Technical report

This report was made by Frederik Kjær s174478, Jonas Christian Rask Levin s174462, Sofie Bach s174500, Caroline Sofie Ljørring s174476

Link til github: https://github.com/FKjaeren/Advanced_BA_project (https://github.com/FKjaeren/Advanced_BA_project)

Introduction

In this project, we would like to make recommendations to Amazon in order to improve their products and thereby their sales. The aim is to identify features related with high rated products which Amazon can transfer to low rated products. The category of products we will be investigating is Grocery and Gourmet Foods. We use two datasets for the analysis. One dataset with the ratings of the products and one dataset containing metadata of the products.

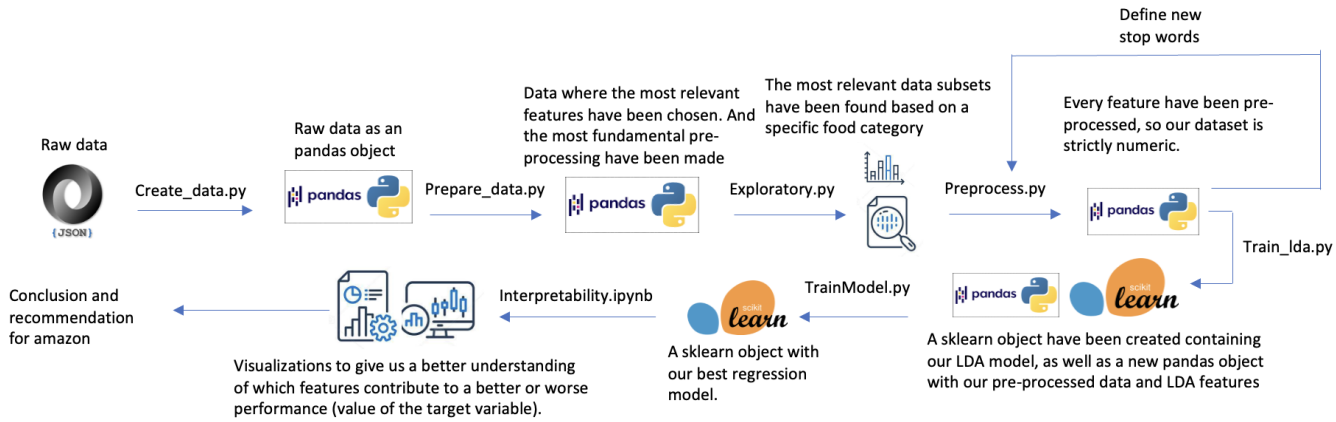
The rating dataset contains all ratings of each product. The rating data is of dimension 5074160 x 4 and contains the following variables: 'item', 'user', 'rating', 'timestamp'.

The metadata dataset contains information of the products. Thus the dataset contains one row per product. The metadatset is of dimension 287051 x 10 with the following variables: 'category', 'description', 'title', 'also_buy', 'brand', 'feature', 'rank', 'also_view', 'price', 'asin'.

To analyze what features give high ratings, we will build a regression model with average rating as response and product features as explanatory variables. One of the interesting explanatory variables is product "description" which we will use to to extract key feautures of the products using LDA (topic modelling). The model will then be analysed through feature importance to see which topics are describing high rated products.

The process of our project

Below we have constructed a visulization of our work flow throughout the process working with our project



Data preparation

Both datasets, the ratings and metadata, are loaded with the function called "load_data' from the script "CreateData.py". The "load_data" function is seen below.

In [1]:

```
## Import packages

# basic
import pandas as pd
import numpy as np
import gower
import pickle
from datetime import date
from math import prod

# plotting
import seaborn as sns
import matplotlib.pyplot as plt
import wordcloud

# text processing
from collections import Counter
from sklearn.feature_extraction import _stop_words
import string
import nltk
nltk.download('wordnet')
nltk.download('punkt')
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
from bs4 import BeautifulSoup

# numeric processing
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()

# modelling
from sklearn.cluster import DBSCAN
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.decomposition import LatentDirichletAllocation
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import catboost
# 7 popular regression models
from sklearn.linear_model import LinearRegression, ElasticNet, BayesianRidge, SGDReg
from xgboost.sklearn import XGBRegressor
from catboost import CatBoostRegressor
from sklearn.ensemble import GradientBoostingRegressor
# validation metrics
from sklearn.metrics import mean_absolute_error, r2_score, explained_variance_score

# Interpretability
import shap
from lime.lime_tabular import LimeTabularExplainer

# For sentiment analysis
nltk.download('vader_lexicon')
from nltk.sentiment.vader import SentimentIntensityAnalyzer
sid = SentimentIntensityAnalyzer()

# ignore warnings
import warnings
warnings.filterwarnings('ignore')
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     /Users/frederikkjaer/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]     /Users/frederikkjaer/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     /Users/frederikkjaer/nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
```

In [2]:

```
### Import raw data
def load_data(rating_filepath, metadata_filepath):
    ratings_df = pd.read_csv(rating_filepath, names = ['item','user','rating','times
    metadata_df = pd.read_json(metadata_filepath, lines=True)
    return ratings_df, metadata_df
```

Then a preparation of the data is performed with the function "prepare_data" from the script "Prepare.py". The purpose of this function is to merge the two datasets and to structure the data. In the function we group on each item of the ratings data such that we get the average rating, standard deviation of rating and the number of ratings. The "category" feature is originally a list of categories with "Grocery and Gourmet Food" as the first element followed by multiple subcategories. Hence we change "category" so it only contains the second element of the list corresponding to the largest subcategory after the main category "Grocery and Gourmet Food".

In [3]:

```
### Function for preparing the data
def prepare_data(ratings_df, metadata_df):
    # create timestamps
    ratings_df['timestamp'] = pd.to_datetime(ratings_df['timestamp'], origin = 'unix
    metadata_df['timestamp'] = pd.to_datetime(metadata_df['date'].apply(str), format

    # drop columns in metadata
    metadata_df = metadata_df.drop(columns=['imageURL','imageURLHighRes'])

    # drop na's and duplicates
    ratings_df = ratings_df.drop_duplicates(keep='first')

    # group ratings_df and merge with metadata, so there is one dataframe with both
    grouped_ratings = ratings_df[['item','rating']].groupby(by='item').agg({'rating'
    grouped_ratings.columns = ['_'.join(col).strip() if col[1] else col[0] for col i
    grouped_ratings = grouped_ratings.rename(columns = {'rating_mean':'avg_rating','
    metadata_df = grouped_ratings.merge(metadata_df, how='outer', left_on='item', ri
    metadata_df['item'].fillna(metadata_df['asin'], inplace=True)
    metadata_df = metadata_df.drop(columns=['asin','date','tech1','tech2','fit'])

    # preprocess price
    metadata_df['price'] =  pd.to_numeric(metadata_df['price'].str.replace('$',''),

    # Fill nan with empty space and use the get_category function
    metadata_df['category'] = metadata_df['category'].fillna('')
    metadata_df['category'] = metadata_df['category'].apply(get_category)

    return metadata_df

# Function to return only the first name in each category variable.
def get_category(row):
    if len(row) > 1:
        category = row[1]
    else:
        category = row
    return category
```

After the data is merged we save it as a new csv file.

In [4]:

```
rating_filepath = 'raw_data/Grocery_and_Gourmet_Food.csv'
metadata_filepath = 'raw_data/meta_Grocery_and_Gourmet_Food.json'

raw_ratings, raw_metadata = load_data(rating_filepath=rating_filepath, metadata_file

metadata_df = prepare_data(raw_ratings, raw_metadata)

# Save the new dataframes to later use.
metadata_df.to_csv('data/metadata_df.csv',index=False)
```

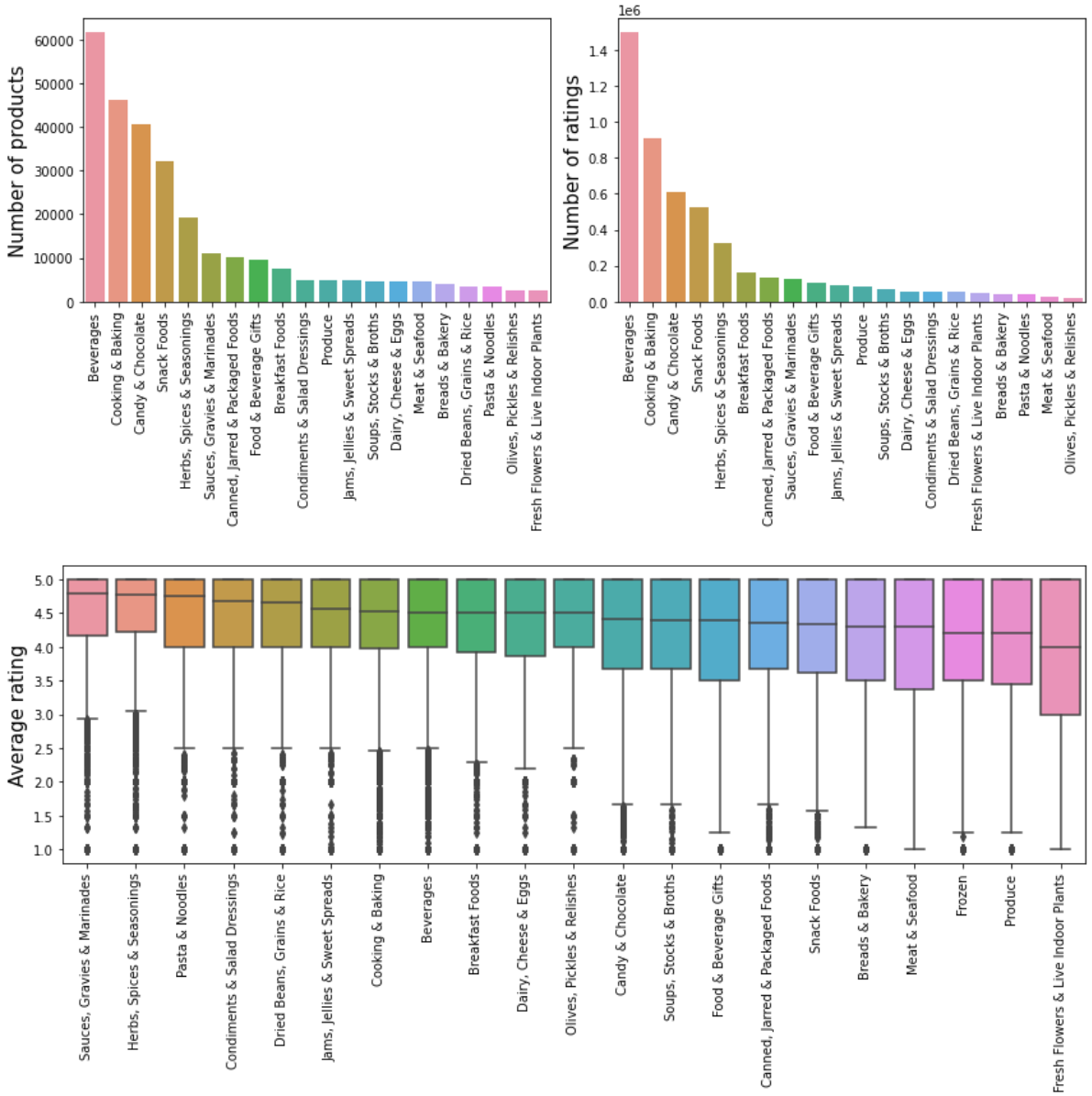
Data Exploration

The idea behind our investigation is to find features of highly rated products and recommend to add the features to low rated products. However, if the products are too different like e.g. chocolate and apples it does not make sense to compare features for these products even if one has a high rating and the other a low rating.

Therefore, we analyze the data by looking at the different product categories and explore the number of products, the number of ratings and variance in the average rating of the products. It is important for our analysis that we have a category with both many products and many ratings. However it is also important that we have something to improve, meaning we need a category with variation in the average rating of products. We will only be looking at the top 20 categories in the exploration. This means that when exploring number of ratings, we will only look at the 20 categories with highest number of ratings.

In [5]:

```
# load data
df = pd.read_csv('data/metadata_df.csv')

# Number of products in each category
def get_category(row, categories):
    if row in categories:
        return row
    else:
        return ''

# We will only look at the top 20 categories
top = 20
categories = df['category'].value_counts().sort_values(ascending=False).index[0:top]

# Copy the dataframe
df_category = df.copy(deep=True)

# Return number of products in each category
df_category['category'] = df_category['category'].apply(lambda row: get_category(row
df_category = df_category[df_category['category'] != '']

# Number of ratings in each category
df_num_ratings = df[['category','num_ratings']].groupby(by=["category"]).sum(["num_r
df_num_ratings = df_num_ratings['num_ratings'].sort_values(ascending=False).reset_ir

# Plot of the number of productts in each category and the number of ratings in each
fig, ((ax1, ax2)) = plt.subplots(1,2, figsize=(12,6))
sns.countplot(x="category", data=df_category, order=categories, ax=ax1)
ax1.set_ylabel('Number of products', fontsize=15)
ax1.set_xlabel('')
ax1.set_xticklabels(categories,rotation=90)
sns.barplot(x="category", y="num_ratings", data=df_num_ratings[0:top], ax=ax2)
ax2.set_ylabel('Number of ratings', fontsize=15)
ax2.set_xlabel('')
ax2.set_xticklabels(df_num_ratings.loc[0:(top-1),'category'].to_list(),rotation=90)
fig.tight_layout()
fig.show()

# Plot with variance of average ratings in each category
categories_union = list(set().union(categories,df_num_ratings.loc[0:top,'category'])
df_mean_avg_rating = df[df['category'].isin(categories_union)].groupby('category').m
categories_union = df_mean_avg_rating.index.to_list()
plt.figure(2, figsize=(12,6))
sns.boxplot(x = 'category', y = 'avg_rating', data = df[df['category'].isin(categori
plt.xticks(rotation=90)
plt.ylabel('Average rating', fontsize=15)
plt.xlabel('')
plt.tight_layout()
plt.show()
```



By investigating the plot with number of ratings and the plot with number of products we find the categories "Beverages", "Cooking & Baking", "Candy & Chocolate" and "Snack Foods" are top 4 in both plots. When we compare these categories with the boxplot showing the variance of the average rating, both "Snack Foods" and "Candy & Chocolate" have a decent spread. The categories "Beverages" and "Cooking and Baking" has a smaller spread as the 25% quantile is higher than for "Candy & Chocolate" and "Snack Foods". Therefore, we want to carry on with either "Candy & Chocolate" or "Snack Foods". Since "Candy & Chocolate" has more products and more ratings, we choose to use the data within the category "Candy & Chocolate" for further analysis.

The data for the chosen category "Candy & Chocolate" is saved as a separate csv file.

In [6]:

```
# Select category
category = 'Candy & Chocolate'
df_cat = df[df['category']==category]

# Save dataframe for category
df_cat.to_csv('data/'+category+'/df_'+category+'.csv',index=False)
```

Data preprocessing

We have now chosen a category to focus on, but the data for this category need to be further processed before it can be used for modelling. In the function "preprocess_price" from the script "Preprocess.py" we are splitting our datasets into a training set and a test set. A lot of product in the candy and chocolate category are missing price. To overcome this issue and still be able to use price as a feature in the modelling, the missing price values are replaced with the average price of the category.

In [7]:

```
# Function for preprocessing the prize where the data also will splitted in a train
def preprocess_price(metadata_df):
    # Drop columns we don't use
    df = metadata_df.drop(columns = ['title','feature','main_cat','similar_item','de
    # Empty columns can't be used for anything
    df = df.dropna(axis=0,subset=['avg_rating','num_ratings','description'])
    # Split the data in a 75 % split train and test set.
    df_train, df_test = train_test_split(df, train_size=0.75, random_state = 42)

    # Below we find the mean price for every cateogry.
    categories = []
    category_means = []
    categories = df_train.category.unique()
    for i in categories:
        temp = df_train[df_train['price'].isna() == False]
        mean_value = temp[temp['category'] == i]['price'].mean()
        category_means.append(mean_value)
    dict = {'categories': categories,'category_means': category_means}
    category_stat_df = pd.DataFrame(dict)
    category_stat_df = category_stat_df.set_index('categories')

    # Next for all NULL values we assign the price to be the mean price in that cate

    df_train['price'] = df_train.apply(lambda row: category_stat_df.loc[row['categor
    df_test['price'] = df_test.apply(lambda row: category_stat_df.loc[row['category'

    # Here we drop the category column.
    columns = df_train.columns
    if 'category' in columns:
        df_train = df_train.drop(columns = ['category'])
        df_test = df_test.drop(columns = ['category'])
    if 'orig category' in columns:
        df_train = df_train.drop(columns = ['orig category'])
        df_test = df_test.drop(columns = ['orig category'])
    return df_train, df_test
```

The training set and the test set are now further processed in the "preprocess_data" function from the script "Preprocess.py where the features are cleaned and prepared for modelling.

In the function we clean the features "also_buy", "also_view", "rank" and "description". For each product, the two features "also_buy" and "also_view" contain a list of other products that are often bought or viewed together with the product. These features are converted to the length of the list, which is the number of products that they are often bought or viewed together with. For the feature "rank", which contains the sales rank information of the products, all characters are removed besides the sales rank number which is converted to an integer value. Note: If the sales rank is low, the product is selling a lot, if the sales rank is high, the product is not selling a lot.

In the "description" feauture we clean text from html code, remove punctuation and stop words etc. When removing stop words from "description" we only use the English stop words. However when we later on perform LDA on the data we observe there are words that either occur in multiple topics or doesn't make sense

in the topic, and therefore we add them to stop words iteratively.

In [8]:

```
# Function to preprocess the train and test dataset
def preprocess_data(df_train, df_test):
    # get number of also_buy
    df_train['also_buy'] = df_train['also_buy'].fillna('').apply(get_number_also_buy
    df_test['also_buy'] = df_test['also_buy'].fillna('').apply(get_number_also_buy)

    # sales rank information
    df_train['rank'] = df_train['rank'].apply(get_rank).str.replace(',','').str.extr
    df_train['rank'] = pd.to_numeric(df_train['rank'], errors = 'coerce').fillna(0).
    df_test['rank'] = df_test['rank'].apply(get_rank).str.replace(',','').str.extrac
    df_test['rank'] = pd.to_numeric(df_test['rank'], errors = 'coerce').fillna(0).ap

    # Fill nan values for price data
    # get number of also_view
    df_train['also_view'] = df_train['also_view'].fillna('').apply(get_number_also_b
    df_test['also_view'] = df_test['also_view'].fillna('').apply(get_number_also_buy

    # Clean description
    df_train['description'] = df_train['description'].apply(get_description)
    # Drop rows with no information
    df_train = df_train.dropna(axis = 0, subset=['description'])
    # Make it a string and clean html text
    df_train['description'] = df_train['description'].apply(str)
    df_train['description'] = df_train['description'].str.replace('\n', '')
    df_train['description'] = df_train[['description']].applymap(lambda text: Beauti
    # Perform text processing where stop words are removed etc.
    df_train['description'] = df_train['description'].apply(text_processing)

    # Do the same for test dataset
    df_test['description'] = df_test['description'].apply(get_description)
    df_test = df_test.dropna(axis = 0, subset=['description'])
    df_test['description'] = df_test['description'].apply(str)
    df_test['description'] = df_test['description'].str.replace('\n', '')
    df_test['description'] = df_test[['description']].applymap(lambda text: Beautifu
    df_test['description'] = df_test['description'].apply(text_processing)
    return df_train, df_test

# Number of also bougtht products
def get_number_also_buy(row):
    number = len(row)
    return number
# Get the brand
def get_brand(row, brands):
    if row in brands:
        return row
    else:
        return 'Other'
# Get the rank from list
def get_rank(row):
    if isinstance(row, list):
        if len(row) > 0:
            return row[0]
        else:
            return ''
    else:
        return row

# Use only rows with list of information else nan
def get_description(row):
```

```
    if isinstance(row, list):
        if len(row)>0:
            return row
        else:
            return np.nan
    else:
        return row

# Function used to clean text data
def text_processing(text):
    # remove punctuation
    text = "".join([c for c in text
        if c not in string.punctuation])
    # lowercase
    text = "".join([c.lower() for c in text])
    # stemming / lematizing (optional)
    text = " ".join([lemmatizer.lemmatize(w) for w in text.split()])
    # remove stopwords
    text = " ".join([w for w in text.split()
        if w not in Stop_Words])
    return text
    # Words from already trained lda model. The words are removed because they eithe


# Words from already trained lda model. The words are removed because they either oc
category = 'Candy & Chocolate'
if category == 'Candy & Chocolate':
    Stop_Words = _stop_words.ENGLISH_STOP_WORDS.union(['chocolate','supplement','coc
                                                       'health','milk','intended','
                                                       'evaluated','disease','treat
                                                       'just','make', 'artificial']
elif category == 'Snack Foods':
    Stop_Words = _stop_words.ENGLISH_STOP_WORDS.union(['snack','food','fda','flavor'
elif category == 'Beverages':
    Stop_Words = _stop_words.ENGLISH_STOP_WORDS.union(['tea','coffee','water','cup',
```

The training and test set are now almost fully processed, but we still need to handle categorical data. The "brand" feature contains almost 15000 different brands and it would not make any sense to one-hot-encode a feature with this many unique values, as we would get an enormous dataset. We will instead perform a principle components analysis (PCA) of the encoded dataset in order to decompose this enormous dataset. Calculating the variance explained by the components in the new space, we found that two components was sufficient to explain 99% of the variance. Using these two components describing the new space of the data, we defined 5 clusters in the data using the K-means algorithm. We hope that enough information about the brands are stored in the decomposition and through that in the clusters, so we carry some of this information over in the new dataset through the clusters.

Finally, we assign the clusters to the preprocessed data and one-hot-encode it, before saving the new dataset which is to be used for modelling.

In [9]:

```
# First we read the "prepared" data and the proper category found in the "explorator
metadata_df = pd.read_csv('data/'+category+'/df_'+category+'.csv')

#We clone the category, as we will need this in order to preprocess price, but we wi
# when we create dummy data
#metadata_df['orig category'] = metadata_df['category']
dummy_df = pd.get_dummies(metadata_df, columns=['brand'])
dummy_df = dummy_df.drop(columns=['item'])
# We drop brand as we can't have non numerical values.
metadata_df = metadata_df.drop(columns=['brand'])

# We apply the preprocess price function, which handles some of the preprocess funct
df_train, df_test = preprocess_price(metadata_df)
df_train_dummy, df_test_dummy = preprocess_price(dummy_df)

# Next the second part of the preprocess will be applied.
df_train, df_test = preprocess_data(df_train, df_test)
df_train_dummy, df_test_dummy = preprocess_data(df_train_dummy, df_test_dummy)

## We drop description when we do pca on the data as it is non-numerical. Also we dr
# Avg. rating, and it seems unlikely that std_rating can be gathered in a situation
df_train_dummy = df_train_dummy.drop(columns = ['description','std_rating'])
df_test_dummy = df_test_dummy.drop(columns = ['description','std_rating'])

## Next we decompose our data.
pca = PCA(n_components=100)
pca.fit(df_train_dummy)

## The explained variance have been calculated, and it seems only 2 components are r
pca = PCA(n_components=3).fit(df_train_dummy)
pca_values = pca.fit_transform(df_train_dummy)

# ### Plot the pca values in order to identify if there are any clusters:
# fig = plt.figure()
# ax = fig.add_subplot(projection='3d')
# ax.scatter(pca_values[:,0], pca_values[:,1], pca_values[:,2])
# ax.set_xlabel('X Label')
# ax.set_ylabel('Y Label')
# ax.set_zlabel('Z Label')

# plt.show()

# plt.scatter(pca_values[:,0], pca_values[:,1])
# plt.show()

## Then we create 5 clusters using KMeans.
kmeans = KMeans(n_clusters=2).fit(pca_values)

## We assign these clusters to our original (preprocessed) data.
df_train['cluster'] = kmeans.labels_
pca_values_test = pca.transform(df_test_dummy)
df_test['cluster'] = kmeans.predict(pca_values_test)

## We one hot encode the clusters, so that any model trained on the data, wan't assu
df_train = pd.get_dummies(df_train, columns = ['cluster'])
df_test = pd.get_dummies(df_test, columns = ['cluster'])

# ensure same shape of train and test
if df_train.shape[1] != df_test.shape[1]:
```

```
    setdiff = set(df_train.columns).difference(set(df_test.columns))
    for name in setdiff:
        df_test[name] = np.zeros(df_test.shape[0])
        df_test = df_test.astype({name:'int'})

# order columns in test set
df_test = df_test[df_train.columns]
```

We save the training and test set as csv files, since these datasets are now ready for modelling except for the feature "description" which we will perform LDA on in the next step.

In [10]:

```
df_train.to_csv('data/' + category + '/df_train.csv',index=False)
df_test.to_csv('data/' + category + '/df_test.csv',index=False)
```

LDA Modeling

After the preprocessing of the feature "description", we will now make a last preprocess step in order to make every single feature numeric. We will decompose the product descriptions using the Latent Dirichlet Allocation to find topics - hopefully containing key product features. We use the function "train_lda" from the "train_lda.py" script to perform the topic modelling and tuning the hyperparameter, the number of topics.

In [11]:

```
# set random seed
np.random.seed(42)

#%% Functions
def get_len(text):
    if text != text:
        return 0
    elif isinstance(text, float):
        return 1
    else:
        return len(text)

# Function where LDA is trained on both the trainf and test. Both datasets are retur
def train_lda(df_train, df_test, gridsearch, text):
    docs_train = df_train[text].values
    #docs_train = np.vectorize(docs_train)
    #docs_train = [word_tokenize(d) for d in docs_train]
    docs_test = df_test[text]
    #docs_test = [word_tokenize(d) for d in docs_test]
    count_vect = CountVectorizer()
    bow_counts_train = count_vect.fit_transform(docs_train)
    bow_counts_test = count_vect.transform(docs_test)

    cv_matrix = count_vect.fit_transform(docs_train)
    gridsearch.fit(cv_matrix)

    ## Save the best model
    best_lda = gridsearch.best_estimator_
    # What did we find?
    print("Best Model's Params: ", gridsearch.best_params_)

    # Train LDA with best params
    n_topics = gridsearch.best_params_['n_components']
    ld = gridsearch.best_params_['learning_decay']

    lda = LatentDirichletAllocation(n_components=n_topics, max_iter=10,
                                    learning_method='batch',
                                    learning_offset=10.,
                                    learning_decay=ld)

    X_train_lda = lda.fit_transform(bow_counts_train)
    X_test_lda = lda.transform(bow_counts_test)

    return X_train_lda, X_test_lda, lda, count_vect, best_lda, n_topics


# Function to print the top words in a topic
def print_top_words(model, feature_names, n_top_words):
    norm = model.components_.sum(axis=1)[:, np.newaxis]
    for topic_idx, topic in enumerate(model.components_):
        print(80 * "-")
        print("Topic {}".format(topic_idx))
        for i in topic.argsort()[:-n_top_words - 1:-1]:
            print("{:.3f}".format(topic[i] / norm[topic_idx][0])
                  + '\t' + feature_names[i])

# Function to visualize the topics in a wordcloud
def visualize_topics(lda, count_vect, terms_count):
    terms = count_vect.get_feature_names()
```

```
    for idx, topic in enumerate(lda.components_):
        title = 'Topic ' + str(idx+1)
        abs_topic = abs(topic)
        topic_terms = [[terms[i],topic[i]] for i in abs_topic.argsort()[:-terms_cour
        topic_terms_sorted = [[terms[i], topic[i]] for i in abs_topic.argsort()[:-te
        topic_words = []
        for i in range(terms_count):
            topic_words.append(topic_terms_sorted[i][0])
            # print(','.join( word for word in topic_words))
            # print("")
            dict_word_frequency = {}
        for i in range(terms_count):
            dict_word_frequency[topic_terms_sorted[i][0]] = topic_terms_sorted[i][1]
            wcloud = wordcloud.WordCloud(background_color="white",mask=None, max_wor
            max_font_size=60,min_font_size=10,prefer_horizontal=0.9,
            contour_width=3,contour_color='black')
            wcloud.generate_from_frequencies(dict_word_frequency)
        plt.imshow(wcloud, interpolation='bilinear')
        plt.axis("off")
        plt.title(title, fontsize=20)
        # plt.savefig("Topic#"+str(idx+1), format="png")
        plt.show()
    return

#%% RUN LDA

# Get data
category = 'Candy & Chocolate'
train_path = 'data/' + category + '/df_train.csv'
test_path = 'data/' + category + '/df_test.csv'
df_train = pd.read_csv(train_path)
df_test = pd.read_csv(test_path)
df_train = df_train.dropna(axis=0,subset=['description'])
df_test = df_test.dropna(axis=0,subset=['description'])

# Options to tune hyperparamets in LDA model
# Beware it will try *all* of the combinations, so it'll take ages
search_params = {'n_components': [4, 6, 8], 'learning_decay': [0.5, .7]}

# Set up LDA with the options we'll keep static
model = LatentDirichletAllocation(learning_method='online',
                                  max_iter=5,
                                  random_state=0)

# Try all of the options
gridsearch = GridSearchCV(model,
                          param_grid=search_params,
                          n_jobs=-1,
                          verbose=2,
                         )
count_vect = CountVectorizer()


# Run LDA on description with tuned parameters
X_train_lda, X_test_lda, lda, count_vect, best_lda, n_topics = train_lda(df_train, c

# Visualize topics as wordclouds
visualize_topics(lda, count_vect, 25)

# Merge df with lda
lda_list = []
```

```
for i in range(n_topics):
    lda_list.append('lda'+str(i+1))
X_train_lda_df = pd.DataFrame(X_train_lda, columns = lda_list)
X_test_lda_df = pd.DataFrame(X_test_lda, columns = lda_list)
df_train_lda = df_train.merge(X_train_lda_df, left_index=True, right_index=True)
df_test_lda = df_test.merge(X_test_lda_df, left_index=True, right_index=True)

# Save merged data + model
today = date.today()
df_train_lda.to_csv('data/' + category + '/df_train_4_components_lda.csv',index=Fals
df_test_lda.to_csv('data/' + category + '/df_test_4_components_lda.csv',index=False)
filename = 'models/'+category+'/lda_model_4_components'+str(today)+'.sav'
pickle.dump(lda, open(filename, 'wb'))
filename = 'models/'+category+'/count_vect_model_'+str(today)+'.sav'
pickle.dump(count_vect, open(filename, 'wb'))
```

```
Fitting 5 folds for each of 6 candidates, totalling 30 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent wor
kers.
```

We can conclude, that for our dataset, it is the most effecient, to have a few broad topics. This may also make sense, as our data have already been categorised into a smaller dataset being Chocolate & Candy. Maybe if we hadn't already made a subset of the Grocery & Gourmet Foods dataset, more topics would have been necessary to describe each category. The LDA values have been concatenated with the preprocessed dataframe, so every data transformation done so far can be combined and used with a regression model, in order to make predictions.

The final LDA model was found by making a parameter grid of possible hyperparameters, where we used cross validation to find the best combination.

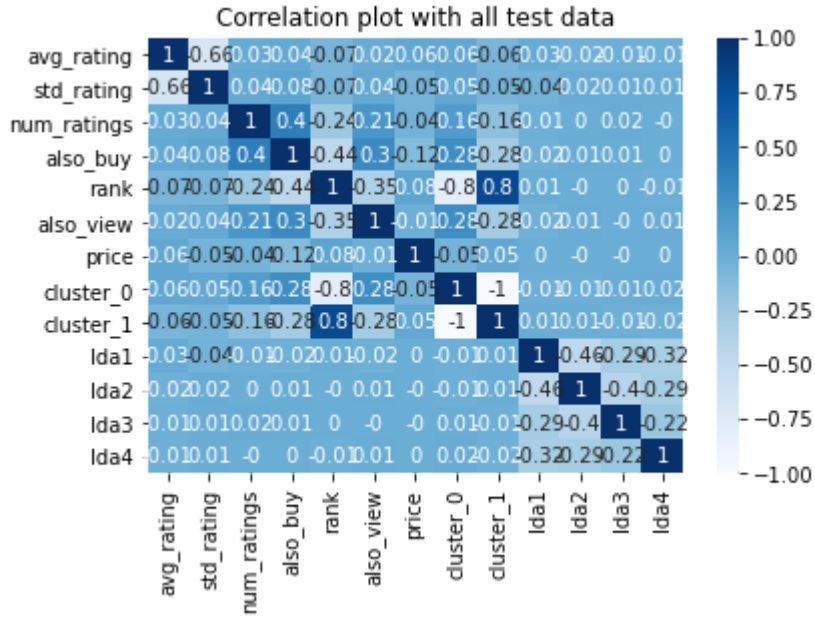
For each topic, the top 25 words are plotted using a wordcloud plot. Words being replicated in multiple topics or words irrelevant for product recommendations are added to stopwords and the LDA is run again to create topics of higher quality.

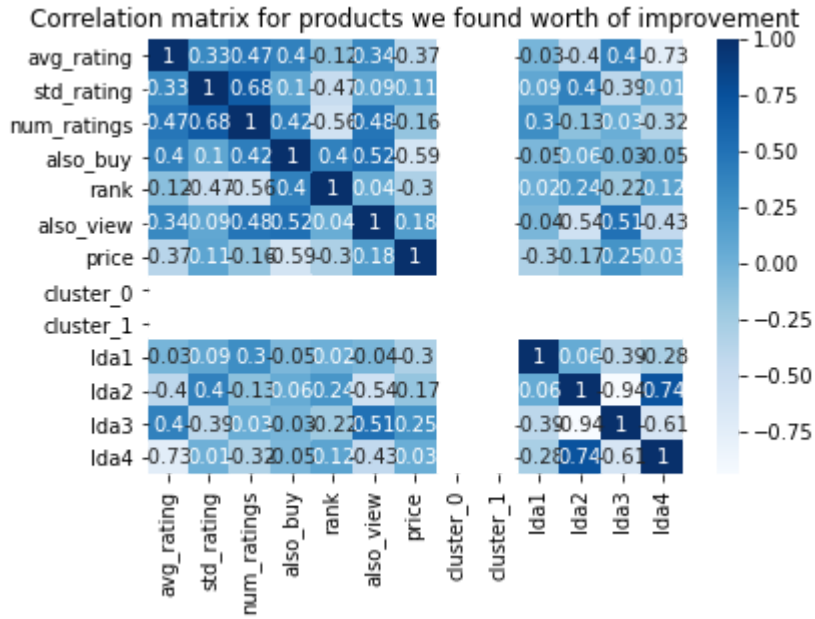
Exploratory analysis post data preprocessing and LDA

Next we went back to explore our data in order to find a product which have a lot of sales, but a low average rating. This product will then be of focus, since there is a lot of potential profit if such product is improved. The product will be referred to as the product to enhance. The code for this exploration is done in the script "Exploratort_Post_LDA.py" and seen below.

In [12]:

```
## Define the category to investigate
category = 'Candy & Chocolate'

##Load data
df_test = pd.read_csv('data/'+category+'/df_test_4_components_lda.csv')

## Drop description as we assume we have gotten the most out of this variable using
df_test = df_test.drop(columns=['description'])

## Create a correlation plot?
sns.heatmap(df_test.corr().round(2), cmap='Blues', annot=True)\
   .set_title('Correlation matrix')
plt.title('Correlation plot with all test data')
plt.show()


### find the right product to enchance

## We find subsets of the data, to limit the amount of products to check.

## Defining that subset

products_to_enhance = df_test[df_test['rank'] != 0]

products_to_enhance = products_to_enhance[products_to_enhance['rank']<=products_to_e

# Finding the 75% quantile of the 75% quantile of the number of ratings
products_to_enhance = products_to_enhance[products_to_enhance['num_ratings']>=produc

# Finding the 25% quantile of avg. ratings of the subset.

products_to_enhance = products_to_enhance[products_to_enhance['avg_rating'] <= produ

# Finding the 25% quantile of standard deviations for ratings in the subset
products_to_enhance = products_to_enhance[products_to_enhance['std_rating'] <= produ

# Finding the 25% quantile of sales rank.
products_to_enhance = products_to_enhance[products_to_enhance['rank']<=products_to_e
product_to_enhance = products_to_enhance.loc[products_to_enhance['rank'] == min(prod

sns.heatmap(products_to_enhance.corr().round(2), cmap='Blues', annot=True)\
   .set_title('Correlation matrix for products we found worth of improvement')
plt.show()

## We find

# Find the 200 best rated products

indexes = df_test['avg_rating'].sort_values(ascending=False)[0:199].index.to_list()
```





In the figures above we show the correlation matrices for the entire dataset, and for the products we have identified as especially interesting. This is done to see if there is dependecies which can be exploided to enhance the products.

We see that of the 10 products we identified as the most worth of improvement, every product is in the first cluster, which is also why a correlation can't be computed and therefore appear blank in the figure.

In [13]:

```
print("After exploratory analysis, we have identified the following subset of produc
products_to_enhance
```

```
After exploratory analysis, we have identified the following subset of
products worthy of improvement.
```

Out[13]:

	item	avg_rating	std_rating	num_ratings	also_buy	rank	also_view	price	clus
738	B00ZPQ7GNS	4.128205	1.316595	117.0	952	9586	2	6.90	
925	B002ZQ8F2M	3.977273	1.373024	176.0	616	13909	2	11.71	
1246	B00FK49EX0	4.058020	1.487406	293.0	672	4419	2	16.86	
3054	B000JEGQ9G	4.104137	1.446868	701.0	1316	1228	756	15.00	
3631	B01D88CLNE	3.938462	1.367901	65.0	56	2348	126	28.99	
4333	B00QT29ZFY	3.727273	1.335798	44.0	784	11833	2	16.72	
5876	B0161VHGN0	4.023256	1.405709	129.0	1190	17765	2	13.50	
6812	B00176CO6O	4.043478	1.344212	69.0	1190	20965	798	18.58	

In [14]:

```
print("The product we have chosen to improve is: ")
product_to_enhance
```

```
The product we have chosen to improve is:
```

Out[14]:

	item	avg_rating	std_rating	num_ratings	also_buy	rank	also_view	price	cluste
3054	B000JEGQ9G	4.104137	1.446868	701.0	1316	1228	756	15.0	

We have now found a product which could be interesting to enchance.

Also we found the indexes of the best 200 rated products. We will use these indexes, when we later will try and interpret the decisioning from our regression model. This will help us find the most important features, which customers values when choosing a product to buy.

Building regression models

After preprocessing data and applying LDA on "description", we are ready to build a regression model with average rating as response and the explanatory variables: the number of often bought products, the number of often viewed products, sales rank, price, description topics and brand clusters.

The function "train_regression_models" from the script "TrainModel.py fits 7 different regression models to the training data and returns the model that performs best on the test data in terms of mean absolute error (MAE).

In [15]:

```
# set random seed
np.random.seed(42)

# load data and select specific category
category = 'Candy & Chocolate'
df_train = pd.read_csv('data/' + category + '/df_train_4_components_lda.csv')
df_test = pd.read_csv('data/' + category + '/df_test_4_components_lda.csv')

df_train = df_train.drop(columns=['description','std_rating','item'])
df_test = df_test.drop(columns=['description','std_rating','item'])
df_train = df_train.dropna()
df_test = df_test.dropna()

# prepare for training
y_train = df_train['avg_rating']
y_test = df_test['avg_rating']
X_train = df_train.drop(columns=['avg_rating'])
X_test = df_test.drop(columns=['avg_rating'])

# Function to train 7 regression models
def train_regression_models(X_train, X_test, y_train, y_test):
    # Linear Regression
    linear_regression = LinearRegression().fit(X_train, y_train)
    y_linear_regression = linear_regression.predict(X_test)
    MAE_linear_regression = mean_absolute_error(y_test, y_linear_regression)
    r2_linear_regression = r2_score(y_test, y_linear_regression)
    var_linear_regression = explained_variance_score(y_test, y_linear_regression)
    print("----------------------")
    print("Linear Regression: ")
    print("MAE ", MAE_linear_regression)
    print("R2 ", r2_linear_regression)
    print("Explained variance ", var_linear_regression)
    print("----------------------")

    # XGBoost Regressor
    xgb_regressor = XGBRegressor().fit(X_train, y_train)
    y_xgb_regressor = xgb_regressor.predict(X_test)
    MAE_xgb_regressor = mean_absolute_error(y_test, y_xgb_regressor)
    r2_xgb_regressor = r2_score(y_test, y_xgb_regressor)
    var_xgb_regressor = explained_variance_score(y_test, y_xgb_regressor)
    print("----------------------")
    print("XGBoost Regressor: ")
    print("MAE ", MAE_xgb_regressor)
    print("R2 ", r2_xgb_regressor)
    print("Explained variance ", var_xgb_regressor)
    print("----------------------")

    # CatBoost Regressor
    catboost_regressor = CatBoostRegressor(allow_writing_files=False).fit(X_train, y
    y_catboost_regressor = catboost_regressor.predict(X_test)
    MAE_catboost_regressor = mean_absolute_error(y_test, y_catboost_regressor)
    r2_catboost_regressor = r2_score(y_test, y_catboost_regressor)
    var_catboost_regressor = explained_variance_score(y_test, y_catboost_regressor)
    print("----------------------")
    print("CatBoost Regressor: ")
    print("MAE ", MAE_catboost_regressor)
    print("R2 ", r2_catboost_regressor)
    print("Explained variance ", var_catboost_regressor)
    print("----------------------")
```

```
# Stochastic Gradient Descent Regression
sgd_regressor = SGDRegressor().fit(X_train, y_train)
y_sgd_regressor = sgd_regressor.predict(X_test)
MAE_sgd_regressor = mean_absolute_error(y_test, y_sgd_regressor)
r2_sgd_regressor = r2_score(y_test, y_sgd_regressor)
var_sgd_regressor = explained_variance_score(y_test, y_sgd_regressor)
print("----------------------")
print("Stochastic Gradient Descent Regression: ")
print("MAE ", MAE_sgd_regressor)
print("R2 ", r2_sgd_regressor)
print("Explained variance ", var_sgd_regressor)
print("----------------------")

# Elastic Net Regression
elastic_net = ElasticNet().fit(X_train, y_train)
y_elastic_net = elastic_net.predict(X_test)
MAE_elastic_net = mean_absolute_error(y_test, y_elastic_net)
r2_elastic_net = r2_score(y_test, y_elastic_net)
var_elastic_net = explained_variance_score(y_test, y_elastic_net)
print("----------------------")
print("Elastic Net Regression: ")
print("MAE ", MAE_elastic_net)
print("R2 ", r2_elastic_net)
print("Explained variance ", var_elastic_net)
print("----------------------")

# Bayesian Ridge Regression
bayesian_ridge = BayesianRidge().fit(X_train, y_train)
y_bayesian_ridge = bayesian_ridge.predict(X_test)
MAE_bayesian_ridge = mean_absolute_error(y_test, y_bayesian_ridge)
r2_bayesian_ridge = r2_score(y_test, y_bayesian_ridge)
var_bayesian_ridge = explained_variance_score(y_test, y_bayesian_ridge)
print("----------------------")
print("Bayesian Ridge Regression: ")
print("MAE ", MAE_bayesian_ridge)
print("R2 ", r2_bayesian_ridge)
print("Explained variance ", var_bayesian_ridge)
print("----------------------")

# Gradient Boosting Regression
gb_regressor = GradientBoostingRegressor().fit(X_train, y_train)
y_gb_regressor = gb_regressor.predict(X_test)
MAE_gb_regressor = mean_absolute_error(y_test, y_gb_regressor)
r2_gb_regressor = r2_score(y_test, y_gb_regressor)
var_gb_regressor = explained_variance_score(y_test, y_gb_regressor)
print("----------------------")
print("Gradient Boosting Regression: ")
print("MAE ", MAE_gb_regressor)
print("R2 ", r2_gb_regressor)
print("Explained variance ", var_gb_regressor)
print("----------------------")

MAEs = [MAE_linear_regression, MAE_xgb_regressor, MAE_catboost_regressor, MAE_sg
        MAE_elastic_net, MAE_bayesian_ridge, MAE_gb_regressor]
R2s = [r2_linear_regression, r2_xgb_regressor, r2_catboost_regressor, r2_sgd_reg
        r2_elastic_net, r2_bayesian_ridge, r2_gb_regressor]
models = [linear_regression, xgb_regressor, catboost_regressor, sgd_regressor, e
        bayesian_ridge, gb_regressor]
names = ['linear_regression', 'xgb_regressor', 'catboost_regressor', 'sgd_regres
        'bayesian_ridge', 'gb_regressor']
```

```
    best_idx = np.argmin(MAEs)
    return models[best_idx], names[best_idx]
```

For all model runs, either catboost regressor or gradient boost regressor performed best. Thus we have created a function called "tune_model" from the cript "TrainModel.py" that are tuning the hyperparatemers of these two regression models. The tuning of the models is cumbersome due to computation time, why we have chosen to keep the dimension of the parameter space low. After tuning we saw little to no improvement of the performance, why we decided to go with the simpler untuned model.

In [16]:

```
# Function to tune the best model where the two best are catboost or gb_regressor
def tune_model(model, name, X_train, y_train):
        if name == 'catboost_regressor':
                # tune parameters of catboost
                parameters = {'depth' : [5, 10, 15],
                              'learning_rate' : [0.02, 0.03]}
                # Perform gridsearch with parameters
                Grid_CBC = GridSearchCV(estimator=model, param_grid=parameters, cv=5
                Grid_CBC.fit(X_train, y_train)
                depth = Grid_CBC.best_params_['depth']
                learning_rate = Grid_CBC.best_params_['learning_rate']
                catboost_regressor = CatBoostRegressor(allow_writing_files=False, de
                regressor = catboost_regressor
                parameters = Grid_CBC.best_params_
        elif name == 'gb_regressor':
                # tune parameters of gradient boost regressor
                parameters = {'max_depth' : [5, 10, 15],
                              'learning_rate' : [0.02, 0.03]}
                Grid_GBR = GridSearchCV(estimator=model, param_grid=parameters, cv=5
                Grid_GBR.fit(X_train, y_train)
                depth = Grid_GBR.best_params_['depth']
                learning_rate = Grid_GBR.best_params_['learning_rate']
                gb_regressor = GradientBoostingRegressor(depth=depth, learning_rate=
                regressor = gb_regressor
                parameters = Grid_GBR.best_params_
        return parameters, regressor

# train and tune model
model, name = train_regression_models(X_train, X_test, y_train, y_test)
#params, tuned_model = tune_model(model, name, X_train, y_train)

# validate model
predictions = model.predict(X_test)

print("The best model is: ", model)
#print("The predictions using the best performing models are: ", predictions)
#print("The true values are: ", y_test)
#print("This gives the difference between predictions and true values: ", predictior
print("The MAE of the model is: ", mean_absolute_error(y_test, predictions))

today = date.today()
# save model
filename = 'models/'+category+'/best_performing_model_4_'+str(today)+'.sav'
pickle.dump(model, open(filename, 'wb'))
```

```
----------------------
Linear Regression:
MAE  0.79495348496628
R2  0.0075648087597244285
Explained variance  0.008240432075987902
----------------------
----------------------
XGBoost Regressor:
MAE  0.8035146221746664
R2  -0.043488120546474995
Explained variance  -0.042683746913409815
----------------------
----------------------
```

```
CatBoost Regressor:
MAE  0.7952293380926716
R2  -0.012619540829346576
Explained variance  -0.01202332798998551
----------------------
----------------------
Stochastic Gradient Descent Regression:
MAE  6.088640271444029e+19
R2  -5.752192415474331e+39
Explained variance  -2.267051582483962e+39
----------------------
----------------------
Elastic Net Regression:
MAE  0.7956908617038672
R2  0.006767538743922996
Explained variance  0.007451871279382627
----------------------
----------------------
Bayesian Ridge Regression:
MAE  0.794846722022595
R2  0.007122975323912484
Explained variance  0.007808794382174411
----------------------
----------------------
Gradient Boosting Regression:
MAE  0.78950826360244
R2  0.013999467227412521
Explained variance  0.014626480610017811
----------------------
The best model is:  GradientBoostingRegressor()
The MAE of the model is:  0.78950826360244
```

The Gradient Boosting Regressor model have been saved being the best model, so we can load it for future testing.

Interpretability of the models

After training the model, we are ready to interpret the model to find out what features are important and how they relate to the product ratings. The interpretability of the models was done in the notebook "Interpretability.ipynb".

In [17]:

```
category = 'Candy & Chocolate' # get category

# load best model
today = date.today()
model = pd.read_pickle('models/'+category+'/best_performing_model_4_2022-05-11.sav')

# load data
df_train = pd.read_csv('data/' + category + '/df_train_4_components_lda.csv')
df_test = pd.read_csv('data/' + category + '/df_test_4_components_lda.csv')
df_train = df_train.drop(columns=['description','std_rating','item'])
df_test = df_test.drop(columns=['description','std_rating','item'])
df_train = df_train.dropna()
df_test = df_test.dropna()
X_train = df_train.drop(columns='avg_rating')
y_train = df_train['avg_rating']
X_test = df_test.drop(columns='avg_rating')
y_test = df_test['avg_rating']
```

Global interpretability

We start by computing the shap values of the test set and do a shap summary plot.

In [18]:

```
# Get shap values
explainer = shap.TreeExplainer(model)
shap_values = explainer.shap_values(X_test)
shap_values_pandas = pd.DataFrame(shap_values)
print('The overall mean (expected value): ', explainer.expected_value)
shap_values_pandas.head()
```

```
The overall mean (expected value):  [4.09064121]
```

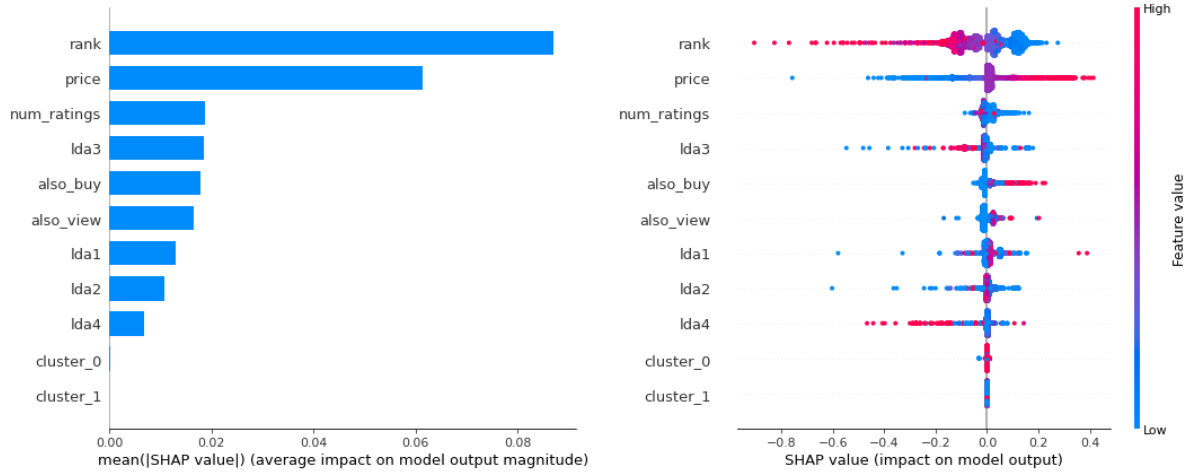
Out[18]:

	0	1	2	3	4	5	6	7	
0	-0.006524	-0.009979	0.007127	-0.011898	0.004759	0.000064	6.248216e-07	0.010843	0.007
1	-0.012110	-0.008913	0.030642	0.019521	0.004743	0.000064	6.248216e-07	0.008687	0.003
2	0.034681	0.018880	0.121310	-0.008453	0.011598	0.000064	6.248216e-07	0.002676	-0.002
3	0.026406	-0.018013	-0.046154	-0.016384	0.002941	0.000064	2.030670e-06	-0.004734	0.001
4	-0.023033	-0.016157	0.119784	0.027026	-0.096953	0.000064	6.248216e-07	0.009484	0.016

The above matrix of dimension (test samples) x (number of features) shows the shap values. The sum of each row gives the difference from the overall mean (the expected value).

In [19]:

```
# shap summary plots
plt.figure(figsize=(15,6))
plt.subplot(1,2,1)
shap.summary_plot(shap_values, X_test, plot_type='bar', show=False, plot_size=None)
plt.subplot(1,2,2)
shap.summary_plot(shap_values, X_test, show=False, plot_size=None)
plt.tight_layout()
plt.show()
```



The above figures show a summary of how the features impact the predictions. The right summary plot shows how the features impact the predictions and the left plot shows how the features impact the predictions in magnitude.

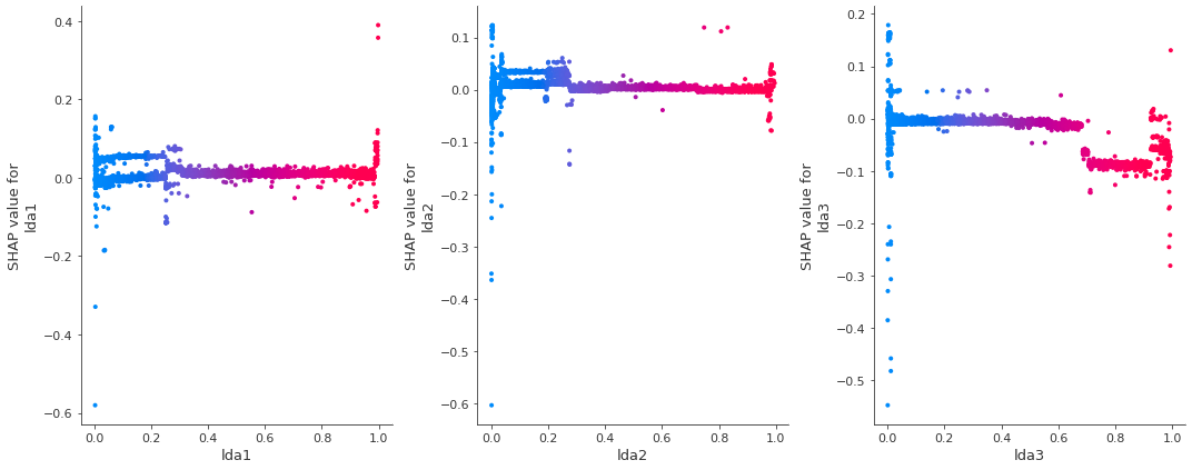
From both plots, it is seen that the features with the highest impact on the average rating are sales rank and price. Looking at the right plot we can see that low values of sales rank increase the average rating for products and high values decrease the average rating. For price it is the opposite.

The lda topic 3 and num_ratings also have an impact, although of small magnitude. Looking at the right plot it is difficult to conclude which way high and low values affect the average rating, and we therefore need a dependence plot to futher inspect. The lda topic 1, 2, 4 and the clusters clusers do not seem to have much impact on average rating and thus we could consider removing the clusters from the model.

We will take a deeper dive into what impact topic 3 has on our model.

In [20]:

```
# Partial dependence plot of relevant variables
fig, axes = plt.subplots(1,3,figsize=(15,6))
shap.dependence_plot(ind='lda1', interaction_index='lda1',shap_values=shap_values, f
shap.dependence_plot(ind='lda2', interaction_index='lda2',shap_values=shap_values, f
shap.dependence_plot(ind='lda3', interaction_index='lda3',shap_values=shap_values, f
plt.tight_layout()
plt.show()
```



Partial dependence plots show shap values against feature values. In the rightmost plot, we can see a small relationship between the shap values and the lda3 values. It can be seen that the average rating becomes a bit smaller when the lda3 value is high. This could be understood as that the words included in topic 3 is bad for the rating of products. Therefore we could highlight that Amazon should be on alert for using words like these in their descriptions of their products, but we can't make a recommendation to completely avoid them, as there isn't a strong relationsship.

In topic 3 we observed that of the 25 most important/frequent words some of the included words were oil, syrup, lecithin, acid, butter, oil, palm, fat, emulsifier. All words that have an unhealthy association. As it is in the descriptions of the products, there may be law requirements, that means that due to legal resons, they must be included. But if possible, this analysis may be an argument to exclude them, as they might have a negative association.

In [21]:

```
# shap force plot of 200 products with high ratings
shap.initjs()
shap.force_plot(explainer.expected_value, shap_values[indexes,:], X_test.iloc[indexe
```

Out[21]:

Visualization omitted, Javascript library not loaded!
Have you run `initjs()` in this notebook? If this notebook was from another user you must also trust this notebook (File -> Trust notebook). If you are viewing this notebook on github the Javascript has been stripped for security. If you are using JupyterLab this error is because a JupyterLab extension has not yet been written.

The above figure shows how the features impact the predictions of the 200 products in the test data with highest ratings. Blue feature values are lowering the prediction and red feature values are increasing the prediction value. We see that it is mostly the rank and price feature, that dominates the average rating prediction, but also for some products like number 62 on the plot above, it can be seen that the model predicted a lower rating than then mean rating, partially duo to a high lda3 value, meaning that the description includes some of the words, that we have advised to be avoided.

Local interpretability

Here we look at how the features influence the prediction of the product to enhance.

In [22]:

```
# Get the index of the product to enhance:
product_to_enhance = product_to_enhance.index

# shap force plot
shap.initjs()
shap.force_plot(explainer.expected_value, shap_values[product_to_enhance,:], X_test.
```

Out[22]:

Visualization omitted, Javascript library not loaded!
Have you run `initjs()` in this notebook? If this notebook was from another user you must also trust this notebook (File -> Trust notebook). If you are viewing this notebook on github the Javascript has been stripped for security. If you are using JupyterLab this error is because a JupyterLab extension has not yet been written.

Here we clearly see that for the product we have found with a large potential to improve, the lda3 value is quite high, meaning that the description consists of a lot of the words we suggested to aviod. This have resulted in a prediction of a lower average rating. Sadly we are not able to validate this result as we do not have a product, which have all the same features, but do not have the identified words frequent in topic 3 in the description. Therefore our hypothesis can't really be confirmed.

In [23]:

```
# lime explainer
explainer = LimeTabularExplainer(X_test.values,mode='regression',feature_names = X_t
```

We have a LimeTabularExplainer because the data is in a dataframe. We use regression and discretize_continuous if True, all non-categorical features will be discretized into quartiles.

In [24]:

```
product_to_enhance.values
```

Out[24]:

```
array([3054])
```

In [25]:

```
exp = explainer.explain_instance(X_test.values[int(product_to_enhance.values)], mode
exp.show_in_notebook(show_all=False)
# only the features used in the explanation are displayed
```

Again we see that as the lda3 value is higher than 0.31 the prediction of the average rating is lowered with 0.07. This might not seem like a lot, but as the standard deviation in the ratings is quite low, it is actually a relevant change. When that being said the most significant features are rank and price.

Sales analysis

Since we are not able to give Amazon recommendations of how to improve their products in terms of product properties extracted from the product descriptions, we shift our focus to how the product descriptions affect sales. The question is if Amazon and their suppliers can do something with their product descriptions which could increase the sales of the product.

Again we will analyse the Grocery and Gourmet Food category "Candy & Chocolate".

For the sales analysis we have created a new branch on our GitHub repository called "sales_rank"

In [26]:

```
# load data
category = 'Candy & Chocolate' # define category
metadata_df = pd.read_csv('data/'+category+'/df_'+category+'.csv')
print('Check category: ', metadata_df.category.unique())
# drop features not used for sales analysis
metadata_df = metadata_df.drop(columns = ['std_rating','category','brand','feature',
```

```
Check category:  ['Candy & Chocolate']
```

Data preprocessing

The preprocessing steps look much like the ones for the first analysis. The features "also_buy", "also_view", "price" and "description" are preprocessed in the same way as described in the first "Data preprocessing" section. For the sales analysis, we will not perform LDA or other topic modelling methods on description. Instead we add the description length and the description sentiment to the features. The sentiment analysis is

performed using a pretrained sentiment model called "vader-lexicon" from the nltk module. Each product description gets a sentiment score between -1 and 1 for the three sentiments: negative, neutral and positive. The compound of these three values is the feature we call "description_sentiment". The larger the compound value is, the more positive the product description is classified as. For this analysis, the length of the product title is also included as a feature.

Amazon's sales rank is a number assigned to products that can range from 1 to over 1 million. The sales rank number tells how well a product sells with 1 being best. That is the lower sales rank number, the more the product is selling. REF: https://sellics.com/blog-amazon-sales-rank/ (https://sellics.com/blog-amazon-sales-rank/) Since the range of sales rank in our training data is from 72 to 6151705, we need to scale the sales rank feature. We choose to use sklearn's MinMaxScaler to scale all the numeric features in the dataset except "description sentiment".

In [27]:

```
## Preprocess data
def preprocess_data(metadata_df):
    df_train, df_test = train_test_split(metadata_df, train_size=0.75, random_state=

    # get number of also_buy
    df_train['also_buy'] = df_train['also_buy'].fillna('').apply(get_number_also_buy
    df_test['also_buy'] = df_test['also_buy'].fillna('').apply(get_number_also_buy)

    # get number of also_view
    df_train['also_view'] = df_train['also_view'].fillna('').apply(get_number_also_b
    df_test['also_view'] = df_test['also_view'].fillna('').apply(get_number_also_buy

    # sales rank information
    df_train['rank'] = df_train['rank'].apply(get_rank).str.replace(',','').str.extr
    df_train['rank'] = pd.to_numeric(df_train['rank'], errors = 'coerce').fillna(0).
    df_test['rank'] = df_test['rank'].apply(get_rank).str.replace(',','').str.extrac
    df_test['rank'] = pd.to_numeric(df_test['rank'], errors = 'coerce').fillna(0).ap
    # remove samples where rank = 0 (not assigned)
    df_train = df_train[df_train['rank']>0]
    df_test = df_test[df_test['rank']>0]
    #print(df_train['rank'].min(),df_train['rank'].max())

    # get title length
    df_train['title_length'] = df_train['title'].apply(get_length)
    df_test['title_length'] = df_test['title'].apply(get_length)

    # get description length
    df_train['description_length'] = df_train['description'].apply(get_length)
    df_test['description_length'] = df_test['description'].apply(get_length)

    # clean description
    df_train['description'] = df_train['description'].apply(get_description)
    df_train = df_train.dropna(axis = 0, subset=['description'])
    df_train['description'] = df_train['description'].apply(str)
    df_train['description'] = df_train['description'].str.replace('\n', '')
    df_train['description'] = df_train[['description']].applymap(lambda text: Beauti
    df_train['description'] = df_train['description'].apply(text_processing)
    df_test['description'] = df_test['description'].apply(get_description)
    df_test = df_test.dropna(axis = 0, subset=['description'])
    df_test['description'] = df_test['description'].apply(str)
    df_test['description'] = df_test['description'].str.replace('\n', '')
    df_test['description'] = df_test[['description']].applymap(lambda text: Beautifu
    df_test['description'] = df_test['description'].apply(text_processing)

    # get sentiment score of description
    df_train['description_sentiment'] = df_train['description'].apply(get_sentiment)
    df_test['description_sentiment'] = df_test['description'].apply(get_sentiment)

    # set price=nan to average price
    temp = df_train[df_train['price'].isna() == False]
    #print('number of products with missing price: ', (df_train.shape[0]-temp.shape[
    mean_value = temp['price'].mean()
    df_train['price'] = df_train.apply(lambda row: mean_value if row['price'] != row
    df_test['price'] = df_test.apply(lambda row: mean_value if row['price'] != row['
    #print('number of rows with price set to average price: ', (df_train.price == me

    # scale
    features_to_scale = ['avg_rating','num_ratings','rank','also_buy','also_view','p
    df_train[features_to_scale] = scaler.fit_transform(df_train[features_to_scale])
```

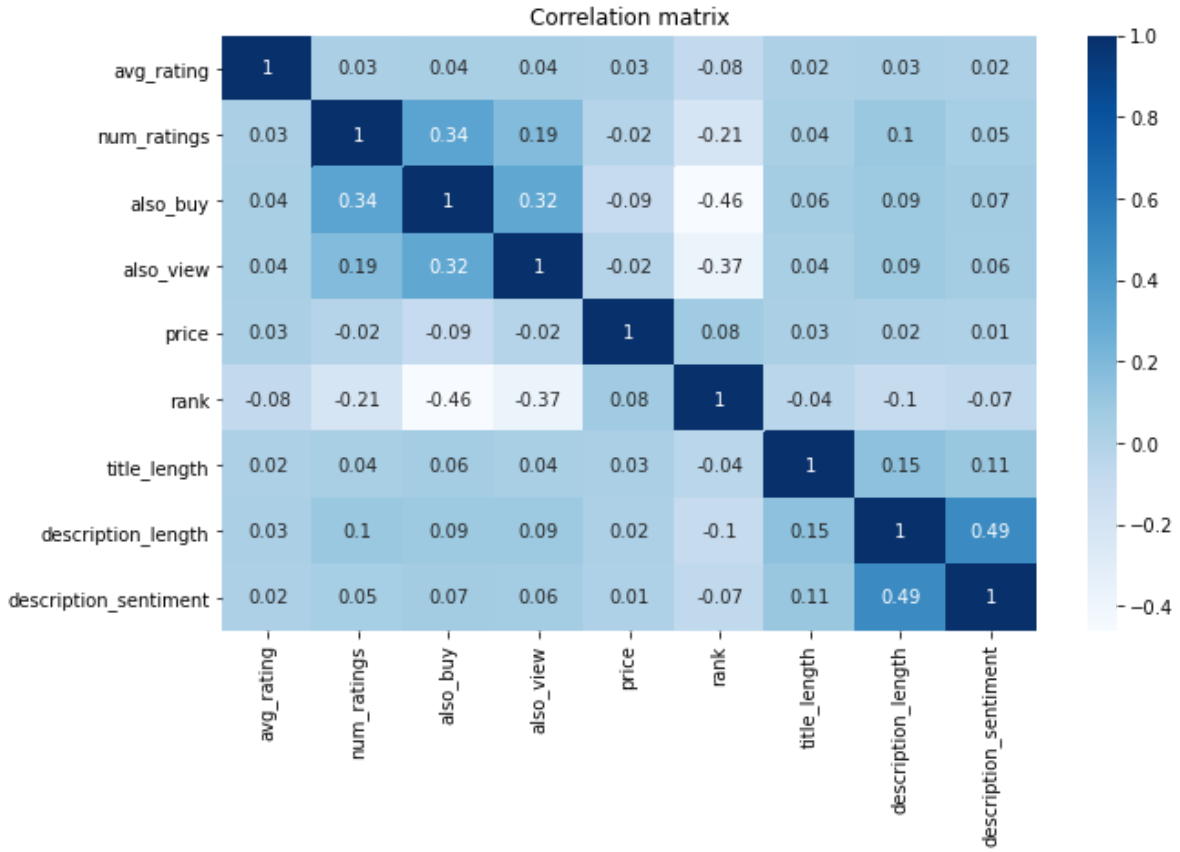
```
    df_test[features_to_scale] = scaler.transform(df_test[features_to_scale])

    return df_train, df_test

# get length of text
def get_length(row):
    if isinstance(row, list):
        if len(row)>0:
            return len(row)
        else:
            return np.nan
    else:
        return len(row)

# sentimental analysis using pretrained sentiment model
def get_sentiment(row):
        compound = sid.polarity_scores(row)['compound']
        return compound

df_train, df_test = preprocess_data(metadata_df)

df_train.to_csv('data/' + category + '/df_train_sales.csv',index=False)
df_test.to_csv('data/' + category + '/df_test_sales.csv',index=False)
```

Exploratory analysis

Before modelling, we start by looking at the correlation between the explanatory features and the response variable "rank". Below the correlation matrix of the training data is plotted.

In [28]:

```
# load data
category = 'Candy & Chocolate' # define category
df_train = pd.read_csv('data/'+category+'/df_train_sales.csv')

# get numeric features
num_features = ['avg_rating','num_ratings','also_buy','also_view','price','rank','ti

# plot correlation matrix
plt.figure(figsize=(10,6))
sns.heatmap(df_train[num_features].corr().round(2), cmap='Blues', annot=True).set_ti
plt.show()
```



Looking at sales rank, it is seen, that this variable is inversely correlated with "also_buy" and "also_view".

This means that products that are selling well are often bought together with other products. This makes a lot of sense, since people tend to buy several items at a time when online shopping and especially when considering chocolate and candy products. Who will buy a single chocolate bar for 2,83 dollars (20 DKK) online?

It also makes sense, that popular products have a longer list of also viewed products. When a product is bought, it is also viewed and thus Amazon is able to get more data on products that the customer have viewed during the same shopping.

Rank is also inversely correlated with the number of ratings. However, since customers are only allowed to rate products after buying, this correlation just tells us that product that sells a lot have more ratings. This is not that interesting and thus we choose not to include the number of ratings in the later modeling of sales rank.

Finally, we see that both description length and description sentiment is a bit inversely correlated with sales rank. Price, title length and average rating do not seem to have much effect on the sales rank.

Besides the correlation plot, we also want to look at the distribution of the description sentiment feature.

In [29]:

```
df_train.description_sentiment.describe()
```

Out[29]:

```
count    29819.000000
mean         0.456610
std          0.425372
min         -0.931300
25%          0.000000
50%          0.526700
75%          0.893400
max          0.999600
Name: description_sentiment, dtype: float64
```

The product descriptions are in general categorised as very positive. It is seen that 50% of the product descriptions are in the interval [0.52,1] which is a very high positive score. This is not surprising since it is the producers themselves who write the descriptions. Below we look at the product description of the product with the lowest sentiment score and the product with the highest sentiment score.

In [30]:

```
product_id_low = (df_train[df_train['description_sentiment'] == df_train['descriptic
product_id_high = (df_train[df_train['description_sentiment'] == df_train['descripti
print('product description classified with sentiment score:', df_train['description_
print('product id: ', product_id_low)
print('product description:', print(metadata_df[metadata_df['item']==product_id_low]
print(' ')
print('product description classified with sentiment score:', df_train['description_
print('product id: ', product_id_high)
print('product description:', print(metadata_df[metadata_df['item']==product_id_high
```

```
product description classified with sentiment score: -0.9313 :
product id:  B00VCEBE4O
['["These flavored lollipops (apple, watermelon, blueberry and orange)
contain a real worm that has been suspended within the flavored confin
es of the sucker. You don\'t have to eat the insect larva, but that wo
uld just defeat the purpose of having a bug in your lollipop, now woul
dn\'t it? Each lollipop is made from a sugar-free, flavored syrup. Pac
k of 4 individually wrapped suckers includes 1 apple, 1 watermelon, 1
blueberry and 1 orange flavor. May contain soy. Each sucker net weight
1 oz.", \'Statements regarding dietary supplements have not been evalu
ated by the FDA and are not intended to diagnose, treat, cure, or prev
ent any disease or health condition.\']']
product description: None

product description classified with sentiment score: 0.9996 :
product id:  B001UG7LRI
['[\'Size: 45 X 45 Mm This Heart Donut Is Absolutely Gorgeous And Powe
rful Love Talisman. It Comes In Genuine And Natural Rose Quartz Gemsto
ne, Which Is Increasing The Powers Of Love Energies And Forces. \\n<br
>The Heart - Donut Also Comes In Precious Pouch, In Case You Wish To G
et This Unique Love Talisman As A Gift For That Special One. \\n<br>Th
is Gift Will Always Be Remembered, Will Always Influence The Love Spir
it Around Your Loved One. This Love Donut Talisman May Also Be Used As
A Pendant. Lucky Powers Of Rose Quartz: Rose Quartz Is A Powerful Love
Stone. Rose Quartz Is Associated With The Heart Chakra As A Crystal Of
Love. It Attracts Love And Helps The Owner Of It To Prepare Emotionall
y For Love. Rose Quartz Is The Essential Stone For Increasing Love. It
Stimulates The BodyS Love Centers And Can Result In Peace And Fidelity
In Committed Relationships. As One Of The Most Important Crystals For
Attracting Love, It Does Emotional Maintenance Clearing Out Emotional
Baggage, Converting Negative Emotions, And Calms Hot Tempers, All Of W
hich Prepare Us For Love. The High Energy Of Quartz Gives Rose Quartz
The Property Of Enhancing Love In Virtually Any Situation. It Is In Al
l A Very Soothing And Happy Stone. Emotionally Rose Quartz Brings Gent
leness, Forgiveness, Compassion, Kindness And Tolerance. Rose Quartz T
akes Its Name From The Flower, Because It Opens The Heart To The Beaut
y Within And All Around Us, And Helps Us Attract Positive, Gentle Love
Into Our Lives. Having A Fantastic Metaphysical Properties, Rose Quart
z Is Like A Wise Old Woman That Knows All Of The Answers And Can Heal
With A Mere Glance. True Love Is The Highest Degree Attained At The He
art Chakra And This Process Is Initiated With Rose Quartz. Rose Quartz
Is The Most Powerful For Dealing With Affairs Of The Heart. It Opens U
p The Heart For Both Giving And Receiving Love. It Soothes Negative In
fluences. It Is "The Stone Of Gentle Love" Bringing Peace And Calm To
Relationships.\', \'Statements regarding dietary supplements have not
been evaluated by the FDA and are not intended to diagnose, treat, cur
e, or prevent any disease or health condition.\']']
product description: None
```

The first product description contains words as: suspended, dont, insect, defeat, bug, dietary, fda, diagnose, prevent and disease, while the second product description includes words as: absolutely, gorgeous, powerful, love, genuine, rose, energy, force, happy, positive, gentle, forgiveness, compassion etc. Hence looking at the words from the two product descriptions, it makes sense that the first product description is associated with a lot more negative energy than the second.

Even though the description of product 'B00VCEBE4O' is categorised as being negative, it may be due to the last statement of the description regarding FDA.

Train model

We are now ready to fit a regression model with sales rank as response variable and average rating, also buy, also view, price, title length, description length and description sentiment as explanatory variables. We use the same training function "train_regression_models" from the script "TrainModel.py" as we used for the first analysis described in the section "Building regression models" to find the best regression model for our data.

In [31]:

```
# load data
category = 'Candy & Chocolate'
df_train = pd.read_csv('data/' + category + '/df_train_sales.csv')
df_test = pd.read_csv('data/' + category + '/df_test_sales.csv')
X_train = df_train.drop(columns=['item','description','title','num_ratings'])
X_test = df_test.drop(columns=['item','description','title','num_ratings'])
X_train = X_train.dropna()
X_test = X_test.dropna()
y_train = X_train['rank']
X_train = X_train.drop(columns='rank')
y_test = X_test['rank']
X_test = X_test.drop(columns='rank')

model, name = train_regression_models(X_train, X_test, y_train, y_test) # use functi
print('best default model: ', name)
#params, tuned_model = tune_model(model, name, X_train, y_train)

# Save tuned model
today = date.today()
filename = 'models/'+category+'/best_performing_model_sales_'+str(today)+'.sav'
pickle.dump(model, open(filename, 'wb'))
```

```
----------------------
Linear Regression:
MAE  0.041383666748311405
R2  0.2641986777760885
Explained variance  0.2643748728283616
----------------------
----------------------
XGBoost Regressor:
MAE  0.02717447253972236
R2  0.6037175095575003
Explained variance  0.6037229861707516
----------------------
----------------------
CatBoost Regressor:
MAE  0.026765660886062467
R2  0.6130841496952512
Explained variance  0.6131148898356515
----------------------
----------------------
Stochastic Gradient Descent Regression:
MAE  0.04178882022493026
R2  0.2623809426262548
Explained variance  0.262582627203811
----------------------
----------------------
Elastic Net Regression:
MAE  0.04995274741299304
R2  -8.580319691153804e-05
Explained variance  2.220446049250313e-16
----------------------
----------------------
Bayesian Ridge Regression:
MAE  0.041384330124082304
R2  0.264258060026636
Explained variance  0.2644339061263551
----------------------
----------------------
```

```
Gradient Boosting Regression:
MAE  0.02742984610695653
R2  0.5981603043045507
Explained variance  0.5982016837084416
----------------------
best default model:  catboost_regressor
```

The best model after running the function "train_regression_models" is a catboost regressor.

Interpretability

Now we want to interpret the model in order to investigate how the features affect the sales rank. We start by computing the shap values of test set and do a shap summary plots.

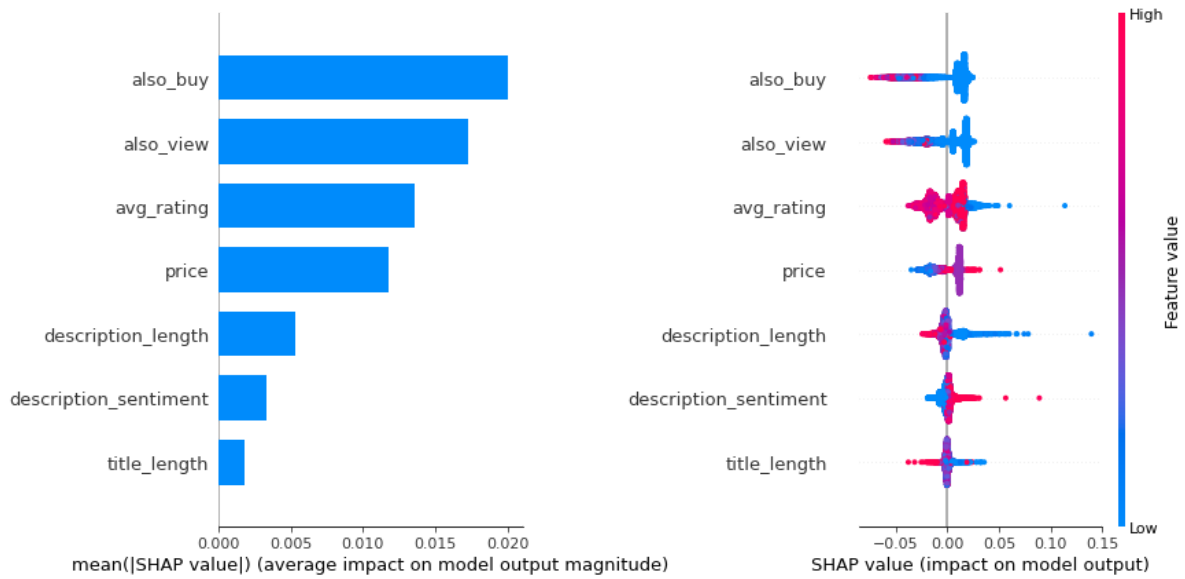
Global interpretability

First we start by investigating the feature importance globally.

In [32]:

```
# load model and data used for modeling
category = 'Candy & Chocolate' # get category
today = date.today()
model = pd.read_pickle('models/'+category+'/best_performing_model_sales_2022-05-11.s

# get features and response
df_train = pd.read_csv('data/' + category + '/df_train_sales.csv')
df_test = pd.read_csv('data/' + category + '/df_test_sales.csv')
X_train = df_train.drop(columns=['item','description','title','num_ratings'])
X_test = df_test.drop(columns=['item','description','title','num_ratings'])
X_train = df_train
X_train = X_train.dropna()
X_test = X_test.dropna()
y_train = X_train['rank']
X_train = X_train.drop(columns='rank')
y_test = X_test['rank']
X_test = X_test.drop(columns='rank')

# MAE
preds = model.predict(X_test)
print('MAE of the best model: ', mean_absolute_error(y_test, preds))

# Get shap values
explainer = shap.TreeExplainer(model)
shap_values = explainer.shap_values(X_test)
shap_values_pandas = pd.DataFrame(shap_values)

# shap summary plots
plt.figure(figsize=(12,6))
plt.subplot(1,2,1)
shap.summary_plot(shap_values, X_test, plot_type='bar', show=False, plot_size=None)
plt.subplot(1,2,2)
shap.summary_plot(shap_values, X_test, show=False, plot_size=None)
plt.tight_layout()
plt.show()
```

```
MAE of the best model:  0.026765660886062467
```



The above figures show summary plots of how the different features affect the prediction of sales rank. The summary plot to the right shows how the features impact the predictions and the left plot shows how the

features impact the predictions in magnitude.

At the left plot, it is seen that "also_buy" and "also_view" are the features affecting the prediction of sales rank the most. Looking at the right plot, it is seen that low values of "also_buy" and "also_view" are increasing the prediction of sales rank, i.e predicting a product that sells worse. This is in correspondance with the correlation matrix, where we saw that sales rank is inversely correlated with both "also_buy" and "also_view".

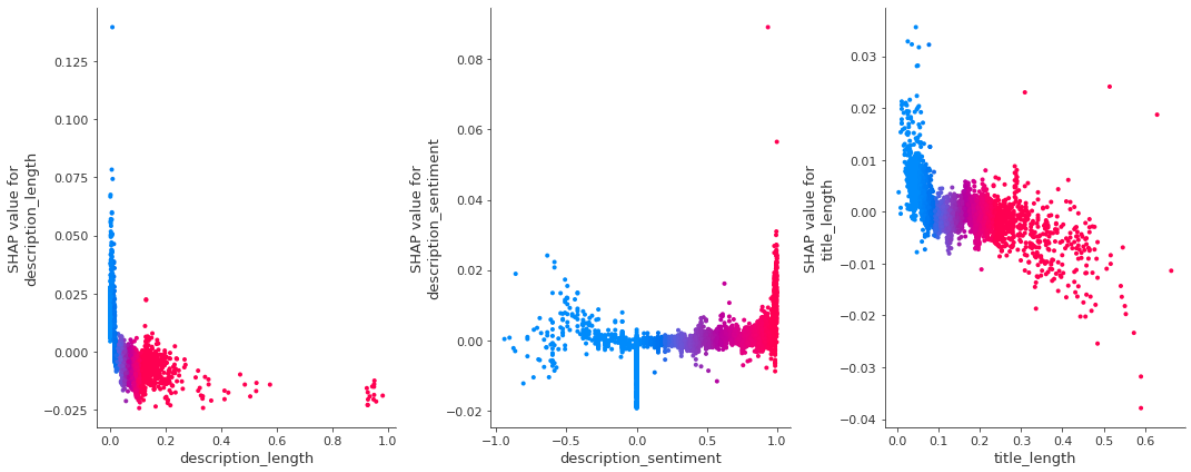
Looking at the average rating, the summary plot to the right shows that a high average rating lowers the prediction of the sales rank which also makes a lot of sense - highly rated products will be bought more.

Regarding the price, the shap values tell that a high price for candy and chocolate make the product less sellable. Our model shows that lowering prices may be a good idea to increase sales, but that is somewhat obvious. However this can be bad for other aspects of their business than the volume of sales.

Regarding description length, description sentiment and title length, the output is very interesting. Even though the importance of these three features are not large, the distribution of low and high values at the right summary plot is very clear. The summary plot to the right shows that a short product description gives a higher prediction of the sales rank, meaning that products that sell less tend to have shorter descriptions. The same tendency holds for the title length, however description length has stronger impact. When looking at the description sentiment, it is seen that high values, meaning a very positive description will give predictions of the prodcut selling less. This might seem weird at first - how can a positive description be a bad thing? Let's investigate this further with a partial dependence plot.

In [33]:

```
# shap dependence plots
fig, axes = plt.subplots(1,3,figsize=(15,6))
shap.dependence_plot(ind='description_length', interaction_index='description_length
shap.dependence_plot(ind='description_sentiment', interaction_index='description_ser
shap.dependence_plot(ind='title_length', interaction_index='title_length',shap_value
plt.tight_layout()
plt.show()
```



The above figure shows a partial dependence plot of the description length, description sentiment and the interaction between these two features. When looking at the partial depence plot of the description length, it is seen that it is none or very short descriptions that pulls the sales rank upwards. When the description length reaches a certain level (around 0.2 in the transformed domain), the effect on sales rank seem to stagnate.

The partial dependence plot of sentiment analysis shows that descriptions can decrease the sales rank a bit if they are catgorised as neutral or negative. Recall that the most negative description from the training set may have been categorised as negative due to the statement about fda. Thus we might not be able to conluce that

it is a good thing when the parts of the description reffering directly to the product has negative sentiment. But we can conclude that very positive over-excited descriptions are not popular and tend to give predictions of larger sales ranks.

Finally, the partial dependence plot of the title length shows that the longer title length the better which was the same affect seen at the shap summary plot.

Local interpretability

We want to find out which features influence the prediction of the worst selling products (high rank). We find the products with the 25% highest sales rank and calculate their prediction errors, since it is best to interpret the model by looking at products where the prediction error is low.

In [34]:

```
idx_high_rank = df_test[df_test['rank'] >= df_test['rank'].describe().loc['75%']].ir
preds_high_rank = model.predict(X_test.iloc[list(idx_high_rank.values)])
idx = np.abs(y_test[list(idx_high_rank.values)]-preds_high_rank).argmin()
```

In [35]:

```
explainer = LimeTabularExplainer(X_test.values,
                                 mode='regression',
                                 feature_names = X_test.columns,
                                 discretize_continuous=True)
exp = explainer.explain_instance(X_test.values[idx], model.predict, num_features=5)
exp.show_in_notebook(show_all=False)
```

For this product, the features "also_buy", "also_view" are increasing the prediction of the sales rank, which makes sense since this product has values zero for these features. This means that the products are rarely bought or viewed together with other products. For this product is just below why the predicion of sales rank is increased. When looking at description length, it is seen that a value above 0.02 will give a better sales rank. This is in correspondance whith the partial dependence plot where we saw that it is no or very short description length that makes the sales rank worse. For this product, the description length is 0.07 which is just above the decision threshold at 0.06. Hence the the sales rank might improve if the description were longer and more elaborate.

Discussion and conclusion

In this project we wanted to find specific features in high rated Candy & Chocolate products and recommend Amazon to use these features on some of their poorly rated products. However through our analysis we were only able to find one topic with some impact on the product rating. This topic contained the words lecithin, butter, syrup, (palm) oil, fat and emulsifier and had a negative impact on the product rating. Due to a vague relationship between the topic and the average rating, it is not possible to recommend Amazon to avoid these ingredients or using them in the product descriptions. However, we can alert Amazon to avoid these ingredients or using them in the product descriptions. In retrospect, it was a difficult task to extract key product features from the product descriptions using topic modelling. Thus we changed the view of our analysis to focus on how to improve the sales rank of the products. From this analysis we can recommend Amazon to write elaborate product descriptions, since an elaborate description showed to have a positive impact on the amount of sales. Furthermore, our findings showed that the product descriptions are best when neutral and not overselling by using extremely positive words.

Generally it has been difficult if not impossible to validate our recommendations. If we were to validate e.g. removing the negatively impacted words from the product description, we would need the average rating of the product after this removal. However, this data has not been accessible and thus the validation can first be made if Amazon chooses to implement our recommendations. The same holds for the sales rank analysis, where the validation of the impact of the description length and sentiment on the sales rank requires these changes to be applied.

Type *Markdown* and LaTeX: α^2