPC" in the transaction data, including the product's manufacturer, product category, subcategory and product size.

In [7]:
```
#read products CSV file into dataframe using Pandas
products=pd.read_csv('dunnhumby_products.csv')
products.head()
```

Out[7]:

	UPC	DESCRIPTION	MANUFACTURER	CATEGORY	SUB_CATEGORY	PRODUCT_SIZE
0	1.111009e+09	PL MINI TWIST PRETZELS	PRIVATE LABEL	BAG SNACKS	PRETZELS	15 OZ
1	1.111009e+09	PL PRETZEL STICKS	PRIVATE LABEL	BAG SNACKS	PRETZELS	15 OZ
2	1.111010e+09	PL TWIST PRETZELS	PRIVATE LABEL	BAG SNACKS	PRETZELS	15 OZ
3	1.111035e+09	PL BL MINT ANTSPTC RINSE	PRIVATE LABEL	ORAL HYGIENE PRODUCTS	MOUTHWASHES (ANTISEPTIC)	1.5 LT
4	1.111038e+09	PL BL MINT ANTSPTC RINSE	PRIVATE LABEL	ORAL HYGIENE PRODUCTS	MOUTHWASHES (ANTISEPTIC)	500 ML

In [8]:
```
#check info by using the dataframe.info() method
products.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 74 entries, 0 to 73
Data columns (total 6 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   UPC           58 non-null     float64
 1   DESCRIPTION   58 non-null     object 
 2   MANUFACTURER  58 non-null     object 
 3   CATEGORY      58 non-null     object 
 4   SUB_CATEGORY  58 non-null     object 
 5   PRODUCT_SIZE  58 non-null     object 
dtypes: float64(1), object(5)
memory usage: 3.6+ KB
```

Firstly, we merged the transaction data and product data by performing a left join using UPC.

In [9]:
```
#merge the data-left join by UPC
"""A left join combines two data frames by matching the values of the key
  columns in each data frame and retaining all the data from the left data
    frame and any matching data from the right data frame."""

df=pd.merge(transaction,products, on='UPC', how='left')
df.head()
```

Out[9]:

	WEEK_END_DATE	STORE_NUM	UPC	UNITS	VISITS	HHS	SPEND	PRICE	BASE_PRICE	FEATURE	DISPLAY	TPR_ONLY	DESCRIP
0	14-Jan-09	367.0	1.111009e+09	13.0	13.0	13.0	18.07	1.39	1.57	0.0	0.0	1.0	PL MINI T PRET
1	14-Jan-09	367.0	1.111009e+09	20.0	18.0	18.0	27.80	1.39	1.39	0.0	0.0	0.0	PL PRE ST
2	14-Jan-09	367.0	1.111010e+09	14.0	14.0	14.0	19.32	1.38	1.38	0.0	0.0	0.0	PL T PRET
3	14-Jan-09	367.0	1.111035e+09	4.0	3.0	3.0	14.00	3.50	4.49	0.0	0.0	1.0	PL BL ANT F
4	14-Jan-09	367.0	1.111038e+09	3.0	3.0	3.0	7.50	2.50	2.50	0.0	0.0	0.0	PL BL ANT F

Then, we merged the new dataset (transaction + products) with the store data by performing a left join again STORE_ID.

In [10]:
```
#merge transactions and products to stores
"""
It can be seen that the joining variable is named differently in the two
datasets ('STORE_NUM' and 'STORE_ID').
Thus, the merging is achieved by using the left_on and right_on arguments
to the pandas' merge function. """

df1=pd.merge(df,store, left_on='STORE_NUM', right_on='STORE_ID', how='left')
```

In [11]:
```
#store number of rows and columns as a tuple to see the dimensionality of our merge dataset before cleaning.
df1.shape
```

Out[11]:
```
(3056579, 26)
```

In [12]:
```
#check preview of the final merged dataset
"""The final preview of the data (df1) shows all the related attributes
of a product at a particular store, with the rows representing the observations
and the columns representing the variables."""

df1.head()
```

Out[12]:

	WEEK_END_DATE	STORE_NUM	UPC	UNITS	VISITS	HHS	SPEND	PRICE	BASE_PRICE	FEATURE	...	PRODUCT_SIZE	STORE_ID
0	14-Jan-09	367.0	1.111009e+09	13.0	13.0	13.0	18.07	1.39	1.57	0.0	...	15 OZ	367.0
1	14-Jan-09	367.0	1.111009e+09	20.0	18.0	18.0	27.80	1.39	1.39	0.0	...	15 OZ	367.0
2	14-Jan-09	367.0	1.111010e+09	14.0	14.0	14.0	19.32	1.38	1.38	0.0	...	15 OZ	367.0
3	14-Jan-09	367.0	1.111035e+09	4.0	3.0	3.0	14.00	3.50	4.49	0.0	...	1.5 LT	367.0
4	14-Jan-09	367.0	1.111038e+09	3.0	3.0	3.0	7.50	2.50	2.50	0.0	...	500 ML	367.0

5 rows × 26 columns

2.3 Data Filtering

Given that our analysis aims to investigate ways to increase sales, measured by "SPEND", in our study, we only included variables with economic meanings.

In [13]:
```
#Filter data based on its relevance
#include only variables which can potentially affect "SPEND"
df1= df1.filter(items = ["PRICE","BASE_PRICE","SPEND","UNITS",
                         "TPR_ONLY","FEATURE","DISPLAY","WEEK_END_DATE",
                         "CATEGORY","UPC"])
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3056579 entries, 0 to 3056578
Data columns (total 10 columns):
 #   Column         Dtype  
---  ------         -----  
 0   PRICE          float64
 1   BASE_PRICE     float64
 2   SPEND          float64
 3   UNITS          float64
 4   TPR_ONLY       float64
 5   FEATURE        float64
 6   DISPLAY        float64
 7   WEEK_END_DATE  object 
 8   CATEGORY       object 
 9   UPC            float64
dtypes: float64(8), object(2)
memory usage: 256.5+ MB
```

2.4 Data cleaning

Duplicates

Duplicate data may skew the analysis of product sales per category. Thus, we dropped it by using Pandas' drop_duplicated method.

In [14]:
```
#return the number of duplicate rows in the DataFrame df1.
df1.duplicated().sum()
#remove duplicate rows from the DataFrame df1 and modify the DataFrame in place.
df1.drop_duplicates(inplace=True)
```

Missing values

In [15]:
```
"""return the number of missing values in each column of the DataFrame df1
    using pandas isnull().sum()"""

df1.isnull().sum()
```

Out[15]:
```
PRICE             24
BASE_PRICE       184
SPEND              1
UNITS              1
TPR_ONLY           1
FEATURE            1
DISPLAY            1
WEEK_END_DATE      1
CATEGORY           1
UPC                1
dtype: int64
```

The output shows that there are:

- 24 values missing values for 'Price',
- 184 for 'Base_Price',
- 1 missing value for each of the remaining variables.

Below, it can be observed that the last row of the dataset has missing values for all variables, which explains why several attributes are missing one value. Thus, this row is removed.

In [16]:
```
#here I see that the last row has missing values
#this explains why we have several attributes with 1 missing values.
df1.iloc[-1]#return the last row of the DataFrame df1
```

Out[16]:
```
PRICE            NaN
BASE_PRICE       NaN
SPEND            NaN
UNITS            NaN
TPR_ONLY         NaN
FEATURE          NaN
DISPLAY          NaN
WEEK_END_DATE    NaN
CATEGORY         NaN
UPC              NaN
Name: 538643, dtype: object
```

In [17]:
```
#we used iloc[:, :-1] to select all rows and all columns except the last row
df1=df1.iloc[:-1, :]
```

In [18]:
```
#check missing values again
df1.isnull().sum()
```

Out[18]:
```
PRICE             23
BASE_PRICE       183
SPEND              0
UNITS              0
TPR_ONLY           0
FEATURE            0
DISPLAY            0
WEEK_END_DATE      0
CATEGORY           0
UPC                0
dtype: int64
```

After exploring the data, we noticed that the 23 missing values for the price should = the base price because no promotion has been applied. This is because the difference between BASE PRICE and PRICE comes from whether a promotion technique has been applied. Thus, we used pandas' fillna' function to fill the 23 missing values for 'Price' with the value of the respective 'Base_Price'.

In [19]:
```
#since none of the promotions types (feature,display and TPR_only) are applied,
#the 22 missing values for price are =base price.
df1['PRICE'].fillna(df1['BASE_PRICE'],inplace=True)
print(df1)
```

```
        PRICE  BASE_PRICE   SPEND  UNITS  TPR_ONLY  FEATURE  DISPLAY  \
0        1.39        1.57   18.07   13.0       1.0      0.0      0.0
1        1.39        1.39   27.80   20.0       0.0      0.0      0.0
2        1.38        1.38   19.32   14.0       0.0      0.0      0.0
3        3.50        4.49   14.00    4.0       1.0      0.0      0.0
4        2.50        2.50    7.50    3.0       0.0      0.0      0.0
...       ...         ...     ...    ...       ...      ...      ...
538632   5.00        6.99   30.00    6.0       0.0      1.0      1.0
538634   2.50        2.50   40.00   16.0       0.0      0.0      0.0
538640   3.31        3.31  105.92   32.0       0.0      0.0      0.0
538641   3.31        3.31   99.30   30.0       0.0      0.0      0.0
538642   2.22        2.85   44.40   20.0       1.0      0.0      0.0

       WEEK_END_DATE              CATEGORY           UPC
0          14-Jan-09            BAG SNACKS  1.111009e+09
1          14-Jan-09            BAG SNACKS  1.111009e+09
2          14-Jan-09            BAG SNACKS  1.111010e+09
3          14-Jan-09  ORAL HYGIENE PRODUCTS  1.111035e+09
4          14-Jan-09  ORAL HYGIENE PRODUCTS  1.111038e+09
...              ...                   ...           ...
538632     04-Jan-12           FROZEN PIZZA  7.218064e+09
538634     04-Jan-12            BAG SNACKS  7.797502e+09
538640     04-Jan-12           COLD CEREAL  8.849120e+10
538641     04-Jan-12           COLD CEREAL  8.849120e+10
538642     04-Jan-12           COLD CEREAL  8.849121e+10

[353685 rows x 10 columns]
```

In [20]:
```
#check missing values again
df1.isnull().sum()
```

Out[20]:
```
PRICE              0
BASE_PRICE       183
SPEND              0
UNITS              0
TPR_ONLY           0
FEATURE            0
DISPLAY            0
WEEK_END_DATE      0
CATEGORY           0
UPC                0
dtype: int64
```

In [21]:
```
# Calculate the percentage of missing values of BASE_PRICE
100 * df1["BASE_PRICE"].isnull().sum() / len(df1)
```

Out[21]:
```
0.05174095593536621
```

Imputing the mean or median value for missing data can lead to a biased dataset when missing values are not missing at random. Thus, given that the missing values for BASE_PRICE account only for 0.0517% of the total, we decided to drop them.

In [22]:
```
#drop na values for base price, cause data is missing.
df1=df1.dropna()#remove rows or columns containing null values.
```

In [23]:
```
#check missing values again
df1.isnull().sum()
```

Out[23]:
```
PRICE            0
BASE_PRICE       0
SPEND            0
UNITS            0
TPR_ONLY         0
FEATURE          0
DISPLAY          0
WEEK_END_DATE    0
CATEGORY         0
UPC              0
dtype: int64
```

2.5 Data encoding

In [24]:
```
""" 'FEATURE', 'DISPLAY' and 'TPR_ONLY' are binary variables to indicate
    if a promotion technique was being applied.
    They refer to if product was in a in-store circular, part of in-store
    promotional display or temporary price reduction only respectively."""

df1['FEATURE'] = df1['FEATURE'].astype(int)
df1['DISPLAY'] = df1['DISPLAY'].astype(int)
df1['TPR_ONLY'] = df1['TPR_ONLY'].astype(int)
```

2.6 Feature Engineering

In [25]:
```
"""Feature Engineering, added new variable 'Discount'.
'PROMOTION' variable indicates if any of the promotion technique
was being applied, hence whether was there a promotion."""
```

```
df1['PROMOTION'] = df1.FEATURE | df1.DISPLAY | df1.TPR_ONLY
```

In [26]:
```
#check final dataset
df1.head()
```

Out[26]:

	PRICE	BASE_PRICE	SPEND	UNITS	TPR_ONLY	FEATURE	DISPLAY	WEEK_END_DATE	CATEGORY	UPC	PROMOTION
0	1.39	1.57	18.07	13.0	1	0	0	14-Jan-09	BAG SNACKS	1.111009e+09	1
1	1.39	1.39	27.80	20.0	0	0	0	14-Jan-09	BAG SNACKS	1.111009e+09	0
2	1.38	1.38	19.32	14.0	0	0	0	14-Jan-09	BAG SNACKS	1.111010e+09	0
3	3.50	4.49	14.00	4.0	1	0	0	14-Jan-09	ORAL HYGIENE PRODUCTS	1.111035e+09	1
4	2.50	2.50	7.50	3.0	0	0	0	14-Jan-09	ORAL HYGIENE PRODUCTS	1.111038e+09	0

In [27]:
```
#check info
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 353502 entries, 0 to 538642
Data columns (total 11 columns):
 #   Column         Non-Null Count   Dtype  
---  ------         --------------   -----  
 0   PRICE          353502 non-null  float64
 1   BASE_PRICE     353502 non-null  float64
 2   SPEND          353502 non-null  float64
 3   UNITS          353502 non-null  float64
 4   TPR_ONLY       353502 non-null  int64  
 5   FEATURE        353502 non-null  int64  
 6   DISPLAY        353502 non-null  int64  
 7   WEEK_END_DATE  353502 non-null  object 
 8   CATEGORY       353502 non-null  object 
 9   UPC            353502 non-null  float64
 10  PROMOTION      353502 non-null  int64  
dtypes: float64(5), int64(4), object(2)
memory usage: 32.4+ MB
```

In [28]:
```
#checking rows and column of final dataset
df1.shape
```

Out[28]:
```
(353502, 11)
```

The final dataset has 353502 observations and 11 attributes, including the target variable "SPEND".

2.7 Outliers

By looking at the summary statistics, we identified two potential outliers:

- SPEND
- UNITS.

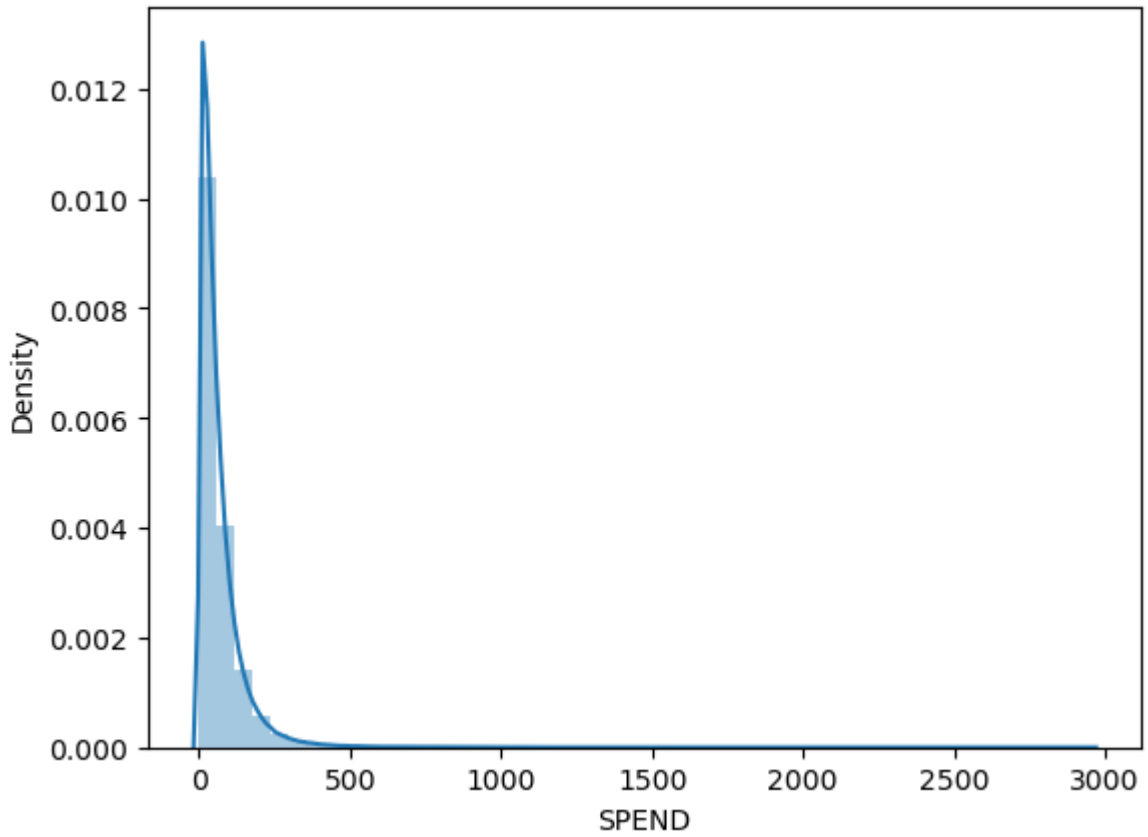
Their maximum values are much bigger than the mean values.

In [29]:
```
#we used the describe method to generate a summary statistics for df1
df1.describe()
```

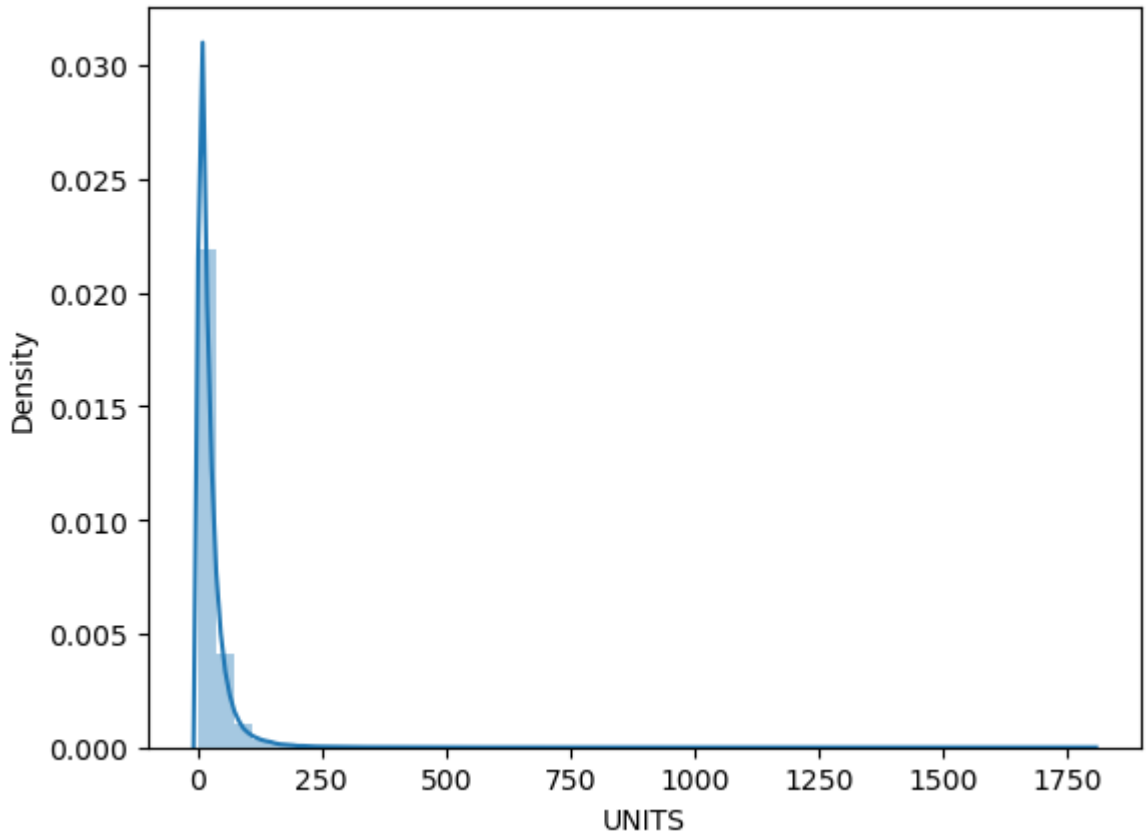
Out[29]:

	PRICE	BASE_PRICE	SPEND	UNITS	TPR_ONLY	FEATURE	DISPLAY	UPC	PRO
count	353502.000000	353502.000000	353502.000000	353502.000000	353502.000000	353502.000000	353502.000000	3.535020e+05	35350
mean	3.050438	3.321916	66.535658	25.649244	0.158737	0.112401	0.152293	1.011671e+10	
std	1.421908	1.556524	77.146459	34.365538	0.365432	0.315860	0.359306	2.182882e+10	
min	0.000000	0.550000	0.000000	0.000000	0.000000	0.000000	0.000000	1.111009e+09	
25%	1.960000	2.320000	20.320000	7.000000	0.000000	0.000000	0.000000	1.111087e+09	
50%	2.890000	2.990000	43.900000	16.000000	0.000000	0.000000	0.000000	3.000006e+09	
75%	3.590000	3.990000	84.830000	31.000000	0.000000	0.000000	0.000000	7.192100e+09	
max	11.460000	11.460000	2952.000000	1800.000000	1.000000	1.000000	1.000000	8.849121e+10	

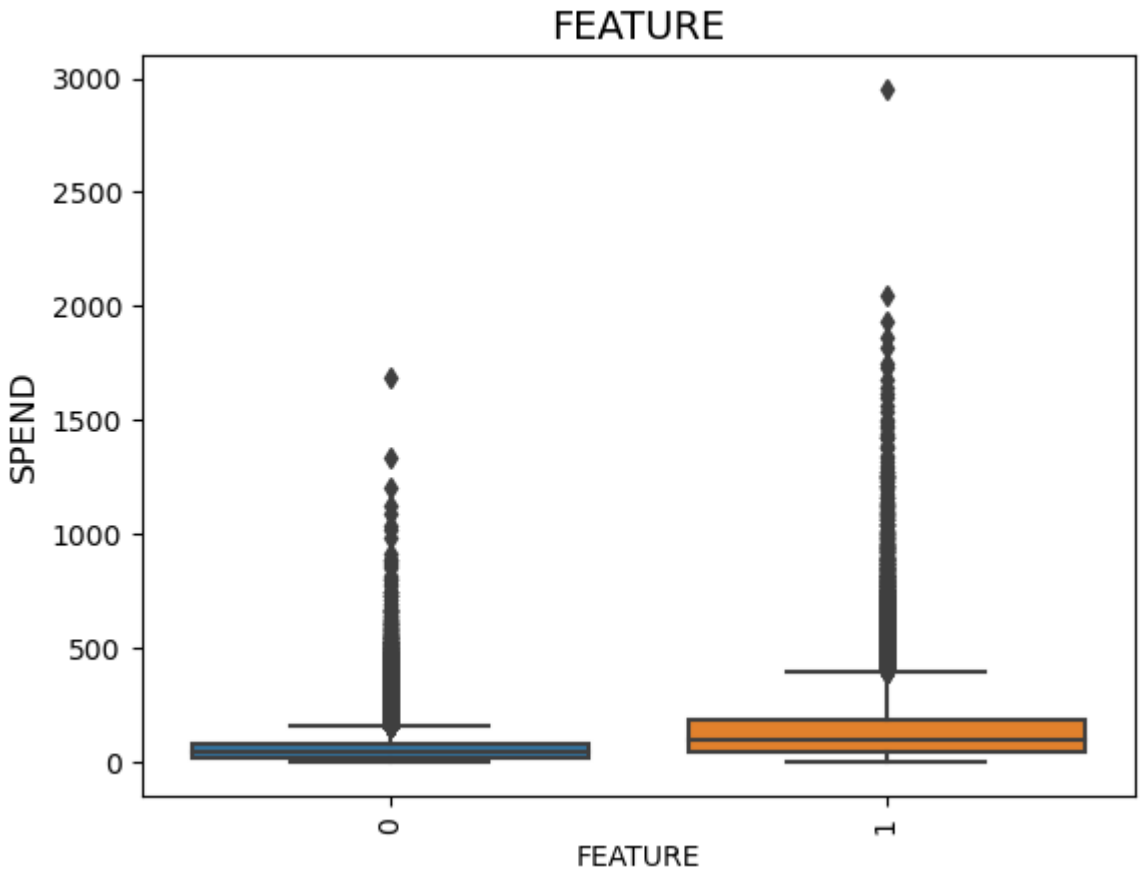
In [30]:
```
"""we used the distplot() function from the seaborn library to
    plot the distribution of SPEND"""

sns.distplot(df1['SPEND']);# DISTRIBUTION OF SPEND
```

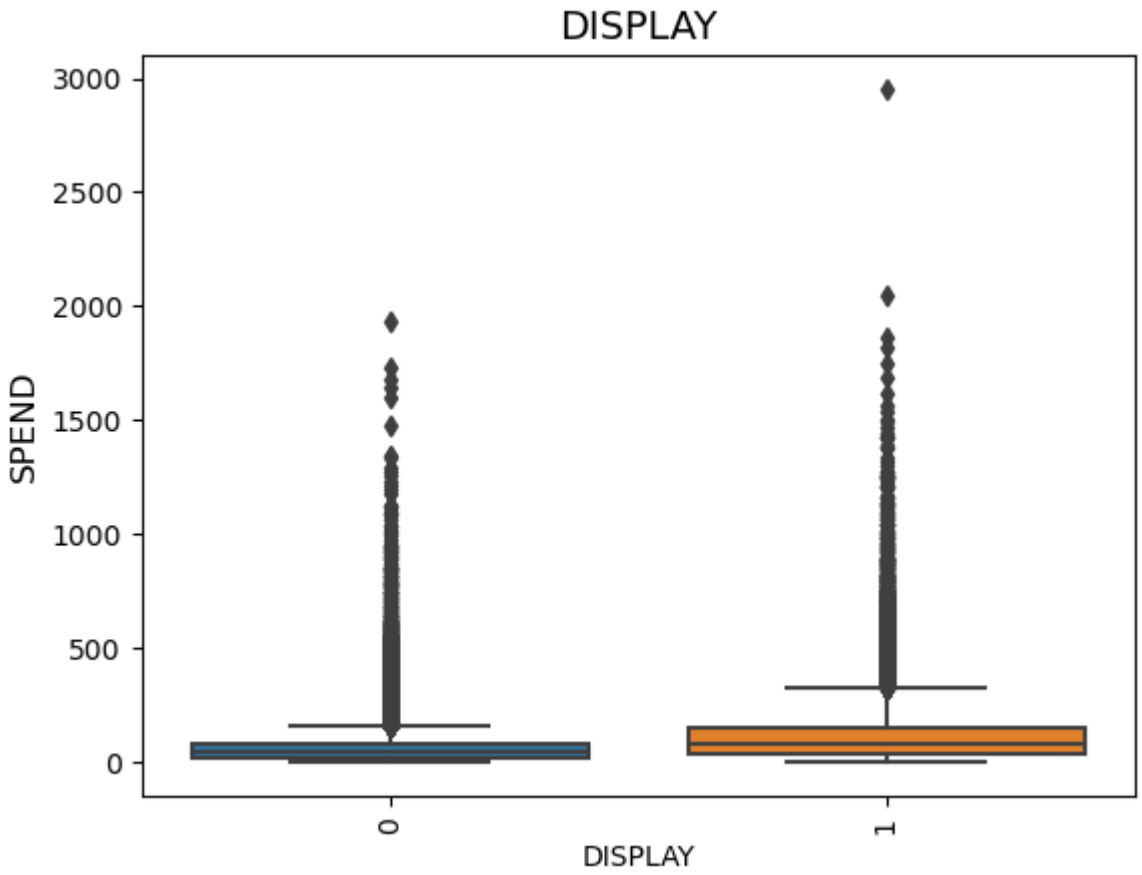


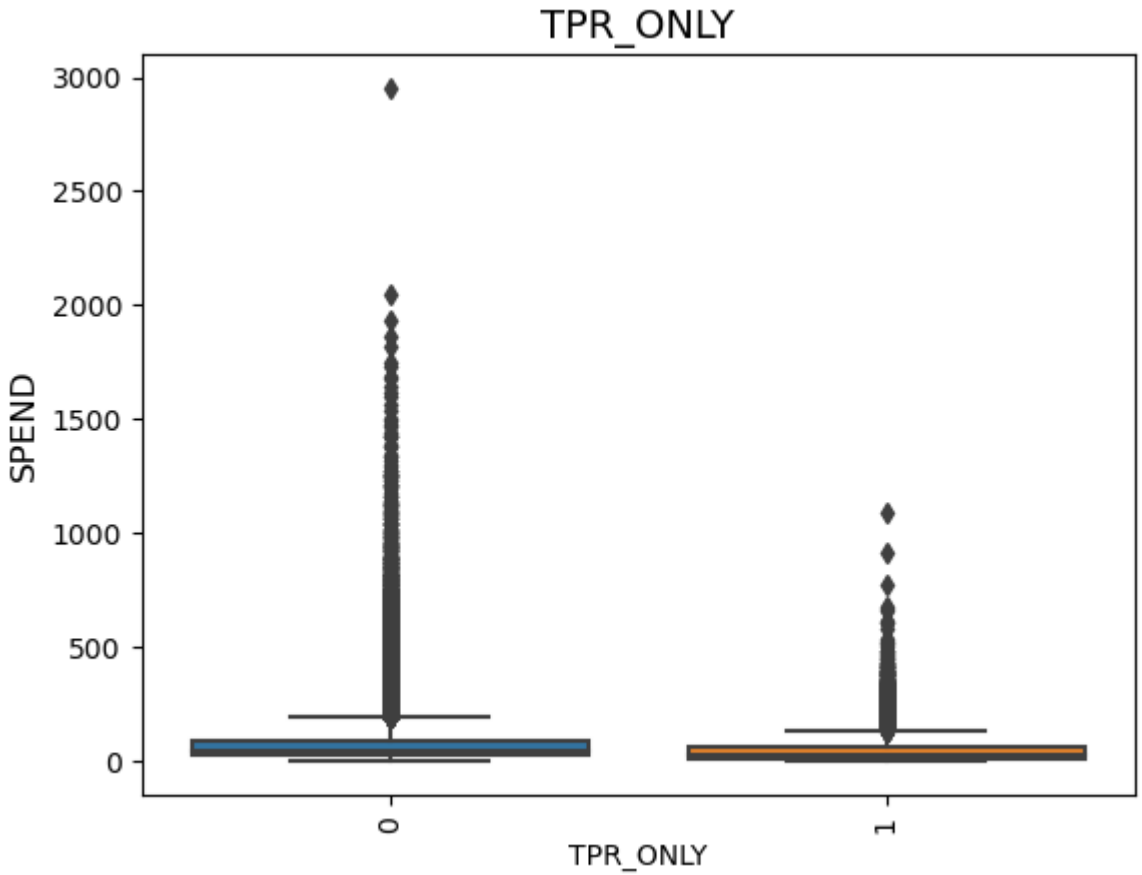
In [31]:
```
"""we used the distplot() function from the seaborn library to plot the
    distribution of UNITS"""

sns.distplot(df1['UNITS']);
```

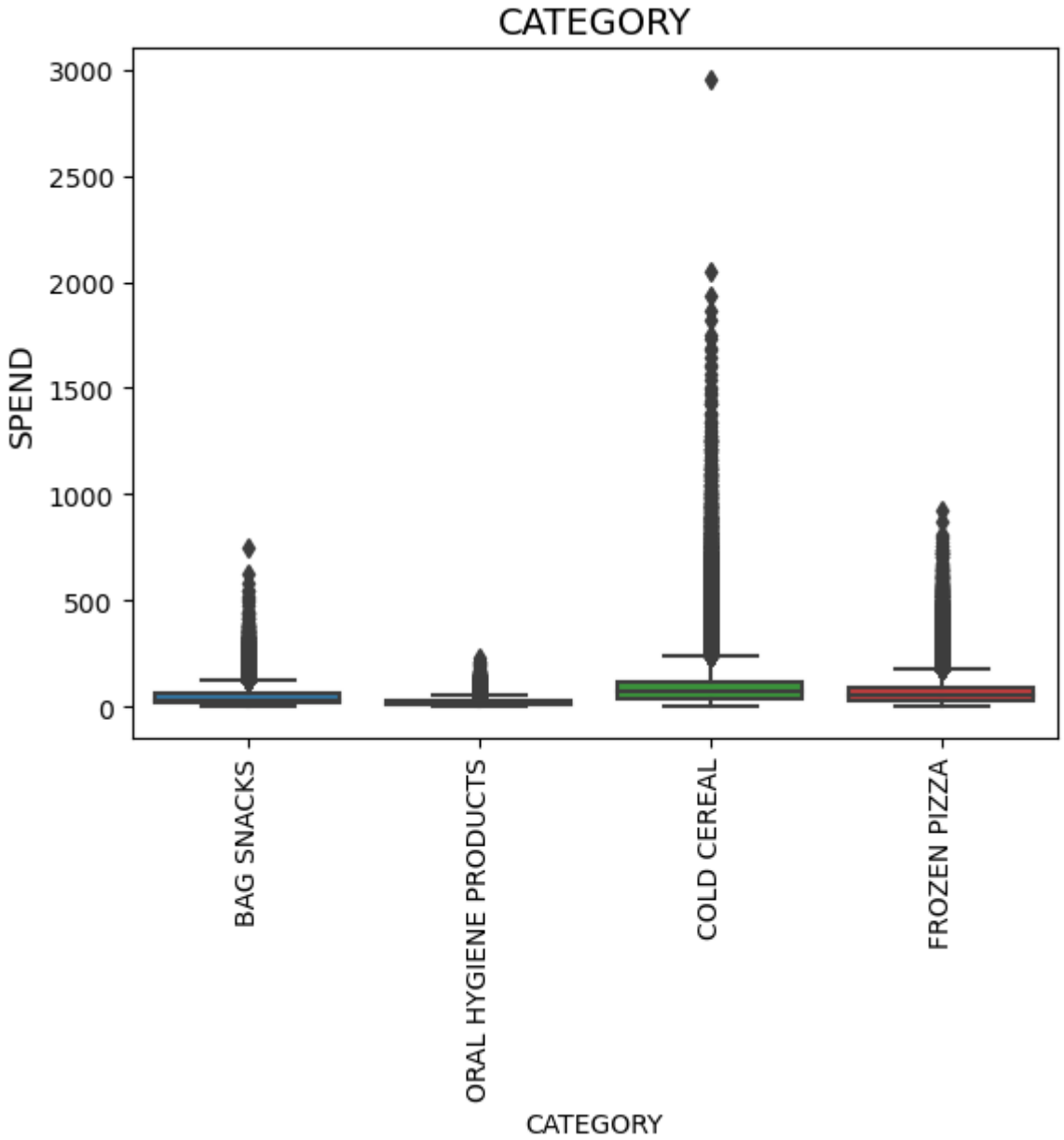


In [32]:
```
#outliers for categorical features
columns = ['FEATURE', 'DISPLAY','TPR_ONLY','CATEGORY']
for col in columns:
    sns.boxplot(x=df1[col], y=df1.SPEND)
    plt.ylabel("SPEND", fontsize=12)
    plt.xticks(rotation='vertical')
    plt.title(col, fontsize=14)
    plt.show()
```


FEATURE
3000
2500
2000
1500
1000
500
0
SPEND
0
1
FEATURE


DISPLAY
3000
2500
2000
1500
1000
500
0
SPEND
0
1
DISPLAY


TPR_ONLY
3000
2500
2000
1500
1000
500
0
SPEND
0
1
TPR_ONLY



When checking outliers for categorical categories, we noticed that a customer has spent 3,000 pounds on cold cereal in the cereal category. Outliers such as this indicate the importance of keeping the outliers within our dataset, as they provide valuable information that is part of our study.

3 Data Visualisations

To explore the data, we create data visualizations which are split into the following sub-sections:

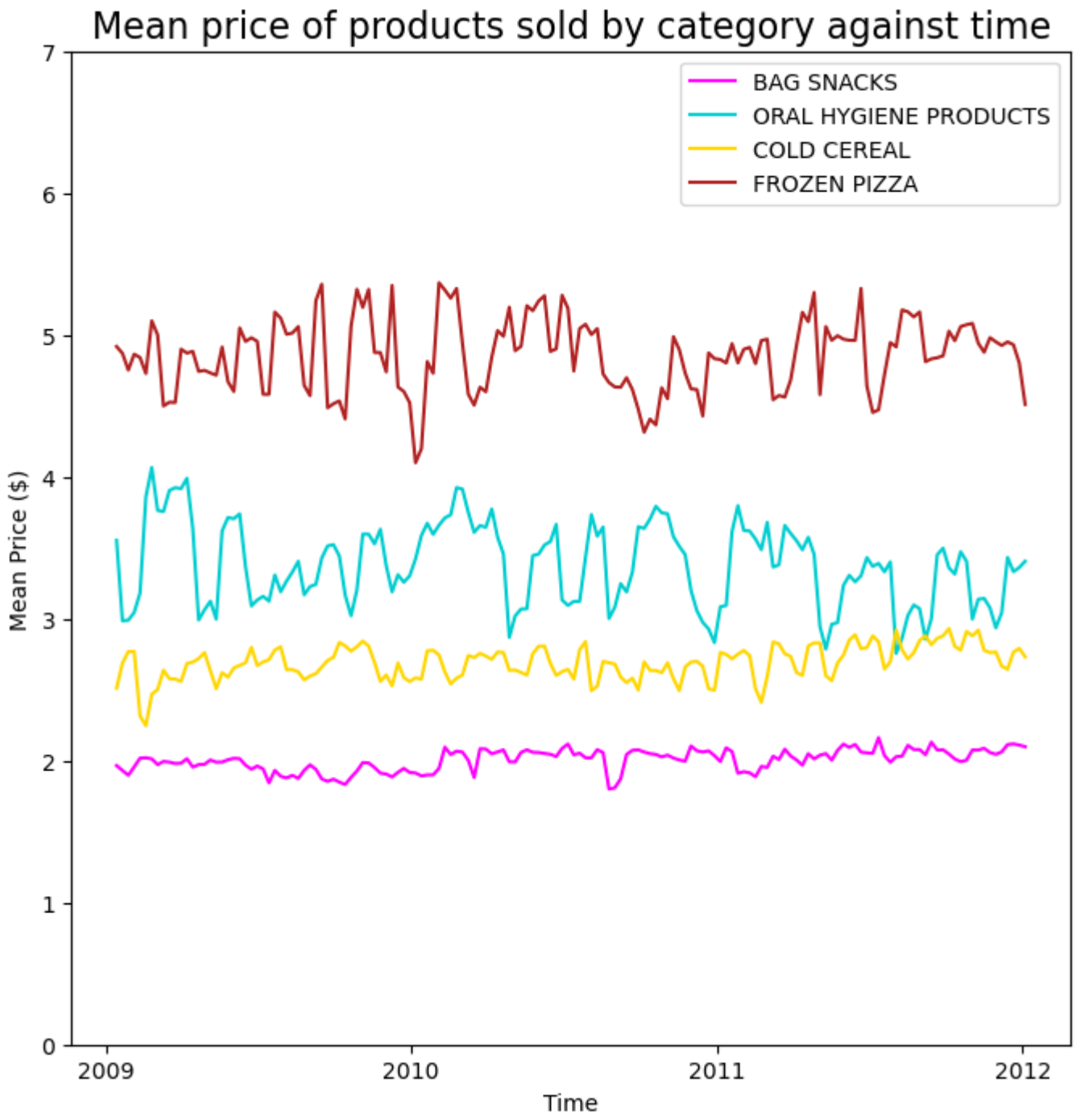
- Examining Price
- Examining Units Sold,
- Examining the effects on Sales.

3.1 Examining Price

What is the mean price of the different items sold across time?

In [33]:
```
#Extracting the data for each category
"""Using groupby function to group the week_end_date and calculate price
    mean for each week."""
#Renaming columns from 'Price' to 'mean_price

BAG_SNACKS_UNITS = df1[df1.CATEGORY=='BAG SNACKS']\
                        .groupby('WEEK_END_DATE').PRICE.mean().reset_index()\
                        .rename(columns={'PRICE':'mean_price_S'})
ORAL_HYGIENE_UNITS= df1[df1.CATEGORY=='ORAL HYGIENE PRODUCTS']\
                        .groupby('WEEK_END_DATE').PRICE.mean().reset_index()\
                        .rename(columns={'PRICE':'mean_price_O'})
COLD_CEREAL_UNITS = df1[df1.CATEGORY=='COLD CEREAL']\
                        .groupby('WEEK_END_DATE').PRICE.mean().reset_index()\
                        .rename(columns={'PRICE':'mean_price_C'})
FROZEN_PIZZA_UNITS = df1[df1.CATEGORY=='FROZEN PIZZA']\
                        .groupby('WEEK_END_DATE').PRICE.mean().reset_index()\
                        .rename(columns={'PRICE':'mean_price_P'})
#T_a as average price
""" Merging Oral_Hygiene_Units on the week_end_date with a left merge to
   create T_P dataframe """

T_a= BAG_SNACKS_UNITS\
                    .merge(ORAL_HYGIENE_UNITS, on='WEEK_END_DATE', how='left')\
                    .merge(COLD_CEREAL_UNITS, on='WEEK_END_DATE', how='left')\
                    .merge(FROZEN_PIZZA_UNITS, on='WEEK_END_DATE', how='left')
```

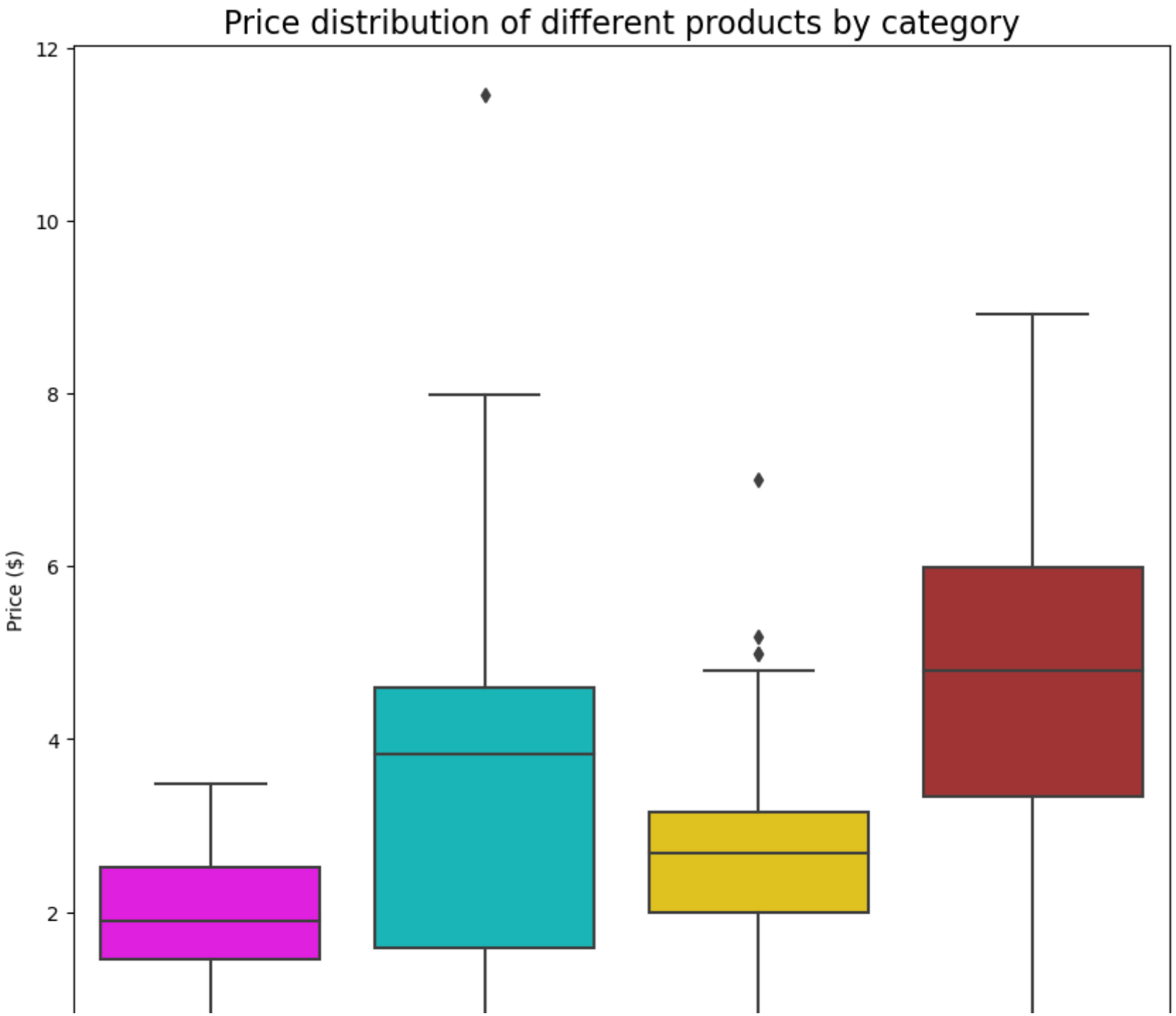
```
#Creating a date time series to plot the data
T_a['WEEK_END_DATE'] = pd.to_datetime(T_a['WEEK_END_DATE'])
#Sorting values by week_end_date
T_a.sort_values(by='WEEK_END_DATE',inplace=True)
T_a['WEEK_END_DATE'] =T_a['WEEK_END_DATE']

#Setting figure size (8,8)
fig=plt.figure(figsize=(8,8))

#Plotting mean price for each category (price as y axis, time as x axis)
plt.plot('WEEK_END_DATE', 'mean_price_S', data=T_a,color="magenta")
plt.plot('WEEK_END_DATE', 'mean_price_O', data=T_a, color="darkturquoise")
plt.plot('WEEK_END_DATE', 'mean_price_C', data=T_a, color="gold")
plt.plot('WEEK_END_DATE', 'mean_price_P', data=T_a, color="firebrick")

#Setting the color legend for the figure
plt.legend(['BAG SNACKS', 'ORAL HYGIENE PRODUCTS','COLD CEREAL','FROZEN PIZZA'])

#Setting title and labels for the graph
plt.title('Mean price of products sold by category against time', fontsize=16)
plt.xlabel('Time')
plt.ylabel("Mean Price ($)")

#Setting the x ticks and x axis labels as the years 2009, 2010, 2011 and 2012
plt.xticks(ticks=[datetime.datetime(2009,1,1),datetime.datetime(2010,1,1),
                  datetime.datetime(2011,1,1),datetime.datetime(2012,1,1)],
           labels=['2009','2010','2011','2012'])

plt.ylim(0, 7) #setting the y axis limit as 0 to the maximum mean price (7)
plt.show()
```


Mean price of products sold by category against time
BAG SNACKS
ORAL HYGIENE PRODUCTS
COLD CEREAL
FROZEN PIZZA
Mean Price ($)
0
1
2
3
4
5
6
7
2009
2010
2011
2012
Time

As observed by the figure above, the category with the highest mean is frozen pizza, and the second highest is oral hygiene products.

What is the price distribution of different products by category?

In [34]:
```
#create color array
my_colors = ["magenta", "darkturquoise", "gold", "firebrick"]

#add color array to set_palette
sns.set_palette( my_colors )

#Plot figure size
plt.figure(figsize=(10,10))

#Create boxplot with x axis as 'CATEGORY' and y axis as 'PRICE'
sns.boxplot(x="CATEGORY", y="PRICE", data=df1)

#Set labels for x and y axis
plt.ylabel("Price ($)")
plt.xlabel("Category")

#Set title for graph
plt.title('Price distribution of different products by category', fontsize=16);
```


Price distribution of different products by category
Price ($)
12
10
8
6
4
2

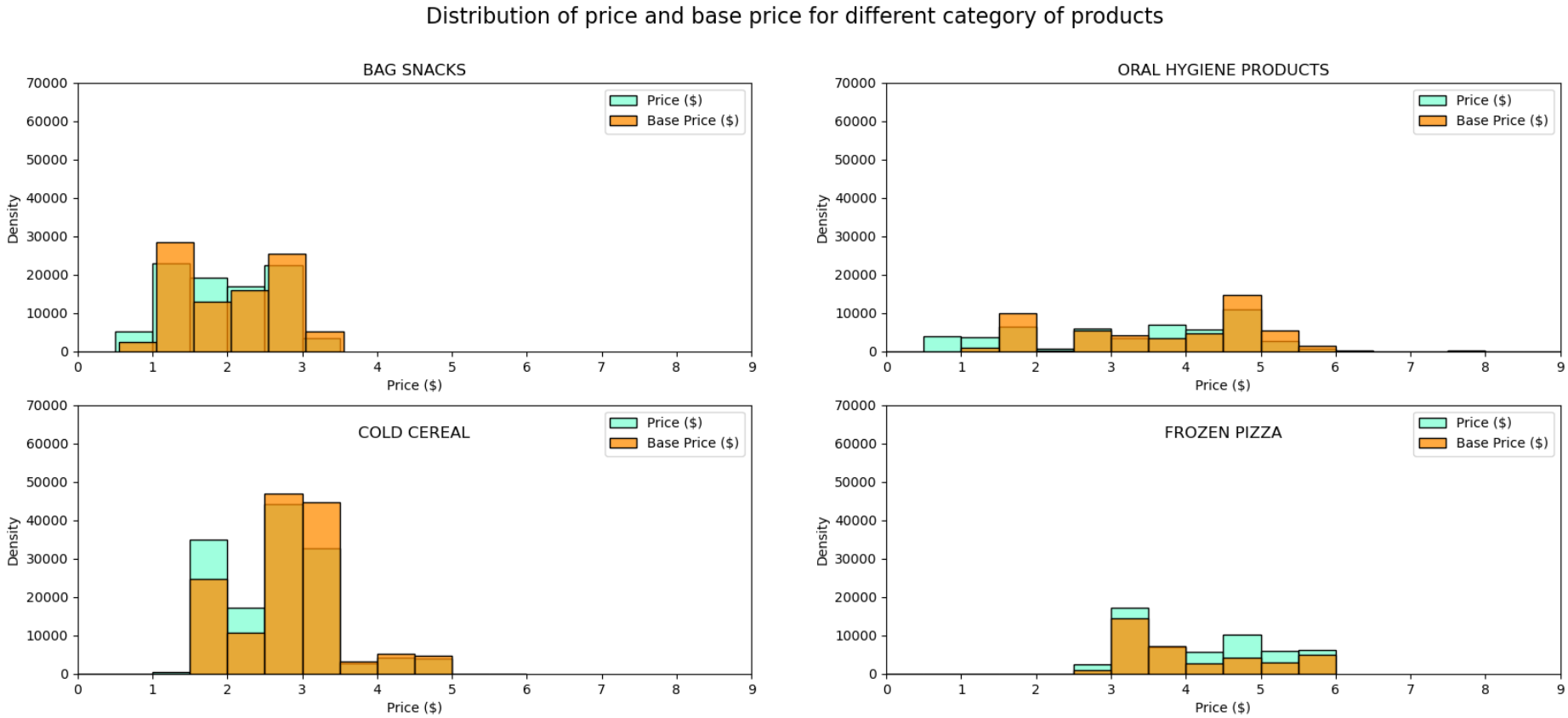
The boxplot above shows the price distribution of each category. 'Oral Hygiene Products' has the greatest range of prices while 'Bag Snacks' has the lowest range. 'Frozen Pizza' has the highest median. 'Cold Cereal' has the greatest number of outliers.

Do cheaper products within the same category recieve more promotion?

In [35]:
```
#create color array
my_colors = ["aquamarine", "darkorange"]

#add color array to set_palette
sns.set_palette( my_colors )

#set figure size
fig=plt.figure(figsize=(20,8))

"""Creating histograms to compare the distribution of base price and
   price for each category """
#plotting subplot
plt.subplot(2,2,1)

#setting title to subplot ('BAG SNACKS')
plt.title('BAG SNACKS')

#Setting category as 'BAG SNACKS' for subplot
df2=df1[df1.CATEGORY=='BAG SNACKS']

#kernal density estimate set to false to indicate that there is no curve
#histogram divided into .5 intervals
#Plotting price and base price for comparison
sns.histplot(df1[df1.CATEGORY=='BAG SNACKS']['PRICE'],
             kde=False,binwidth=.5, label='PRICE',color="aquamarine")
sns.histplot(df1[df1.CATEGORY=='BAG SNACKS']['BASE_PRICE'],
             kde=False,binwidth=.5, label='BASE_PRICE',color="darkorange")
plt.xlim((0,9)) #setting x axis limit from 0 to the highest price (9)
```

```
plt.ylim((0,70000)) #setting y axis limit from 0 to the highest density (70000)

#setting the x and y labels as 'Price ($)' and 'Density'
plt.xlabel('Price ($)')
plt.ylabel('Density')

#setting the legend for the graph
plt.legend(['Price ($)','Base Price ($)'],loc='upper right')

#Creating subplot for Oral Hygiene Products
plt.subplot(2,2,2)
plt.title('ORAL HYGIENE PRODUCTS')
sns.histplot(df1[df1.CATEGORY=='ORAL HYGIENE PRODUCTS']['PRICE'],
             kde=False,binwidth=.5, binrange=(0,9),label='PRICE',color="aquamarine")
sns.histplot(df1[df1.CATEGORY=='ORAL HYGIENE PRODUCTS']['BASE_PRICE'],
             kde=False,binwidth=.5, binrange=(0,9),label='BASE_PRICE',
             color="darkorange")
plt.xlim((0,9))
plt.ylim((0,70000))
plt.xlabel('Price ($)')
plt.ylabel('Density')
plt.legend(['Price ($)','Base Price ($)'],loc='upper right')

#Creating subplot for Cold Cereal Products
plt.subplot(2,2,3)
plt.title('COLD CEREAL', y=0.85)
sns.histplot(df1[df1.CATEGORY=='COLD CEREAL']['PRICE'],
             kde=False,binwidth=.5, binrange=(0,6),label='PRICE',color="aquamarine")
sns.histplot(df1[df1.CATEGORY=='COLD CEREAL']['BASE_PRICE'],
             kde=False,binwidth=.5, binrange=(0,6),label='BASE_PRICE',
             color="darkorange")
plt.xlim((0,9))
plt.ylim((0,70000))
plt.xlabel('Price ($)')
plt.ylabel('Density')
plt.legend(['Price ($)','Base Price ($)'],loc='upper right')

#Creating subplot for Frozen Pizza Products
plt.subplot(2,2,4)
plt.title('FROZEN PIZZA', y=0.85)
sns.histplot(df1[df1.CATEGORY=='FROZEN PIZZA']['PRICE'],
             kde=False,binwidth=.5, binrange=(0,6),label='PRICE',color="aquamarine")
sns.histplot(df1[df1.CATEGORY=='FROZEN PIZZA']['BASE_PRICE'],
             kde=False,binwidth=.5, binrange=(0,6),label='BASE_PRICE',
             color="darkorange")
```

```
plt.xlim((0,9))
plt.ylim((0,70000))
plt.xlabel('Price ($)')
plt.ylabel('Density')
plt.legend(['Price ($)','Base Price ($)'],loc='upper right')

#create overall title for the figure
fig.suptitle('Distribution of price and base price for different category of products',
             fontsize=16);
```

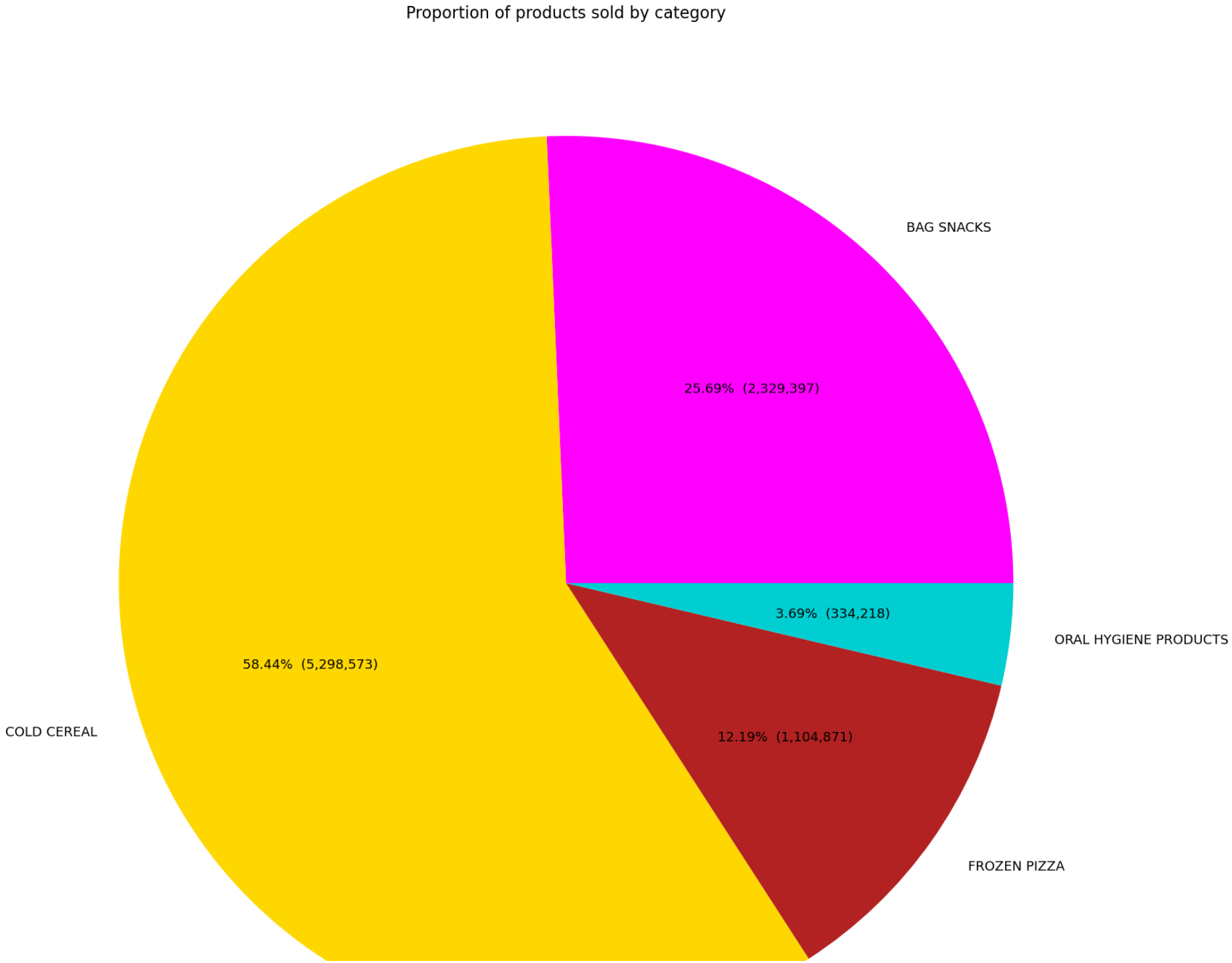


The figure above shows the distribution of price and base price for different categories of products. For cheaper products within the same category, promotions are applied more vigorously than the more expensive products. This is so for all categories.

3.2 Examining Units Sold

Which category has the highest proportion of sales?

In [36]:
```
#create color array
my_colors = ["magenta", "gold", "firebrick", "darkturquoise"]
#add color array to set_palette
sns.set_palette( my_colors )

"""Group by category, sum the total units in each category and
    reset index for the dataframe """
Units_sold_category = df1.groupby('CATEGORY').UNITS.sum().reset_index()

#plot figure size
plt.figure(figsize=(20, 20))

#calculating percentage of units sold for each category
plt.pie('UNITS', labels = 'CATEGORY',
        autopct=lambda p : '{:.2f}%  ({:,.0f})'.format
        (p,p * sum((Units_sold_category['UNITS']))/100) ,
        data=Units_sold_category, textprops={'fontsize': 13})

#Set title for graph
plt.title("Proportion of products sold by category", fontsize=16)
plt.show()
```

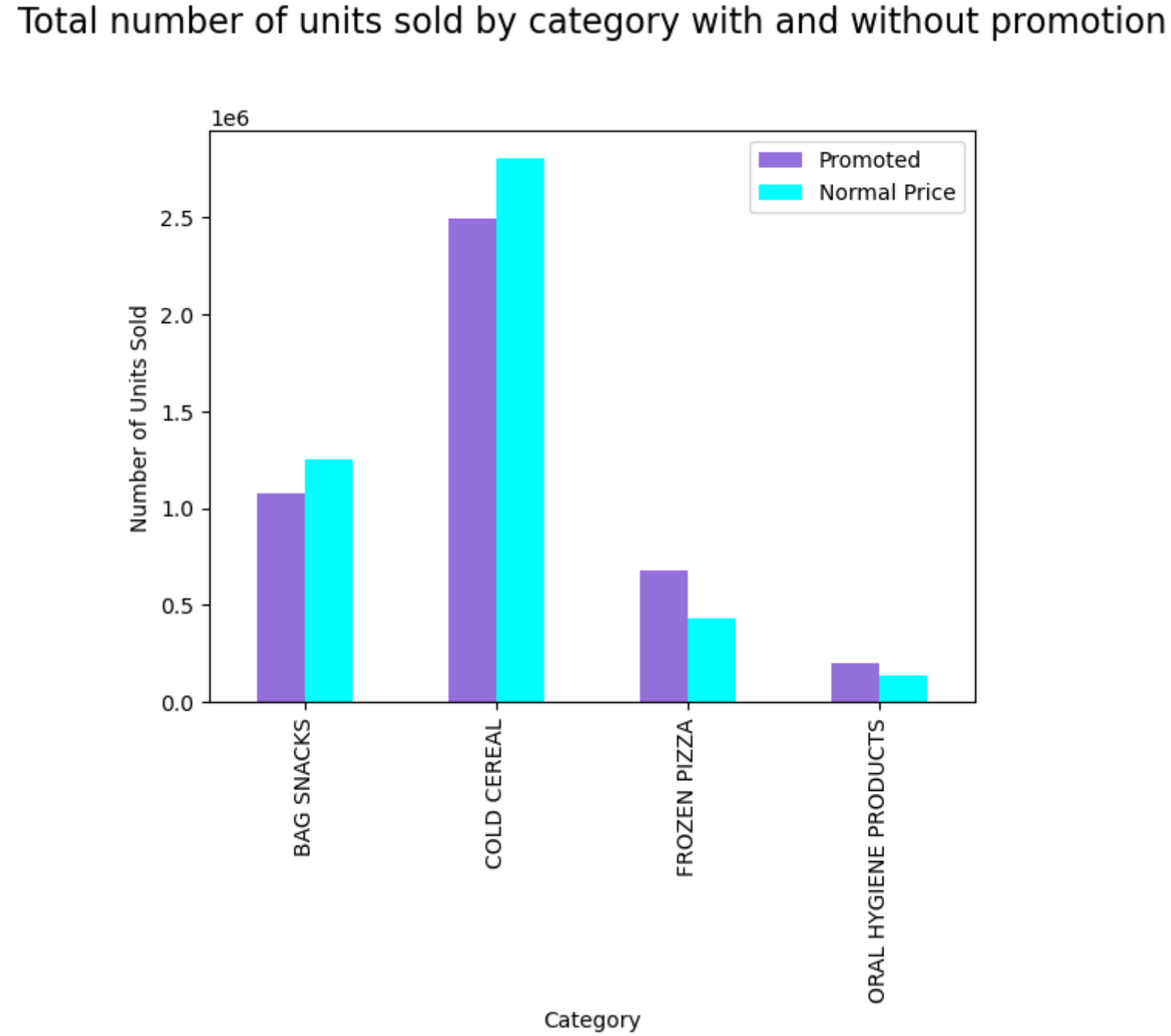

Proportion of products sold by category
BAG SNACKS
25.69% (2,329,397)
ORAL HYGIENE PRODUCTS
3.69% (334,218)
FROZEN PIZZA
12.19% (1,104,871)
COLD CEREAL
58.44% (5,298,573)

Here, the sum of all the items sold over a fixed period of time is found. The results show that cold cereal has the highest proportion of sales.

How does promotion impact the number of units sold?

In [37]:
```
#Group by category, sum the total units in each category and reset index
Total_Units = df1.groupby('CATEGORY').UNITS.sum().reset_index()

"""Filter items on discount, group by category, sum the total units
    in each category and reset index"""
#Rename the column 'UNITS' to 'UNITS_WITH_PROMOTION
Units_Promotion= df1[df1.PROMOTION==1]\
                    .groupby('CATEGORY').UNITS.sum().reset_index()\
                    .rename(columns={'UNITS':'UNITS_WITH_PROMOTION'})

"""Filter items not on discount, group by category, sum the total units
    in each category and reset index """
#Rename column "UNITS" to 'UNITS_WITHOUT_PROMOTION'
Units_Not_Promotion = df1[df1.PROMOTION==0]\
                    .groupby('CATEGORY').UNITS.sum().reset_index()\
                    .rename(columns={'UNITS':'UNITS_WITHOUT_PROMOTION'})

#Merge the two data frames by category using left merge
Total= Total_Units\
                .merge(Units_Promotion, on='CATEGORY', how='left')\
                .merge(Units_Not_Promotion, on='CATEGORY', how='left')

#create color array
my_colors = ["mediumpurple", "aqua"]

#add color array to set_palette
sns.set_palette( my_colors )

"""Setting as bar chart with the category as the x axis and total units
```

```
    sold as y axis for both products on discount and not on discount """

#Setting labels for legend, and x/y axis
Total.plot(kind='bar', x='CATEGORY', y=['UNITS_WITH_PROMOTION',
                                        'UNITS_WITHOUT_PROMOTION'],\
label=['Promoted', 'Normal Price'])
plt.xlabel("Category")
plt.ylabel("Number of Units Sold")

#setting graph title
plt.title('Total number of units sold by category with and without promotion',
          fontsize=16, y=1.15);
plt.show()
```


Total number of units sold by category with and without promotion
1e6
Promoted
Normal Price
Number of Units Sold
2.5
2.0
1.5
1.0
0.5
0.0
BAG SNACKS
COLD CEREAL
FROZEN PIZZA
ORAL HYGIENE PRODUCTS
Category

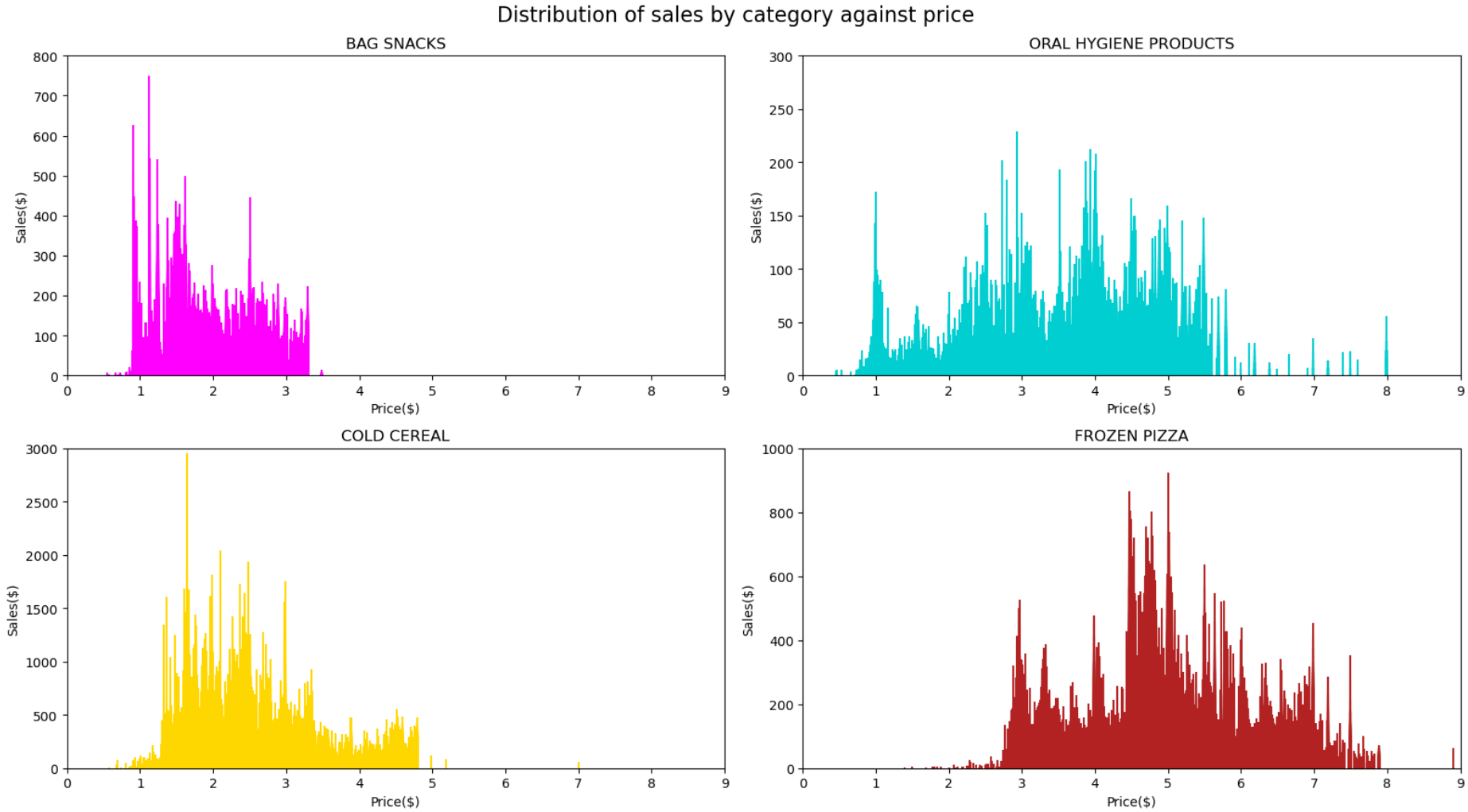
This figure depicts the total number of units sold by category when the promotion is applied and not. For frozen pizza and oral hygiene products, a larger number of units are sold under promotion. For the remaining two, the otherwise is true.

3.3 Examining Effect on Sales

Are there specific price thresholds that, if crossed, drive significant differences in sales?

In [38]:
```
#Creating graph to understand the the price threshold for the bag snacks category
fig=plt.figure(figsize=(16,9))
plt.subplot(2,2,1)

"""for each category set the minimum at 0 and the maximum as the highest spend
    for that category"""

plt.vlines(x = df1[df1['CATEGORY'] == 'BAG SNACKS']['PRICE'],
           ymin = 0,
           ymax = df1[df1['CATEGORY'] == 'BAG SNACKS']['SPEND'],color="magenta")
#for each category set the titles and labels for the graph
plt.title('BAG SNACKS')
plt.xlabel('Price($)')
plt.ylabel('Sales($)')
plt.xlim((0,9)) #for each category set the price limit at the price maximum ($9)
plt.ylim((0,800)) #for each setting the spend limit at the spend maximum ($800)

"""Creating graph to understand the price threshold for the oral hygiene
    products category """
plt.subplot(2,2,2)
plt.vlines(x = df1[df1['CATEGORY'] == 'ORAL HYGIENE PRODUCTS']['PRICE'],
           ymin = 0,
           ymax = df1[df1['CATEGORY'] == 'ORAL HYGIENE PRODUCTS']['SPEND'],
           color="darkturquoise")
plt.title('ORAL HYGIENE PRODUCTS')
plt.xlabel('Price($)')
plt.ylabel('Sales($)')
plt.xlim((0,9))
plt.ylim((0,300)) #setting the spend limit at the spend maximum ($300)

#Creating grapth to understand the price threshold for the cold cereal category
plt.subplot(2,2,3)
plt.vlines(x = df1[df1['CATEGORY'] == 'COLD CEREAL']['PRICE'],
           ymin = 0,
```

```
           ymax = df1[df1['CATEGORY'] == 'COLD CEREAL']['SPEND'],color="gold")
plt.title('COLD CEREAL')
plt.xlabel('Price($)')
plt.ylabel('Sales($)')
plt.xlim((0,9))
plt.ylim((0,3000)) #setting the spend limit at the spend maximum($3000)

#Creating graph to understand the price threshold for the frozen pizza category
plt.subplot(2,2,4)
plt.vlines(x = df1[df1['CATEGORY'] == 'FROZEN PIZZA']['PRICE'],
           ymin = 0,
           ymax = df1[df1['CATEGORY'] == 'FROZEN PIZZA']['SPEND'],color="firebrick")
plt.title('FROZEN PIZZA')
plt.xlabel('Price($)')
plt.ylabel('Sales($)')
plt.xlim((0,9))
plt.ylim((0,1000)) #setting the spend limit at the spend maximum ($1000)

#Setting title for figure
fig.suptitle('Distribution of sales by category against price', fontsize=16);

plt.tight_layout()
plt.show()
```



The peak of each category shows the price threshold in which the price should not be increased further as it will lead to lower sales.

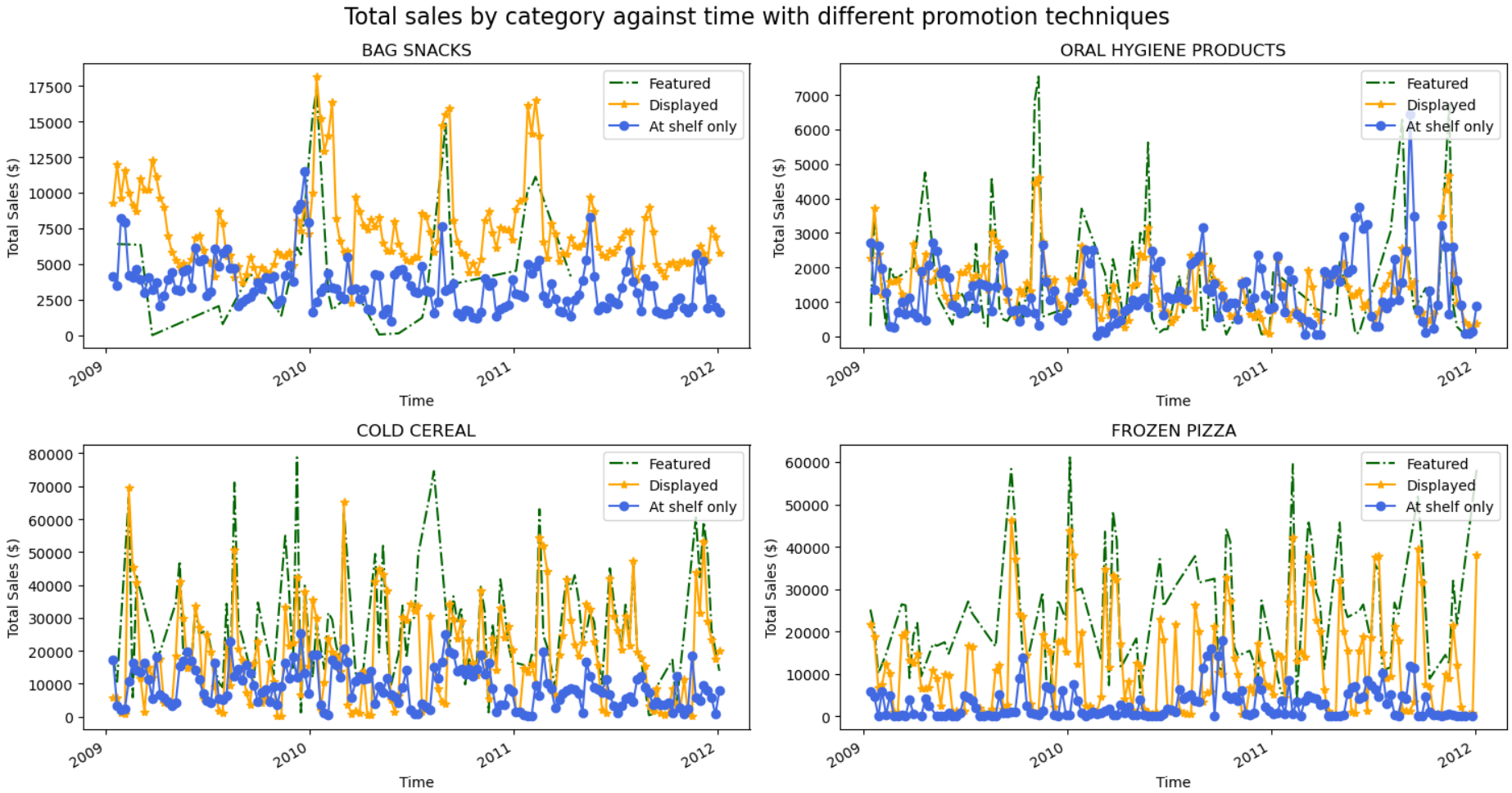
How do the different promotional techniques impact sales over time?

In [39]:
```
#create color array
my_colors = ["darkgreen", "orange","royalblue"]
#add color array to set_palette
sns.set_palette( my_colors )
```

```
"""Evaluating the effect of the different promotional technique on
    each category's sales"""
#Setting figure size
fig=plt.figure(figsize=(15,8))
plt.subplot(2,2,1)
#Filter category for Bag Snacks
bagSnacks = df1[df1['CATEGORY'] == 'BAG SNACKS']
#Creating a date time series to plot the data using WEEK_END_DATE
bagSnacks['WEEK_END_DATE'] = pd.to_datetime(bagSnacks['WEEK_END_DATE'])

#Assigning the promotions to the category
#if promotion applied, include in graph
featured = bagSnacks[bagSnacks['FEATURE'] == 1]
displayed = bagSnacks[bagSnacks['DISPLAY'] == 1]
shelf = bagSnacks[bagSnacks['TPR_ONLY'] ==1]

#for each promotion groupby WEEK_END_DATE and sum SPEND
#set plot style
featured.groupby('WEEK_END_DATE').sum()['SPEND'].plot(style = '-.')
displayed.groupby('WEEK_END_DATE').sum()['SPEND'].plot(style = '*-')
shelf.groupby('WEEK_END_DATE').sum()['SPEND'].plot(style = 'o-')

#set titles for legend, graph, and lables for x/y axis
plt.legend(['Featured', 'Displayed', 'At shelf only'],loc='upper right')
plt.title('BAG SNACKS')
plt.xlabel('Time')
plt.ylabel("Total Sales ($)")

#set ticks and label for date time (2009, 2010, 2011, 2012)
plt.xticks(ticks=[datetime.datetime(2009,1,1),
                  datetime.datetime(2010,1,1),
                  datetime.datetime(2011,1,1),
                  datetime.datetime(2012,1,1)],
           labels=['2009','2010','2011','2012'])

#Create Oral Hygiene Subplot
plt.subplot(2,2,2)
oralHygiene = df1[df1['CATEGORY'] == 'ORAL HYGIENE PRODUCTS']
oralHygiene['WEEK_END_DATE'] = pd.to_datetime(oralHygiene['WEEK_END_DATE'])

featured = oralHygiene[oralHygiene['FEATURE'] == 1]
displayed = oralHygiene[oralHygiene['DISPLAY'] == 1]
shelf = oralHygiene[oralHygiene['TPR_ONLY'] ==1]
featured.groupby('WEEK_END_DATE').sum()['SPEND'].plot(style = '-.')
displayed.groupby('WEEK_END_DATE').sum()['SPEND'].plot(style = '*-')
```

```
shelf.groupby('WEEK_END_DATE').sum()['SPEND'].plot(style = 'o-')
plt.legend(['Featured', 'Displayed', 'At shelf only'],loc='upper right')
plt.title('ORAL HYGIENE PRODUCTS')
plt.xlabel('Time')
plt.ylabel("Total Sales ($)")
plt.xticks(ticks=[datetime.datetime(2009,1,1),
                  datetime.datetime(2010,1,1),
                  datetime.datetime(2011,1,1),
                  datetime.datetime(2012,1,1)],
           labels=['2009','2010','2011','2012'])

#Create Cold Cereal subplot
plt.subplot(2,2,3)
coldCereal = df1[df1['CATEGORY'] == 'COLD CEREAL']
coldCereal['WEEK_END_DATE'] = pd.to_datetime(coldCereal['WEEK_END_DATE'])

featured = coldCereal[coldCereal['FEATURE'] == 1]
displayed = coldCereal[coldCereal['DISPLAY'] == 1]
shelf = coldCereal[coldCereal['TPR_ONLY'] ==1]

featured.groupby('WEEK_END_DATE').sum()['SPEND'].plot(style = '-.')
displayed.groupby('WEEK_END_DATE').sum()['SPEND'].plot(style = '*-')
shelf.groupby('WEEK_END_DATE').sum()['SPEND'].plot(style = 'o-')
plt.legend(['Featured', 'Displayed', 'At shelf only'],loc='upper right')
plt.title('COLD CEREAL')
plt.xlabel('Time')
plt.ylabel("Total Sales ($)")
plt.xticks(ticks=[datetime.datetime(2009,1,1),
                  datetime.datetime(2010,1,1),
                  datetime.datetime(2011,1,1),
                  datetime.datetime(2012,1,1)],
           labels=['2009','2010','2011','2012'])

#Create frozen pizza subplot
plt.subplot(2,2,4)
frozenPizza = df1[df1['CATEGORY'] == 'FROZEN PIZZA']
frozenPizza['WEEK_END_DATE'] = pd.to_datetime(frozenPizza['WEEK_END_DATE'])

featured = frozenPizza[frozenPizza['FEATURE'] == 1]
displayed = frozenPizza[frozenPizza['DISPLAY'] == 1]
shelf = frozenPizza[frozenPizza['TPR_ONLY'] ==1]
featured.groupby('WEEK_END_DATE').sum()['SPEND'].plot(style = '-.')
displayed.groupby('WEEK_END_DATE').sum()['SPEND'].plot(style = '*-')
shelf.groupby('WEEK_END_DATE').sum()['SPEND'].plot(style = 'o-')
plt.legend(['Featured', 'Displayed', 'At shelf only'],loc='upper right')
```

```
plt.title('FROZEN PIZZA')
plt.xlabel('Time')
plt.ylabel("Total Sales ($)")
plt.xticks(ticks=[datetime.datetime(2009,1,1),
                  datetime.datetime(2010,1,1),
                  datetime.datetime(2011,1,1),
                  datetime.datetime(2012,1,1)],
           labels=['2009','2010','2011','2012'])

#Set title for figure
fig.suptitle('Total sales by category against time with different promotion techniques'
             ,fontsize=16);

plt.tight_layout()
plt.show()
```



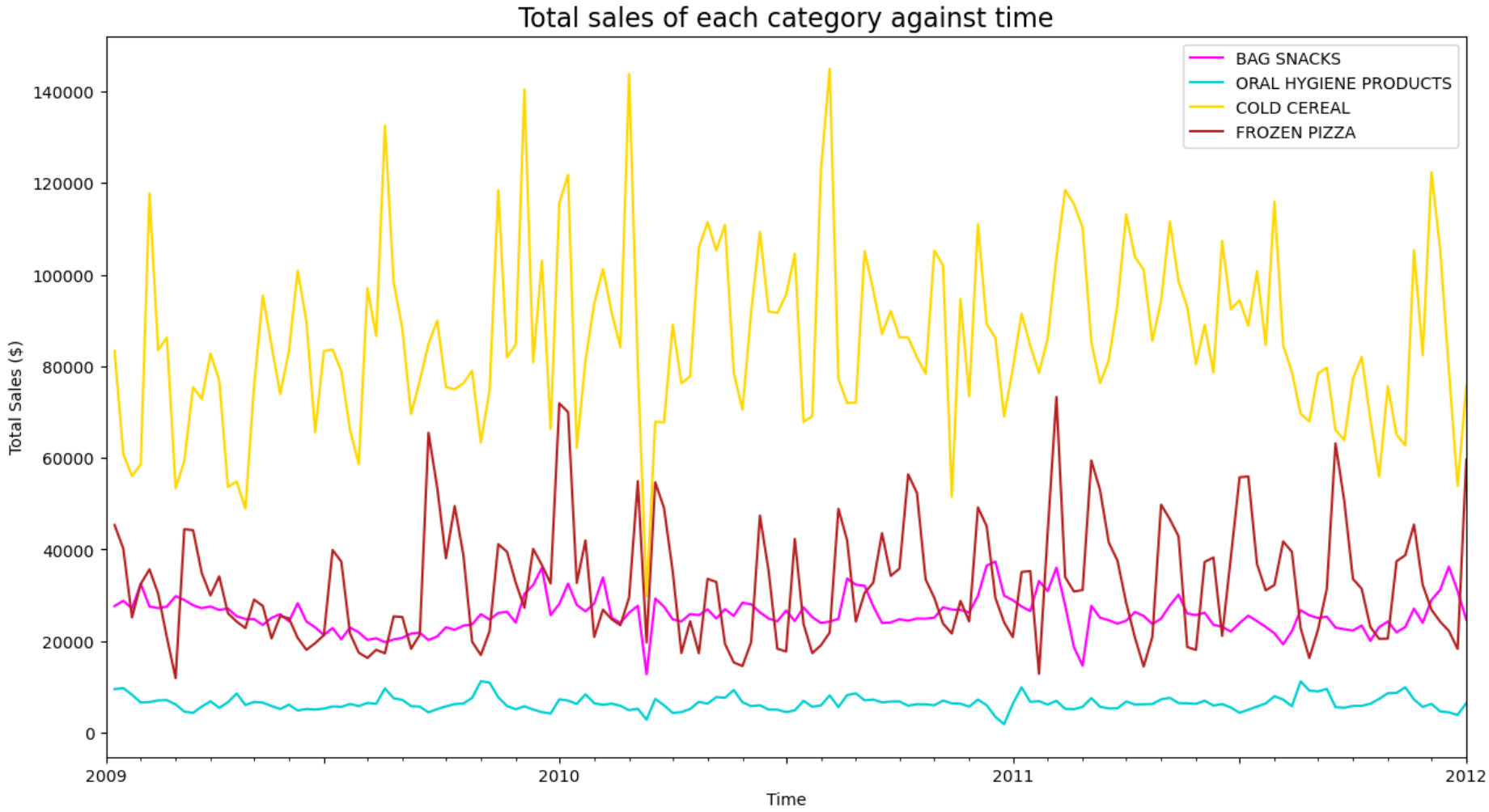
The figure above depicts the weekly sales based on the different promotion techniques. From the above figures, it can be seen that the standard assumption is total sales are lower for 'At Shelf Only'. However, the oral hygiene figure shows increased total sales for 'At Shelf Only,' making it an anomaly in the findings.

How do the total sales of each category vary over time?

In [40]:
```
#Create Bag Snacks Spend dataframe
#grouby WEEK_END_DATE, sum SPEND and reset index
#rename column 'SPEND' to 'SPEND_SNACKS'
BAG_SNACKS_SPEND = df1[df1.CATEGORY=='BAG SNACKS']\
```

```
                        .groupby('WEEK_END_DATE').SPEND.sum().reset_index()\
                        .rename(columns={'SPEND':'SPEND_SNACKS'})

#Create Oral Hygiene Spend dataframe
#grouby WEEK_END_DATE, sum SPEND and reset index
#rename column 'SPEND' to 'SPEND_ORAL'
ORAL_HYGIENE_SPEND= df1[df1.CATEGORY=='ORAL HYGIENE PRODUCTS']\
                        .groupby('WEEK_END_DATE').SPEND.sum().reset_index()\
                        .rename(columns={'SPEND':'SPEND_ORAL'})

#Create Cold Cereal Spend dataframe
#grouby WEEK_END_DATE, sum SPEND and reset index
#rename column 'SPEND' to 'SPEND_COLD_CEREAL'
COLD_CEREAL_SPEND = df1[df1.CATEGORY=='COLD CEREAL']\
                        .groupby('WEEK_END_DATE').SPEND.sum().reset_index()\
                        .rename(columns={'SPEND':'SPEND_COLD_CEREAL'})

#Create Frozen Pizza Spend dataframe
#grouby WEEK_END_DATE, sum SPEND and reset index
#rename column 'SPEND' to 'SPEND_FROZEN_PIZZA'
FROZEN_PIZZA_SPEND = df1[df1.CATEGORY=='FROZEN PIZZA']\
                        .groupby('WEEK_END_DATE').SPEND.sum().reset_index()\
                        .rename(columns={'SPEND':'SPEND_FROZEN_PIZZA'})

#Merge the three dataframes previously created on WEEK_END_DATE with a left merge
T_p= BAG_SNACKS_SPEND\
                    .merge(ORAL_HYGIENE_SPEND, on='WEEK_END_DATE', how='left')\
                    .merge(COLD_CEREAL_SPEND, on='WEEK_END_DATE', how='left')\
                    .merge(FROZEN_PIZZA_SPEND, on='WEEK_END_DATE', how='left')

#Creating a date time series to plot the data using WEEK_END_DATE
T_p['WEEK_END_DATE'] = pd.to_datetime(T_p['WEEK_END_DATE'])

#Sort values by WEEK_END_DATE
T_p.sort_values(by='WEEK_END_DATE',inplace=True)
#Set index to WEEK_END_DATE
T_p.set_index(T_p.WEEK_END_DATE, inplace=True)
#Drop WEEK_END_DATE as column because it was set as an index
T_p.drop(columns=['WEEK_END_DATE'], inplace=True)

#Assign plot colors
T_p.plot(color=["magenta","darkturquoise","gold","firebrick"],figsize=(15,8))
#Set title for legend, set title for graph, and create labels for x and y axis
plt.legend(['BAG SNACKS', 'ORAL HYGIENE PRODUCTS','COLD CEREAL','FROZEN PIZZA'])
plt.title('Total sales of each category against time', fontsize=16)
```

```
plt.xlabel('Time')
plt.ylabel("Total Sales ($)")

#Plot x ticks (2009, 2010, 2011, 2012)
plt.xticks(ticks=[datetime.datetime(2009,1,1),datetime.datetime(2009,7,1),
                  datetime.datetime(2010,1,1),datetime.datetime(2010,7,1),
                  datetime.datetime(2011,1,1),datetime.datetime(2011,7,1),
                  datetime.datetime(2012,1,1)],
           labels=['2009','','2010','','2011','','2012']);
```



The figure above shows that cold cereal and oral hygiene products have high variations in sales over time, whereas oral hygiene products and

bag snacks have low variations. Cold cereal has the highest sales, while bag snacks have the lowest.

4 Modelling

We first experimented with the SVR model. However, due to relatively constant predictions, we decided to use the XGboost and LGBM models to predict sales for each of the four product categories. The reason why we generate price predictions for each category is because the price range for each category differs.

In [41]:
```
#split data into training and testing set. Training data will be data before 1 July,
#while data after 1 July will be testing data.
#data split ratio is around 80:20
def split_into_training_and_testing(df,split_into_training_and_testing='01-Jul-2011'):
    training_df = df.loc[df.index <= split_into_training_and_testing].copy()
    testing_df = df.loc[df.index > split_into_training_and_testing].copy()
    return training_df,testing_df

#Feature engineering, decompose all time-related features, such as date, day of week,
# to prepare for time-series forecasting
def create_the_features(df, label=None):
    df['date'] = df.index
    df['day_of_week'] = df['date'].dt.dayofweek
    df['quarter'] = df['date'].dt.quarter
    df['m'] = df['date'].dt.month
    df['y'] = df['date'].dt.year
    df['day_of_y'] = df['date'].dt.dayofyear
    df['day_of_m'] = df['date'].dt.day
    df['week_of_y'] = df['date'].dt.weekofyear
    X = df[['day_of_week','quarter','m','y',
           'day_of_y','day_of_m','week_of_y']]
    if label:
        y = df[label]
        return X, y
    return X

def training_testing_m(training_df,testing_df,create_the_features,labeler):

    # create the features to generate X train and Y train.
    # X train, Y train, X test, Y test will be the data used for the model training
    X_train, y_train = create_the_features(training_df, labeler)
    X_test, y_test = create_the_features(testing_df, labeler)
```

```
#Section 1: Training and Testing of the SVR model
# list down the possible parameters for C and gamma by guessing
    parameters = {
    "kernel": ["rbf"],
    "C": [1,10,100,1000,10000,100000],
    "gamma": [1e-8, 1e-7, 1e-6, 1e-5, 1e-4, 1e-3, 1e-2, 1e-1]
    }

#finding the best SVR parameters C and gamma that optimise model training
    print("Searching for best SVR parameters, C and gamma for the {} category"
          .format(labeler))
    grid = GridSearchCV(SVR(), parameters, cv=5, verbose=False)
    #fit the data onto GridSearch to find best parameters
    grid.fit(X_train, y_train)
    print("The best SVR parameters are for the {} category are  C={} and gamma={}"
          .format(labeler,grid.best_params_.get("C"),grid.best_params_.get("gamma")))

#set up the SVR model and scale the data using Standard Scaler
    tester_m = make_pipeline(StandardScaler(), SVR(C=grid.best_params_.get("C"),
                                                   gamma=grid.best_params_.get("gamma")))
#train the SVR model
    print("Training SVR model for the {} category"
          .format(labeler))
    tester_m.fit(X_train, y_train)

#use the SVR model to attain predictions
    print("Testing SVR model for the {} category"
          .format(labeler))
    y_pred0=tester_m.predict(X_test)


#calculate the RMSE, MAPE for the SVR model
    mean_squared_error0= mse(y_test, y_pred0)
    acc_score0=mape(y_test, y_pred0)
    root_mean_quare_deviation0=np.sqrt(mean_squared_error0)

# Section 2: Training and Testing of the XGB model

#set up the xgboost model
    first_m = xgb.XGBRegressor()

#train the xgboost model
    print("Training xgboost model for the {} category"
```

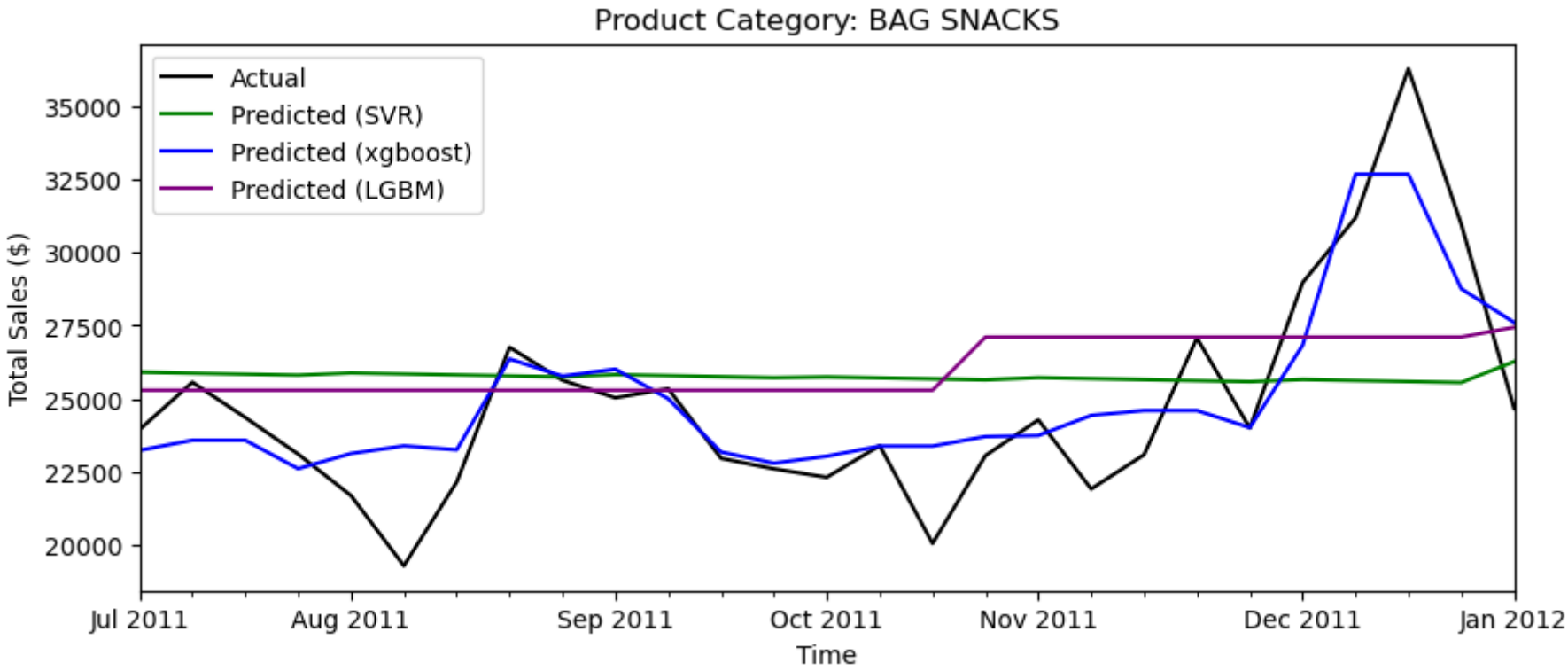
```
          .format(labeler))
    first_m.fit(X_train, y_train,
            eval_set=[(X_train, y_train), (X_test, y_test)],
            early_stopping_rounds=50,
             eval_metric='rmse',
           verbose=False)

#use the xgboost model to attain predictions
    print("Testing xgboost model for the {} category"
          .format(labeler))
    y_pred1=first_m.predict(X_test)

#calcualte the RMSE, MAPE for the xgboost model
    mean_squared_error1 = mse(y_test, y_pred1)
    acc_score1=mape(y_test, y_pred1)
    root_mean_quare_deviation1=np.sqrt(mean_squared_error1)

# Section 3: Training and Testing of the LGBM model

#set up the LGBM model
    second_m = LGBMRegressor()

#train the LGBM model
    print("Training LGBM model for the {} category".format(labeler))
    second_m.fit(X_train, y_train,
            eval_set=[(X_train, y_train), (X_test, y_test)],
            early_stopping_rounds=50,
            eval_metric='rmse',
            verbose=False)

#use the LGBM model to attain predictions
    print("Testing LGBM model for the {} category".format(labeler))
    y_pred2=second_m.predict(X_test)

    print("All training and testing for the {} category finshed.".format(labeler))

#calculate the RMSE, MAPE for the LGBM model
    mean_squared_error2= mse(y_test, y_pred2)
    acc_score2=mape(y_test, y_pred2)
    root_mean_quare_deviation2=np.sqrt(mean_squared_error2)

    errors_rmse = pd.Series([root_mean_quare_deviation0,root_mean_quare_deviation1,
                             root_mean_quare_deviation2], name=labeler)
    errors_acc = pd.Series([acc_score0,acc_score1,acc_score2], name=labeler)
```

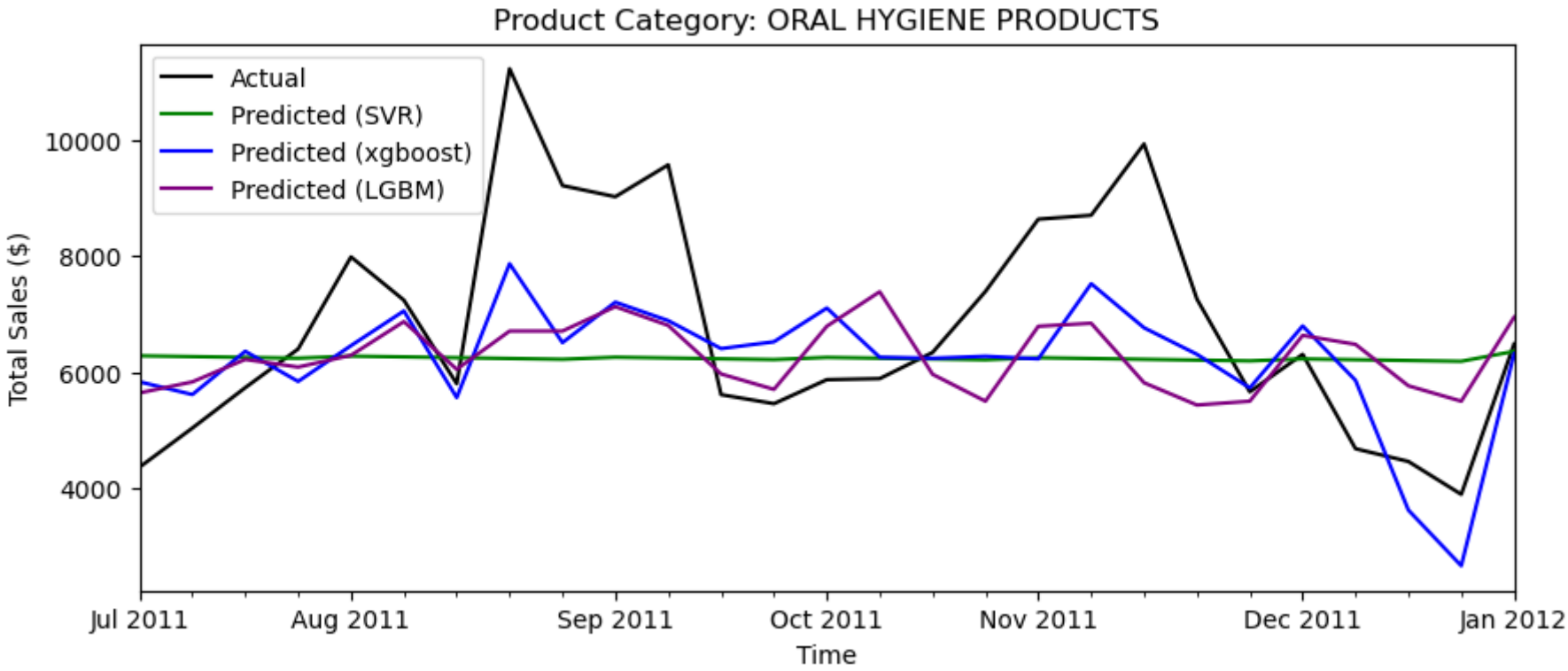
```
    return errors_rmse,errors_acc,y_pred0,y_pred1,y_pred2

def plot_predictions(testing_df,training_df,labeler,y_pred0,y_pred1,y_pred2):
    #append predictions to the testing set
    testing_df[labeler+'_Prediction0'] = y_pred0
    testing_df[labeler+'_Prediction1'] = y_pred1
    testing_df[labeler+'_Prediction2'] = y_pred2

#plot predictions, for both predicted and actual test data
    testing_df[[labeler,labeler+'_Prediction0',labeler+'_Prediction1',
                labeler+'_Prediction2']].plot(figsize=(10,4),color=['black','green',
                                                                    'blue','purple']);

#include legend
    plt.legend(['Actual','Predicted (SVR)','Predicted (xgboost)','Predicted (LGBM)'],
               loc='upper left')

#plot x and y labels
    plt.xlabel('Time')
    plt.ylabel("Total Sales ($)")

#plot x ticks
    plt.xlim(['01-Jul-2011','04-Jan-2012'])
    plt.xticks(ticks=[datetime.datetime(2011,7,1),datetime.datetime(2011,8,1),
                      datetime.datetime(2011,9,1),datetime.datetime(2011,10,1),
                      datetime.datetime(2011,11,1),datetime.datetime(2011,12,1),
                      datetime.datetime(2012,1,1)],labels=['Jul 2011','Aug 2011',
                                                           'Sep 2011','Oct 2011',
                                                           'Nov 2011','Dec 2011',
                                                           'Jan 2012']);

def overall_model_training(origdata,labeler,split_into_training_and_testing,
                           training_testing_m,plot_predictions):
#prepare data for training and testing
    temp=pd.DataFrame(T_p[labeler])
    temp.set_index(T_p.index, inplace=True)

#split data into training and testing set.
    training_df,testing_df=split_into_training_and_testing(temp)

#training and testing of the model
    errors_rmse,errors_acc,y_pred0,y_pred1,y_pred2 = training_testing_m(training_df,
                                                                        testing_df,
                                                                        create_the_features,
                                                                        labeler)
```

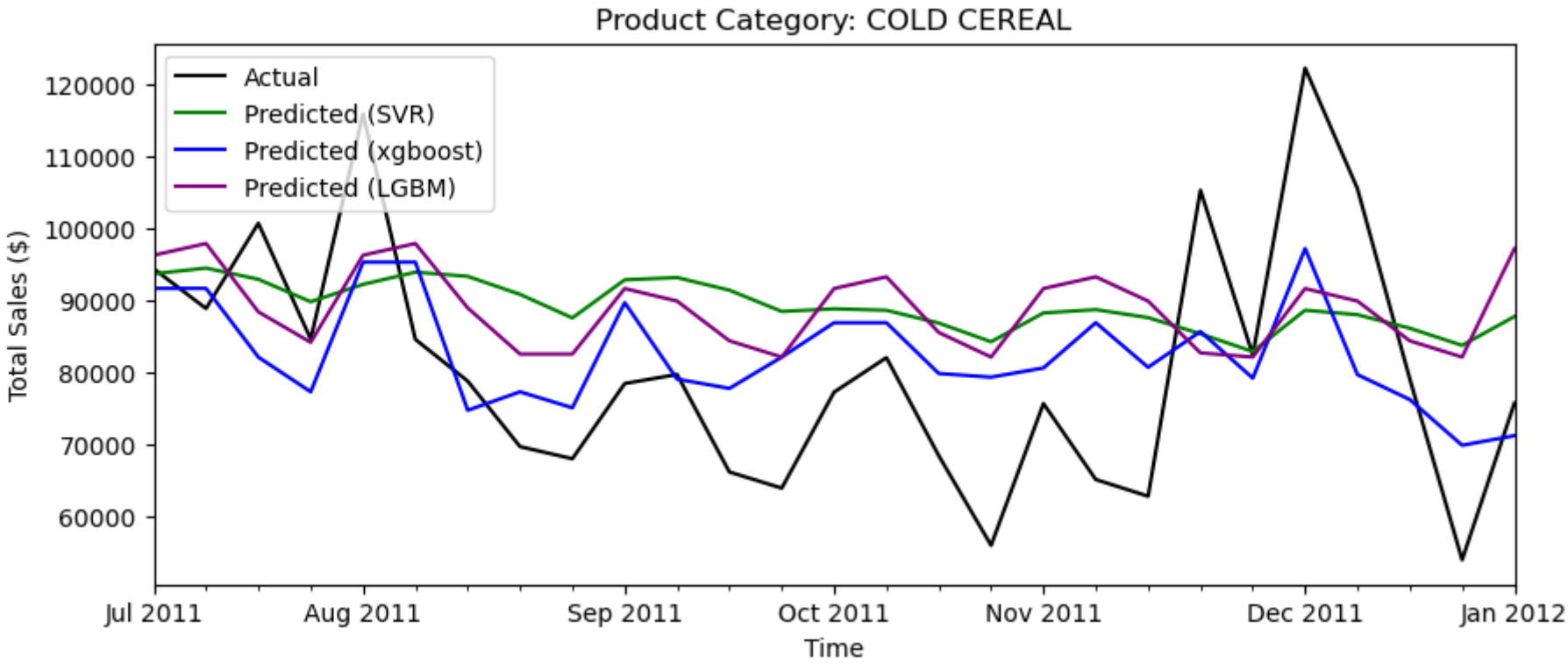
```
#plot predictions, for both predicted and actual test data
    plot_predictions(testing_df, training_df,labeler,y_pred0,y_pred1,y_pred2)
    return errors_rmse,errors_acc
```

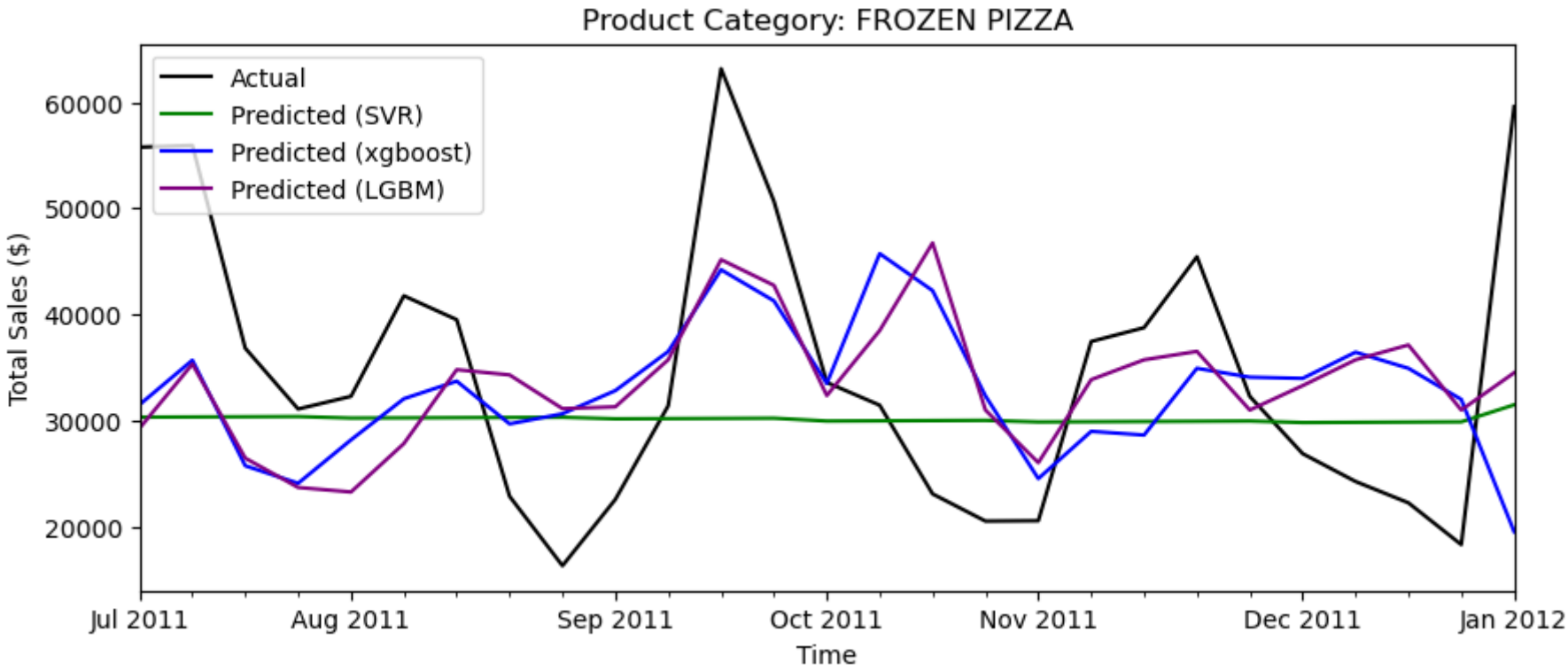
In [42]:
```
# fit and test model onto the SNACKS category
error_snacks_rmse,error_snacks_acc=overall_model_training(T_p,'SPEND_SNACKS',
                                                          split_into_training_and_testing,
                                                          training_testing_m,plot_predictions)
# insert plot title
plt.title('Product Category: BAG SNACKS');

# fit and test model onto the ORAL HYGIENE PRODUCTS category
error_oral_rmse,error_oral_acc=overall_model_training(T_p,'SPEND_ORAL',
                                                      split_into_training_and_testing,
                                                      training_testing_m,plot_predictions)
# insert plot title
plt.title('Product Category: ORAL HYGIENE PRODUCTS');

# fit and test model onto the COLD CEREAL category
error_cereal_rmse,error_cereal_acc=overall_model_training(T_p,'SPEND_COLD_CEREAL',
                                                          split_into_training_and_testing,
                                                          training_testing_m,plot_predictions)
# insert plot title
plt.title('Product Category: COLD CEREAL');

# fit and test model onto the PIZZA category
error_pizza_rmse,error_pizza_acc=overall_model_training(T_p,'SPEND_FROZEN_PIZZA',
                                                        split_into_training_and_testing,
                                                        training_testing_m,plot_predictions)
# insert plot title
plt.title('Product Category: FROZEN PIZZA');
```

```
Searching for best SVR parameters, C and gamma for the SPEND_SNACKS category
The best SVR parameters are for the SPEND_SNACKS category are  C=100000 and gamma=1e-05
Training SVR model for the SPEND_SNACKS category
Testing SVR model for the SPEND_SNACKS category
Training xgboost model for the SPEND_SNACKS category
Testing xgboost model for the SPEND_SNACKS category
Training LGBM model for the SPEND_SNACKS category
Testing LGBM model for the SPEND_SNACKS category
All training and testing for the SPEND_SNACKS category finshed.
Searching for best SVR parameters, C and gamma for the SPEND_ORAL category
The best SVR parameters are for the SPEND_ORAL category are  C=10000 and gamma=0.0001
Training SVR model for the SPEND_ORAL category
Testing SVR model for the SPEND_ORAL category
Training xgboost model for the SPEND_ORAL category
Testing xgboost model for the SPEND_ORAL category
Training LGBM model for the SPEND_ORAL category
Testing LGBM model for the SPEND_ORAL category
All training and testing for the SPEND_ORAL category finshed.
Searching for best SVR parameters, C and gamma for the SPEND_COLD_CEREAL category
The best SVR parameters are for the SPEND_COLD_CEREAL category are  C=10000 and gamma=0.1
Training SVR model for the SPEND_COLD_CEREAL category
Testing SVR model for the SPEND_COLD_CEREAL category
Training xgboost model for the SPEND_COLD_CEREAL category
Testing xgboost model for the SPEND_COLD_CEREAL category
Training LGBM model for the SPEND_COLD_CEREAL category
Testing LGBM model for the SPEND_COLD_CEREAL category
All training and testing for the SPEND_COLD_CEREAL category finshed.
Searching for best SVR parameters, C and gamma for the SPEND_FROZEN_PIZZA category
The best SVR parameters are for the SPEND_FROZEN_PIZZA category are  C=10000 and gamma=0.001
Training SVR model for the SPEND_FROZEN_PIZZA category
Testing SVR model for the SPEND_FROZEN_PIZZA category
Training xgboost model for the SPEND_FROZEN_PIZZA category
Testing xgboost model for the SPEND_FROZEN_PIZZA category
Training LGBM model for the SPEND_FROZEN_PIZZA category
Testing LGBM model for the SPEND_FROZEN_PIZZA category
All training and testing for the SPEND_FROZEN_PIZZA category finshed.
```


Product Category: BAG SNACKS
Actual
Predicted (SVR)
Predicted (xgboost)
Predicted (LGBM)
Total Sales ($)
35000
32500
30000
27500
25000
22500
20000
Jul 2011
Aug 2011
Sep 2011
Oct 2011
Nov 2011
Dec 2011
Jan 2012
Time


Product Category: ORAL HYGIENE PRODUCTS
Actual
Predicted (SVR)
Predicted (xgboost)
Predicted (LGBM)
Total Sales ($)
10000
8000
6000
4000
Jul 2011
Aug 2011
Sep 2011
Oct 2011
Nov 2011
Dec 2011
Jan 2012
Time


Product Category: COLD CEREAL
Actual
Predicted (SVR)
Predicted (xgboost)
Predicted (LGBM)
Total Sales ($)
120000
110000
100000
90000
80000
70000
60000
Jul 2011
Aug 2011
Sep 2011
Oct 2011
Nov 2011
Dec 2011
Jan 2012
Time



From the graphs, xgboost and LGBM performed similarly compared to SVR in predicting total sales across the different categories.

5 Evaluation of Models

RMSE

In [43]:
```
#calculate RMSE for each category
error_total1=pd.concat([error_snacks_rmse, error_oral_rmse,
                        error_cereal_rmse,error_pizza_rmse], axis=1)
error_total1['Model']=['SVR','xgboost', 'LGBM']
error_total1.set_index('Model')
```

Out[43]:

Model	SPEND_SNACKS	SPEND_ORAL	SPEND_COLD_CEREAL	SPEND_FROZEN_PIZZA
SVR	3733.595047	1948.529755	18299.181848	13645.692083
xgboost	1792.012992	1509.074268	13983.331957	14030.968970
LGBM	3464.393735	1781.510331	17557.937990	12850.678999

SVR has the highest RMSE among the models, except for frozen pizza where it is slightly lower than xgboost but still high, indicating that it is the least accurate.

MAPE

In [44]:
```
#calculate MAPE for each category
error_total2=pd.concat([error_snacks_acc, error_oral_acc,
                        error_cereal_acc,error_pizza_acc], axis=1)
error_total2['Model']=['SVR','xgboost', 'LGBM']
error_total2.set_index('Model')
```

Out[44]:

Model	SPEND_SNACKS	SPEND_ORAL	SPEND_COLD_CEREAL	SPEND_FROZEN_PIZZA
SVR	0.120288	0.211639	0.214739	0.302620
xgboost	0.055914	0.166446	0.149720	0.359236
LGBM	0.115073	0.193855	0.206629	0.349638

xgboost and LGBM have lower MAPE than SVR for all categories other than frozen pizza. However, given that the SVR model generated non-meaningful, constant predictions, we disregarded it.

Overall, xgboost and LGBM display promising results that should be investigated and developed further for commercialisation.

6 Findings and Implications

6.1 Summary and Recommendation

From the exploratory data analysis, the findings and the respective recommendation can be summarised as follows:

- Grocery retailers can use the above results from xgboost and LGBM to predict the trend in sales over time. They can start analysing these four categories before expanding to other categories. This can help with inventory management, especially for time periods when they are expecting lower sales.
- Grocery retailers can conduct further research by using the same models to predict prices over time. Thus, this can help them develop optimal pricing strategies based on the model's results. For example, the category with the highest mean price is frozen pizza, and the category with the lowest is bag snacks. The models will allow the grocery retailer to determine the best price for these categories.
- Oral hygiene products have the most extensive range of prices. Generally, categories with a large price range indicate that there are many types of products within that category. We recommend grocery retailers consider implementing strategies to help consumers navigate the oral hygiene product category more easily by providing precise and detailed product information, offering in-store demonstrations, and implementing organised displays to help customers find products best suited to their needs.
- The product category with the most significant price outliers is cold cereal. Therefore, we recommend grocery retailers carefully review the prices of their cold cereal products and make any necessary adjustments.
- Cold cereal is the category with the highest sales. Since cold cereal is already successful, we recommend grocery stores focus their marketing and business strategies on increasing sales in the other three lagging categories.
- More units of snacks and cold cereal are likely to be sold without promotion. To increase sales, we recommend grocery stores continue selling snacks and cold cereal without promotion.
- Each category has a price threshold in which customers are less likely to buy the product if the price surpasses the threshold. Thus, we recommend that prices should not surpass the price thresholds in order to maintain consistent sales.
- Total sales are lower when "At Shelf Only" and highest when featured. This affirms that grocery retailers should continue using featured promotion as it is an effective marketing strategy.

7 Limitations

The dataset records grocery transactions from January 2009 to January 2012. Given that market trends, consumer behaviour, and economic conditions can change over time, the data may not be relevant or representative of the current situation.

Another limitation is that we may not have included all factors which impact sales. If such factors were included, more accurate results would have been attained.

In [45]:
```
#Check wordcount
import json
def wordcount(nb_filename):

  with open(nb_filename) as json_file:
      data = json.load(json_file)

  wordCount = 0
  for each in data['cells']:
      cellType = each['cell_type']
      if cellType == "markdown":
          content = each['source']
          for line in content:
              temp = [word for word in line.split()]
              wordCount = wordCount + len(temp)
  return wordCount
wordcount('MSIN0143_2022_GROUP_A2.ipynb')
```

Out[45]:
```
1934
```

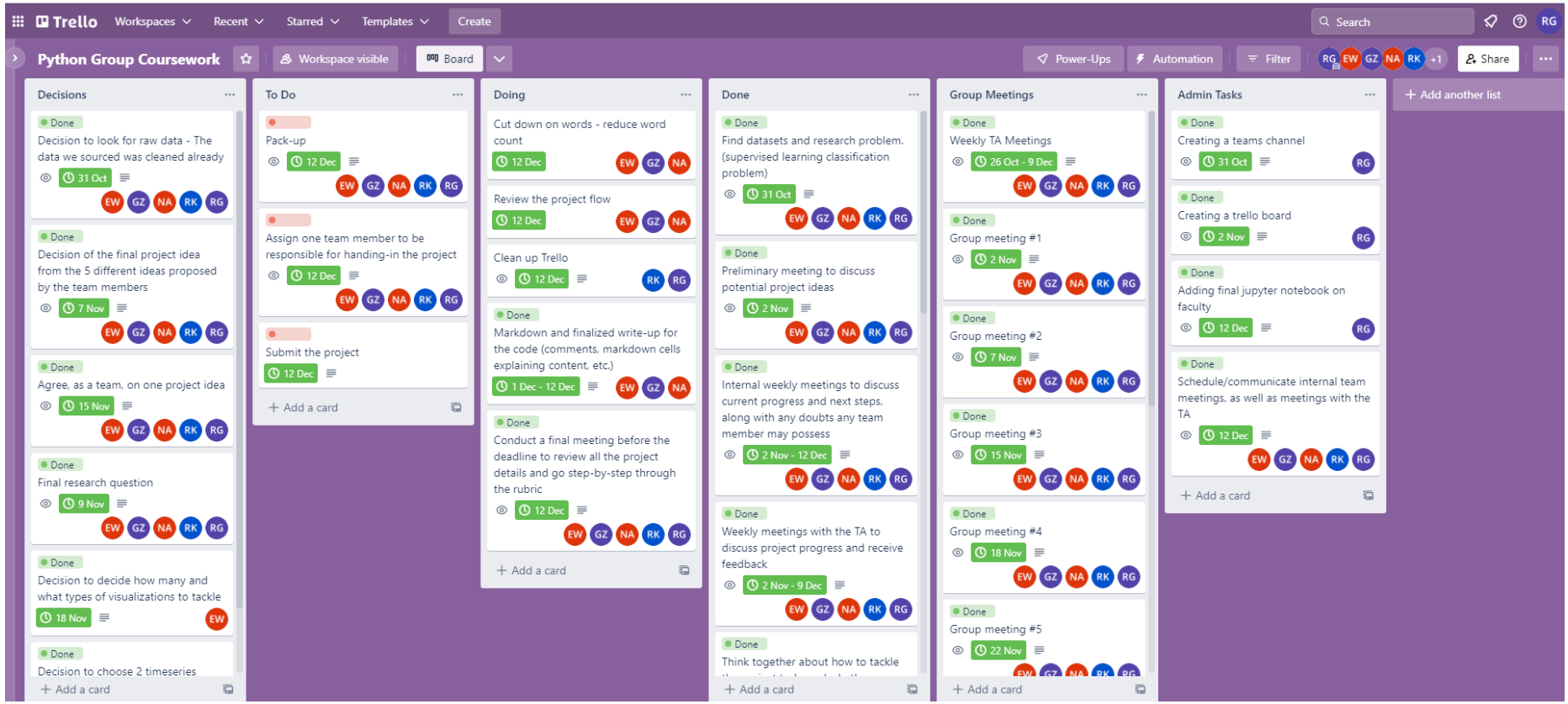
Appendix

A. Trello Board

Trello was used regularly as our project management platform in this group assignment. We benefited greatly from its capacity to present our team with a clear perspective of our project's state, progress, and documentation. We described and categorised new tasks and meetings each week, assigning them to each group member and keeping track of attendance. We had weekly sprints with completed tasks that were then assigned to the relevant week of completion, as shown in the screenshots below. Trello has thus proven to be an effective project management tool for us.

In [46]:
```
Image(filename='TrelloBoard1.png')
```

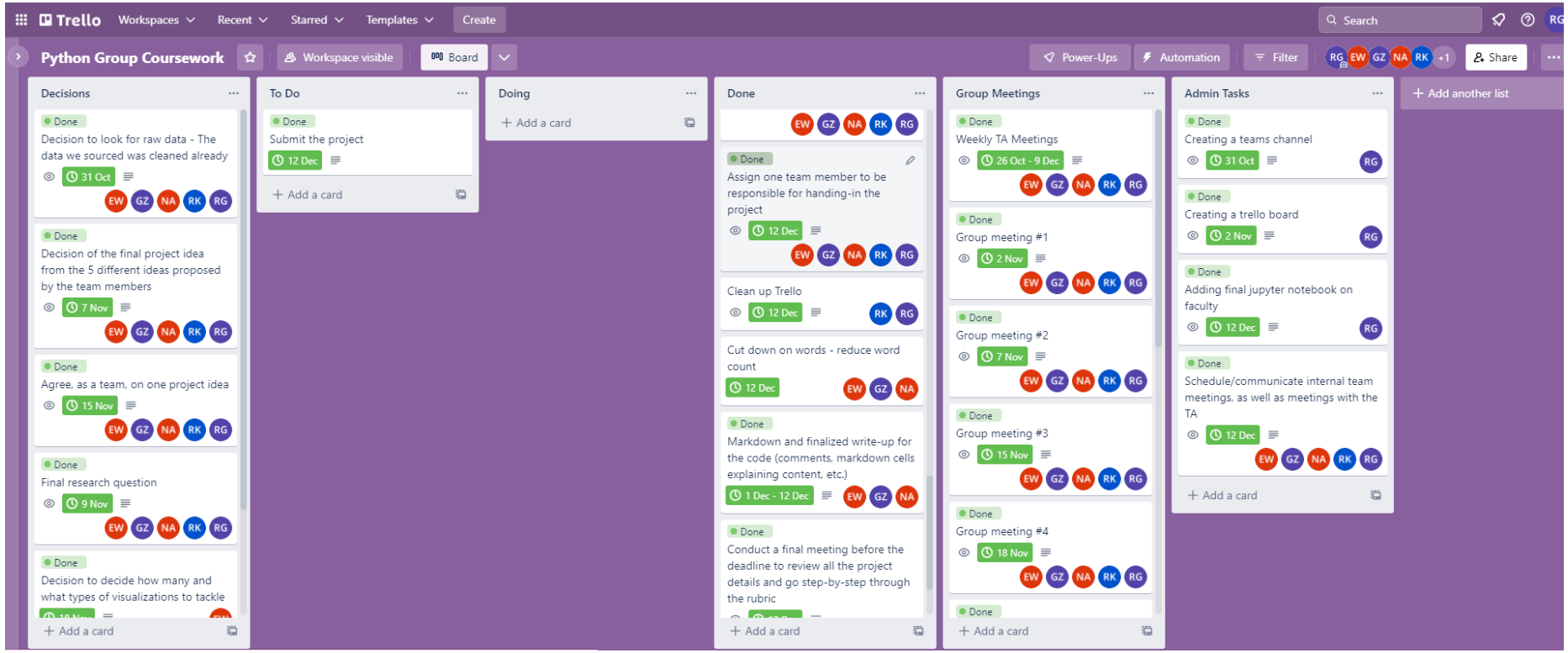
Out[46]:



In [47]:

```
Image(filename='TrelloBoard2.png')
```

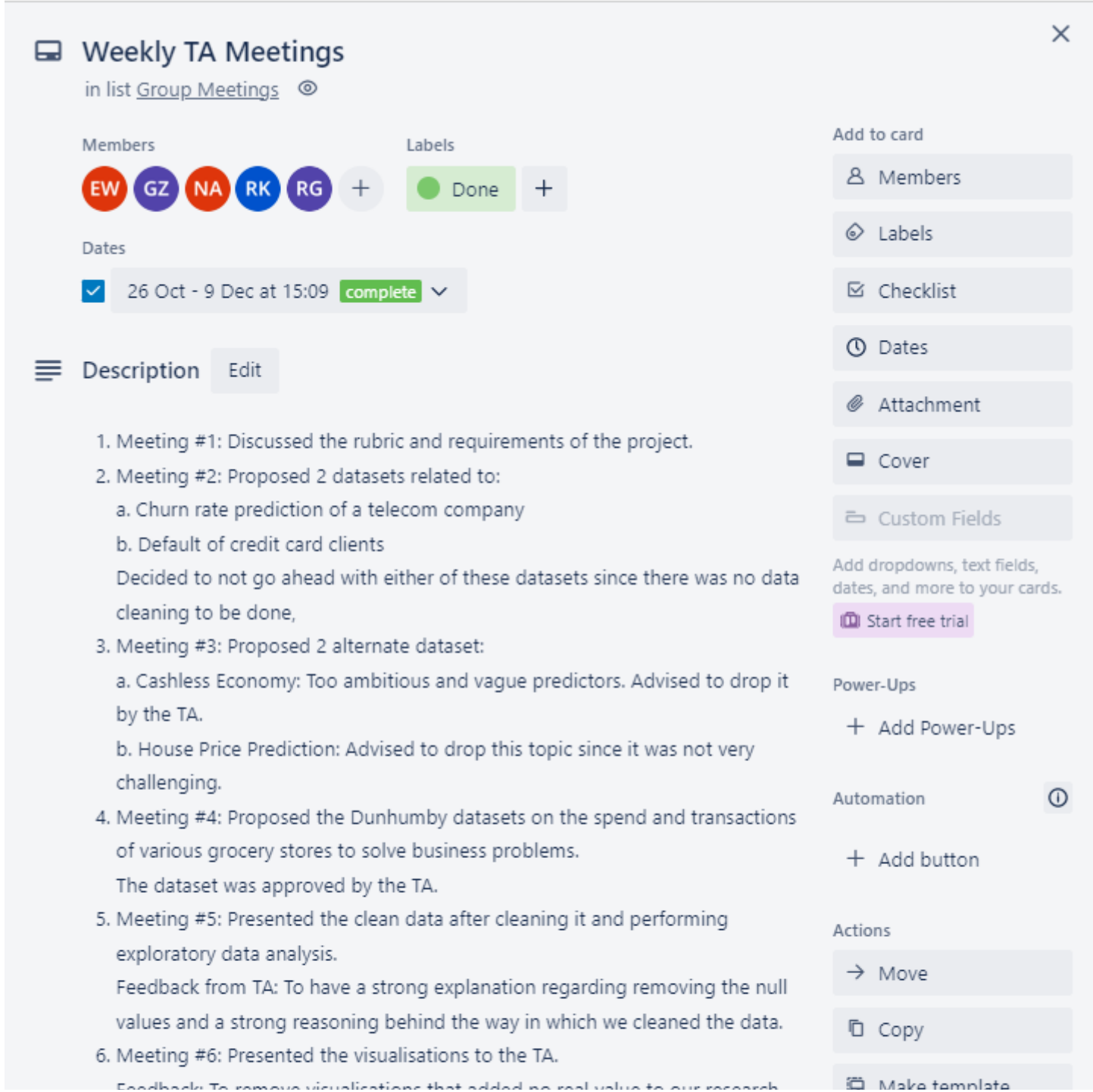
Out[47]:



In [48]:

```
Image(filename='TrelloBoard3.png')
```

Out[48]:


Weekly TA Meetings
in list Group Meetings
Members
EW GZ NA RK RG
Labels
Done
Dates
26 Oct - 9 Dec at 15:09 complete
Description
Edit
1. Meeting #1: Discussed the rubric and requirements of the project.
2. Meeting #2: Proposed 2 datasets related to:
a. Churn rate prediction of a telecom company
b. Default of credit card clients
Decided to not go ahead with either of these datasets since there was no data cleaning to be done,
3. Meeting #3: Proposed 2 alternate dataset:
a. Cashless Economy: Too ambitious and vague predictors. Advised to drop it by the TA.
b. House Price Prediction: Advised to drop this topic since it was not very challenging.
4. Meeting #4: Proposed the Dunhumby datasets on the spend and transactions of various grocery stores to solve business problems.
The dataset was approved by the TA.
5. Meeting #5: Presented the clean data after cleaning it and performing exploratory data analysis.
Feedback from TA: To have a strong explanation regarding removing the null values and a strong reasoning behind the way in which we cleaned the data.
6. Meeting #6: Presented the visualisations to the TA.
Add to card
Members
Labels
Checklist
Dates
Attachment
Cover
Custom Fields
Add dropdowns, text fields, dates, and more to your cards.
Start free trial
Power-Ups
Add Power-Ups
Automation
Add button
Actions
Move
Copy
Make template

problem and only add visualisations if we have a strong reasoning behind it.

7. Meeting #7: Presented the ARIMAX model to the TA.
   Feedback: Our model was underfitting the data and not giving accurate predictions. There were too many steps involved with the model and it was complicated to fit. Advised to drop the model.

Watch

Archive

Share

B. More visualisations

Here, we included extra visuals.

If you were the retailer, for which products would you be more likely to lower the price to increase sales?

In [49]:
```
df_low_spend = df1.groupby(['UPC','CATEGORY'])[['PRICE','SPEND']].mean().sort_values(by=['CATEGORY','SPEND'],ascending
df_low_spend['sub_group_rank_price'] = df_low_spend.groupby('CATEGORY')['PRICE'].rank(ascending=False)
df_low_spend['sub_group_rank_sales'] = df_low_spend.groupby('CATEGORY')['SPEND'].rank(ascending=True)
df_low_spend['total_rank'] = df_low_spend['sub_group_rank_price']+df_low_spend['sub_group_rank_sales']
df_low_spend.sort_values(by=['CATEGORY','total_rank'],ascending = [True, True])
df_low_spend.rename(columns = {'PRICE':'Price ($)', 'SPEND':'Sales ($)'}, inplace = True)

UPC_low=df_low_spend.loc[df_low_spend.groupby('CATEGORY').total_rank.idxmin()]
UPC_low
```

Out[49]:

UPC	CATEGORY	Price ($)	Sales ($)	sub_group_rank_price	sub_group_rank_sales	total_rank
2.840002e+09	BAG SNACKS	2.838999	31.055912	1.0	5.0	6.0
8.849120e+10	COLD CEREAL	3.197299	83.603299	2.0	6.0	8.0
2.066201e+09	FROZEN PIZZA	5.918291	38.724437	2.0	1.0	3.0
3.500069e+09	ORAL HYGIENE PRODUCTS	6.962958	12.345048	1.0	3.0	4.0

In this step, the mean price and sales for each category is calculated. Since there is no clear cut item in each category that has extrememly low sales and high price, the ranking method is used. Each item in the category is ranked by price and sales, with 1 representing the highest price and lowest sales respectively. For each category, the item with the lowest total rank requires a decrease in high average unit price to boost the relatively low average total sales.

References

[1] Aull B, Coggins B, Kohli S, Marohn E. The state of grocery in North America [Internet]. McKinsey & Company. McKinsey & Company; 2022 [cited 2022Dec12]. Available from: https://www.mckinsey.com/industries/retail/our-insights/the-state-of-grocery-in-north-america-2022

[2] Source files - dunnhumby [Internet]. [cited 2022Dec12]. Available from: https://www.dunnhumby.com/source-files/