AMAZON REVIEW DATA ANALYSIS

(TOYS AND GAMES DATA)

Submitted by

SIVASANKARI N

TANISH NAMDEV

SAGAR

KARTIK SHARMA

Under the supervision of

Dr. Amit Kumar

CAPSTONE PROJECT

ADVANCE PGP IN DATA SCIENCE AND MACHINE LEARNING

Abstract

The intense competition to attract and maintain customers online is compelling businesses to implement novel strategies to enhance customer experiences. It is becoming necessary for companies to examine customer reviews on online platforms such as Amazon to understand better how customers rate their products and services. The purpose of this study is to investigate how companies can conduct sentiment analysis based on Amazon reviews to gain more insights into customer experiences. The dataset selected for this capstone consists of customer reviews and ratings from consumer reviews of Amazon products. Amazon product reviews enable a business to gain insights into customer experiences regarding specific products and services. The study will enable companies to pinpoint the reasons for positive and negative customer reviews and implement effective strategies to address them accordingly. The capstone project helps companies use sentiment analysis to understand customer experiences using Amazon reviews.

Excess inventory is prevalent in the Toys and games companies; It takes up space and resources that could be used elsewhere. This project also proposes a method to reduce the excess inventory and associated costs by multiple clustering algorithms such as K Means, Agglomerative clustering, and DBSCAN to group similar categories based on their sales and rating.

Companies today use everything from simple spreadsheets to complex financial planning software in a bid to accurately forecast future business outcomes such as product demand, resource needs, and financial performance. This project uses time series forecasting, its terminology, challenges, and use cases.

Table of Contents

1. Introduction

1.1 Background

The year was 1994 when Bezos launched Amazon out of his garage. In 1995, the first product was launched by Amazon. It was a book that was sold to 50 states in 45 different countries within 30 days (Oberlo 2021).

Within 26 years, Amazon holds the title of the world's largest online retailer and has become a household name. Amazon has become synonymous with online shopping and continues to grow by developing new products, acquisitions, and different service offerings to enlarge the customer base.

Nowadays, people (almost 150.6 million) turn on the Amazon app for everything. Several types of research have proved that customers trust Amazon (Statista 2019). On average, the small and medium-sized businesses located in the USA sell more than 4,000 items per minute (Amazon 2019), which leads to millions of product reviews on Amazon.

Reviews tell what products and features are trending, what is in demand, what is no longer relevant, how products and those of competitors are doing, and much more.

It is observed that the maximum number of customers look at product reviews before they make a purchase. Survey results show that positive product reviews are a key factor for purchasing by 57 percent of Amazon buyers (Statista, 2019).

As product reviews are often the deciding factor for many customers, it's very important to have a well-automated system for monitoring them.

The traditional manual process of Amazon product reviews is time-consuming and inefficient when millions of reviews are being posted all the time. It doesn't show any trend or patterns over time. Moreover, it is tough to understand customers' sentiment towards any product or its delivery.

Review analysis must dynamically adjust to the changing trend.

1.2 Objective

- This project covers our skills in using various aspects of data analysis tools effectively.
- Our solution should include a good analysis of the data and make use of the best approach for forecasting.
- A good solution has a sound application of programming and algorithmic knowledge that matches the given problem statement.
- Put together your modular coding and analytical skills in use for this project.

1.3 Case Study: Amazon Product Review Analysis

Thomas, a global market analyst, wishes to develop an automated system to analyze and monitor an enormous number of reviews. By monitoring the entire review history of products, he wishes to analyze tone, language, keywords, and trends over time to provide valuable insights that increase the success rate of existing and new products and marketing campaigns.

1.4 Scenario 1: Inventory Optimization and Demand Forecasting

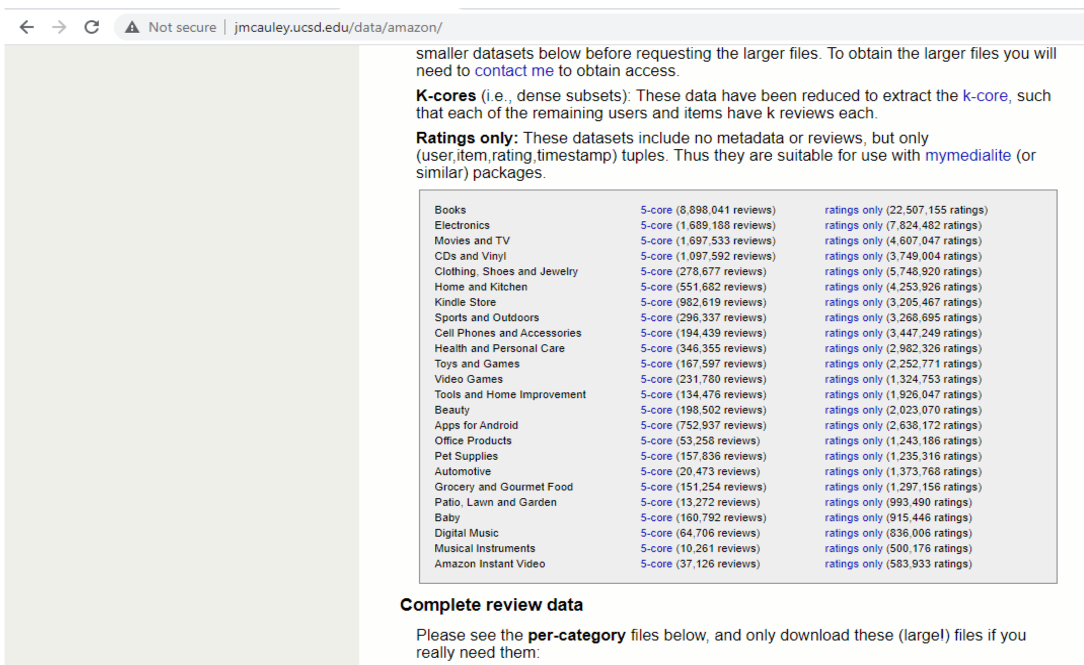
Optimize inventory management by identifying the product categories (Clustering as an outcome of text processing) on the customer review data. Predict what kind of products could be in demand (Time Series Analysis).

2. Data Exploration

Data exploration is the first step of data analysis used to explore and visualize data to uncover insights from the start or identify areas or patterns to dig into more. Using interactive dashboards and point-and-click data exploration, users can better understand the bigger picture and get to insights faster.

2.1 Data Source

We are downloading the datasets from http://jmcauley.ucsd.edu/data/amazon/ and I am using video_games and toys_and_games pair for this project.



Not secure | jmcauley.ucsd.edu/data/amazon/

smaller datasets below before requesting the larger files. To obtain the larger files you will need to contact me to obtain access.

K-cores (i.e., dense subsets): These data have been reduced to extract the k-core, such that each of the remaining users and items have k reviews each.

Ratings only: These datasets include no metadata or reviews, but only (user,item,rating,timestamp) tuples. Thus they are suitable for use with mymedialite (or similar) packages.

Books	5-core (8,898,041 reviews)	ratings only (22,507,155 ratings)
Electronics	5-core (1,689,188 reviews)	ratings only (7,824,482 ratings)
Movies and TV	5-core (1,697,533 reviews)	ratings only (4,607,047 ratings)
CDs and Vinyl	5-core (1,097,592 reviews)	ratings only (3,749,004 ratings)
Clothing, Shoes and Jewelry	5-core (278,677 reviews)	ratings only (5,748,920 ratings)
Home and Kitchen	5-core (551,682 reviews)	ratings only (4,253,926 ratings)
Kindle Store	5-core (982,619 reviews)	ratings only (3,205,467 ratings)
Sports and Outdoors	5-core (296,337 reviews)	ratings only (3,268,695 ratings)
Cell Phones and Accessories	5-core (194,439 reviews)	ratings only (3,447,249 ratings)
Health and Personal Care	5-core (346,355 reviews)	ratings only (2,982,326 ratings)
Toys and Games	5-core (167,597 reviews)	ratings only (2,252,771 ratings)
Video Games	5-core (231,780 reviews)	ratings only (1,324,753 ratings)
Tools and Home Improvement	5-core (134,476 reviews)	ratings only (1,926,047 ratings)
Beauty	5-core (198,502 reviews)	ratings only (2,023,070 ratings)
Apps for Android	5-core (752,937 reviews)	ratings only (2,638,172 ratings)
Office Products	5-core (53,258 reviews)	ratings only (1,243,186 ratings)
Pet Supplies	5-core (157,836 reviews)	ratings only (1,235,316 ratings)
Automotive	5-core (20,473 reviews)	ratings only (1,373,768 ratings)
Grocery and Gourmet Food	5-core (151,254 reviews)	ratings only (1,297,156 ratings)
Patio, Lawn and Garden	5-core (13,272 reviews)	ratings only (993,490 ratings)
Baby	5-core (160,792 reviews)	ratings only (915,446 ratings)
Digital Music	5-core (64,706 reviews)	ratings only (836,006 ratings)
Musical Instruments	5-core (10,261 reviews)	ratings only (500,176 ratings)
Amazon Instant Video	5-core (37,126 reviews)	ratings only (583,933 ratings)

Complete review data

Please see the **per-category** files below, and only download these (large!) files if you really need them:

2.2 Data Acquisition

Importing the CSV files:

We are importing both the files from the folder into the python codebook.

Review dataset

```
[ ]  tg = getDF('F:/NIIT/Capstone Project/Resources/Toys_and_Games_5.json.gz')
```

Metadata

```
[ ]  mtg = getDF('F:/NIIT/Capstone Project/Resources/meta_Toys_and_Games.json.gz')
```

2.4 Data Preparation

Data preparation (also referred to as "data preprocessing") is the process of transforming raw data so that data scientists and analysts can run it through machine learning algorithms to uncover insights or make predictions.

Review Dataset:

Review dataset

```
[ ] print('Toys_and_Games')
    display(tg.head())

    print('Shape:',tg.shape)

    print('COLUMN'.ljust(18),'DATATYPE'.ljust(10),'NULL VALUES')
    print('--'*20)
    for i in tg.columns:
        nanp = tg[i].isnull().sum()/len(tg[i])*100
        print(f'{i:18} {str(tg[i].dtype):10} {nanp:.2f}%')
```

Toys_and_Games

	overall	vote	verified	reviewTime	reviewerID	asin	style	reviewerName	reviewText	summary	unixReviewTime	image
0	5.0	3	True	10 6, 2013	A2LSCFZM2FBZK7	0486427706	{'Format:': ' Paperback'}	Ginger	The stained glass pages are pretty cool. And i...	Nice book	1381017600	NaN
1	5.0	9	True	08 9, 2013	A3IXP5VS847GE5	0486427706	{'Format:': ' Paperback'}	Dragonflies & Autumn Leaves	My 11 y.o. loved this...and so do I (you know ...	Great pictures	1376006400	NaN
2	5.0	NaN	True	04 5, 2016	A1274GG1EB2JLJ	0486427706	{'Format:': '	barbara ann	The pictures are great , I've done one	The pictures are great, I've done	1459814400	NaN

Missing Value Treatment:

Fields in Toys and Games that had more than 70% missing values: vote, style, and image

Since those fields don't have much information, those fields are dropped

```
Shape: (1828971, 12)
COLUMN            DATATYPE   NULL VALUES
----------------------------------------
overall           float64    0.00%
vote              object     88.63%
verified          bool       0.00%
reviewTime        object     0.00%
reviewerID        object     0.00%
asin              object     0.00%
style             object     71.38%
reviewerName      object     0.01%
reviewText        object     0.06%
summary           object     0.02%
unixReviewTime    int64      0.00%
image             object     97.74%
```

Dropping Columns with more than 70% of null value

```
[ ] # for reusability
    tgc = tg.copy()

    # dropping fields that don't have information
    tgc = tgc.drop(['vote','style','image'],axis=1)

    # fields that aren't relavant
    tgc = tgc.drop(['reviewerID','reviewerName','summary','unixReviewTime'],axis=1)

    # changing to relevant data types
    tgc['reviewTime'] = pd.to_datetime(tgc['reviewTime'])
```

```
[ ] print('Toys_and_Games')
    print('Shape:',tgc.shape)


    print('COLUMN'.ljust(18),'DATATYPE'.ljust(10),'NULL VALUES')
    print('--'*20)
    for i in tgc.columns:
        nanp = tgc[i].isnull().sum()/len(tgc[i])*100
        print(f'{i:18} {str(tgc[i].dtype):15} {nanp:.2f}%')
```

```
Toys_and_Games
Shape: (1828971, 5)
COLUMN            DATATYPE    NULL VALUES
----------------------------------------
overall           float64         0.00%
verified          bool            0.00%
reviewTime        datetime64[ns]  0.00%
asin              object          0.00%
reviewText        object          0.06%
```

Metadata

Metadata

```
[ ] # for reusability
mtgc = mtg.copy()

display(mtgc.head())
```

	category	tech1	description	fit	title	also_buy	tech2	brand	feature	rank	also_view	main_cat	similar_item	date
0	[Toys & Games, Puzzles, Jigsaw Puzzles]		[Three Dr. Suess' Puzzles: Green Eggs and Ham,...		Dr. Suess 19163 Dr. Seuss Puzzle 3 Pack Bundle	[]		Dr. Seuss	[Three giant floor puzzles, Includes: Dr. Sues...	[>#2,230,717 in Toys & Games (See Top 100 in T...	[]	Toys & Games		
1	[]		[Prepare to be Afraid! The B...		Pathfinder: Book of Beasts - Legendary Foes	[]		Pathfinder Roleplaying Jon Brazer Productions	[]	[>#2,294,535 in Toys & Games (See Top 100 in T...	[]	Toys & Games		
2	[]		[]		Nursery Rhymes Felt Book	[]		Betty Lukens	[]	[>#2,871,983 in Toys & Games (See Top 100 in T...	[]	Toys & Games		

```
print('Meta Toys_and_Games')
print('Shape:',mtgc.shape)

print('COLUMN'.ljust(18),'DATATYPE'.ljust(10),'NULL VALUES')
print('--'*20)
for i in mtgc.columns:
    nanp = mtgc[i].isnull().sum()/len(mtgc[i])*100
    print(f'{i:18} {str(mtgc[i].dtype):10} {nanp:.2f}%')
```

```
Meta Toys_and_Games
Shape: (633883, 19)
COLUMN             DATATYPE   NULL VALUES
----------------------------------------
category           object     0.00%
tech1              object     0.00%
description        object     0.00%
fit                object     0.00%
title              object     0.00%
also_buy           object     0.00%
tech2              object     0.00%
brand              object     0.00%
feature            object     0.00%
rank               object     0.00%
also_view          object     0.00%
main_cat           object     0.00%
similar_item       object     0.00%
date               object     0.00%
price              object     0.00%
asin               object     0.00%
imageURL           object     0.00%
imageURLHighRes    object     0.00%
details            object     0.23%
```

Data Cleaning

Data cleaning is the process of fixing or removing incorrect, corrupted, incorrectly formatted, duplicate, or incomplete data within a dataset.

To get the price data in as float data type and dropping irrelevant and blank columns

```
def text_to_currency(df,col):
    '''Returns a series of float value
    Parameters: DataFrame df, Field col'''

    import re
    import numpy as np

    for i in df[col].index:
        item = df[col][i]
        res = re.compile('(\$\d+\.\d+)').findall(item)
        if res:
            df[col][i] = float(re.sub('\$','',res[0]))
        else:
            df[col][i] = np.nan

    return df[col]
```

```
[ ] # changing data types
mtgc['price'] = text_to_currency(mtgc,'price')

mtgc['price'] = mtgc['price'].astype(float)
```

```
[ ] # dropping blank + irrelavant fields
mtgc.drop(['tech1','description','fit','also_buy','tech2','feature','rank','also_view','similar_item','date',
           'imageURL','imageURLHighRes','details']
          ,axis=1,inplace=True)
```

```
print('Meta Toys_and_Games')
print('Shape:',mtgc.shape)

print('COLUMN'.ljust(18),'DATATYPE'.ljust(10),'NULL VALUES')
print('--'*20)
for i in mtgc.columns:
    nanp = mtgc[i].isnull().sum()/len(mtgc[i])*100
    print(f'{i:18} {str(mtgc[i].dtype):10} {nanp:.2f}%')
```

```
Meta Toys_and_Games
Shape: (633883, 6)
COLUMN             DATATYPE   NULL VALUES
----------------------------------------
category           object     0.00%
title              object     0.00%
brand              object     0.00%
main_cat           object     0.00%
price              float64    50.29%
asin               object     0.00%
```

Merging Reviews and Metadata

We merged core data and metadata using inner join with asin as the common field because we need both main _category and reviews in one dataset for our analysis.

Merging reviews and product

```
data = tgc.merge(mtgc,how='inner',on='asin')
```

```
print('Toys_and_Games')
print('Shape:',data.shape)


print('COLUMN'.ljust(18),'DATATYPE'.ljust(15),'NULL VALUES')
print('--'*20)
for i in data.columns:
    nanp = data[i].isnull().sum()/len(data[i])*100
    print(f'{i:18} {str(data[i].dtype):15} {nanp:.2f}%')
```

```
Toys_and_Games
Shape: (1884953, 10)
COLUMN             DATATYPE        NULL VALUES
----------------------------------------
overall            float64         0.00%
verified           bool            0.00%
reviewTime         datetime64[ns]  0.00%
asin               object          0.00%
reviewText         object          0.06%
category           object          0.00%
title              object          0.00%
brand              object          0.00%
main_cat           object          0.00%
price              float64         26.12%
```

Data Cleaning

Deleting the records with missing value

```
# deleting the records containing missing values
# since the products not having price don't have and details for appropriate mean/median imputation
# and imputation of category/brand wise means data manipulation and might/not condratict the actual data
data = data.dropna(axis=0)


# picking out only the necessary category for the
data = data[data.main_cat=='Toys & Games'].drop('main_cat',axis=1)


# resetting the index afer dropping
data = data.reset_index()
data = data.drop(['index'],axis = 1)
```

```
print('Toys_and_Games')
print('Shape:',data.shape)


print('COLUMN'.ljust(18),'DATATYPE'.ljust(15),'NULL VALUES')
print('--'*20)
for i in data.columns:
    nanp = data[i].isnull().sum()/len(data[i])*100
    print(f'{i:18} {str(data[i].dtype):15} {nanp:.2f}%')
```

```
Toys_and_Games
Shape: (1280271, 9)
COLUMN             DATATYPE        NULL VALUES
----------------------------------------
overall            float64         0.00%
verified           bool            0.00%
reviewTime         datetime64[ns]  0.00%
asin               object          0.00%
reviewText         object          0.00%
category           object          0.00%
title              object          0.00%
brand              object          0.00%
price              float64         0.00%
```

```
data.head()
```

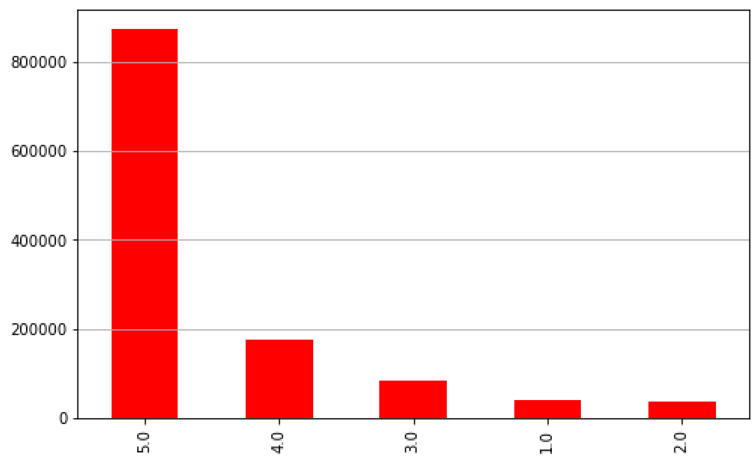
	overall	verified	reviewTime	asin	reviewText	category	title	brand	price
0	3.0	True	2014-02-23	0545346193	The packaging for a draw-your-own DC Universe ...	[Toys & Games, Arts & Crafts, Craft Kits]	Klutz Draw DC Universe: Learn to Draw The Hero...	Klutz	12.92
1	4.0	True	2012-11-14	0545346193	I'm 37 years old and I just took up drawing ag...	[Toys & Games, Arts & Crafts, Craft Kits]	Klutz Draw DC Universe: Learn to Draw The Hero...	Klutz	12.92
2	5.0	True	2015-02-11	0545346193	Our little artist loves this kit.	[Toys & Games, Arts & Crafts, Craft Kits]	Klutz Draw DC Universe: Learn to Draw The Hero...	Klutz	12.92
3	5.0	True	2015-01-04	0545346193	love this delivered on time	[Toys & Games, Arts & Crafts, Craft Kits]	Klutz Draw DC Universe: Learn to Draw The Hero...	Klutz	12.92
4	4.0	True	2016-02-14	0545346193	Useful if you want to train your muscle memory...	[Toys & Games, Arts & Crafts, Craft Kits]	Klutz Draw DC Universe: Learn to Draw The Hero...	Klutz	12.92

2.3 Exploratory Data Analysis

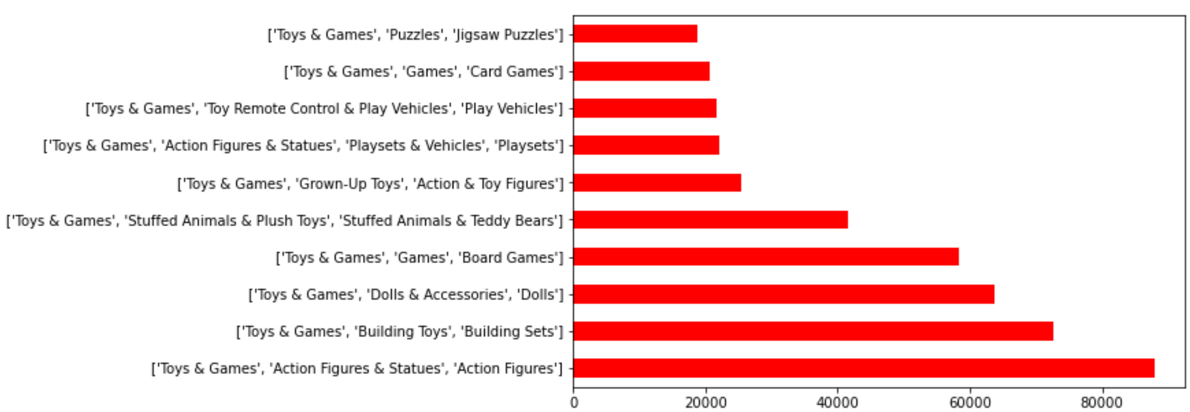
Exploratory Data Analysis (EDA) is an approach to analyze the data using visual techniques. It is used to discover trends, patterns, or to check assumptions with the help of statistical summary and graphical representations.

2.3.1 Univariate analysis

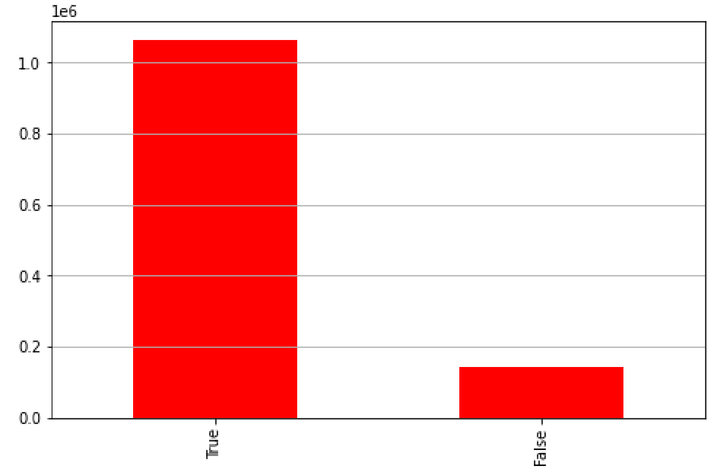
Univariate analysis is the simplest form of analyzing data. Uni means one, so in other words the data has only one variable. Univariate data requires analyzing each variable separately.



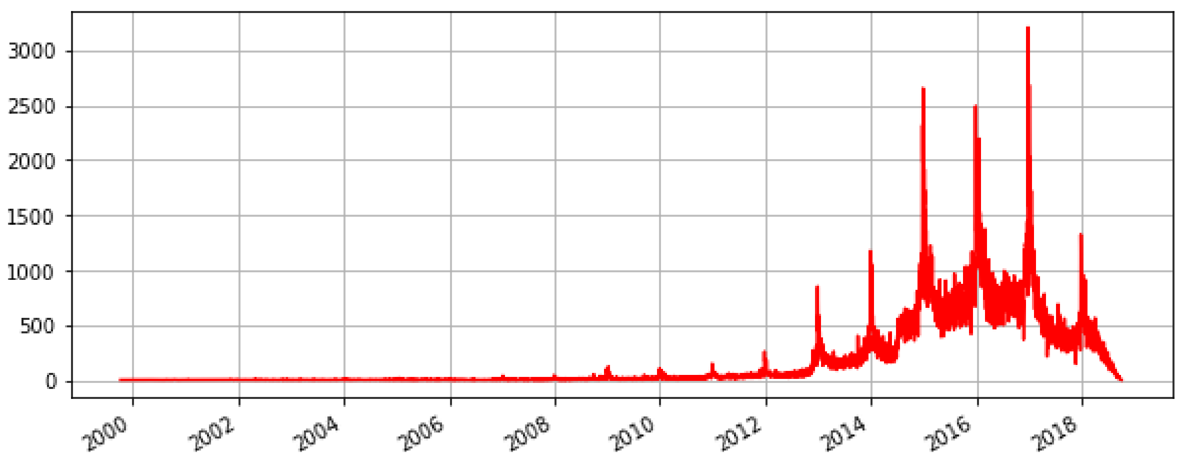
Inference
Higher number of 5/5 reviews implies most customers are pleased with the products and services



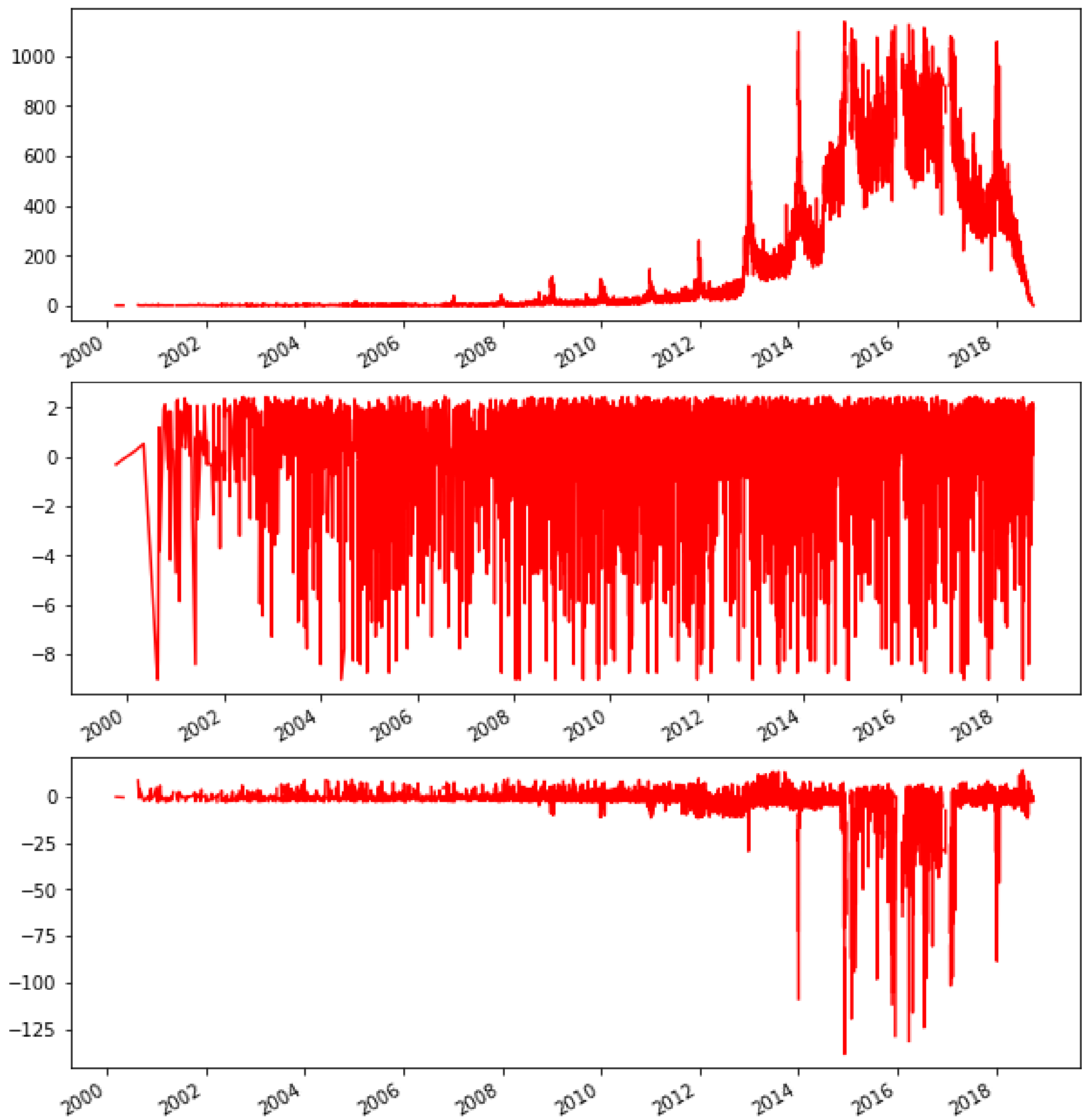
Inference
Action figures and action statues are the top on review stand, its well spoken about- very popular among customers



Inference
Most of the reviewed products are verified

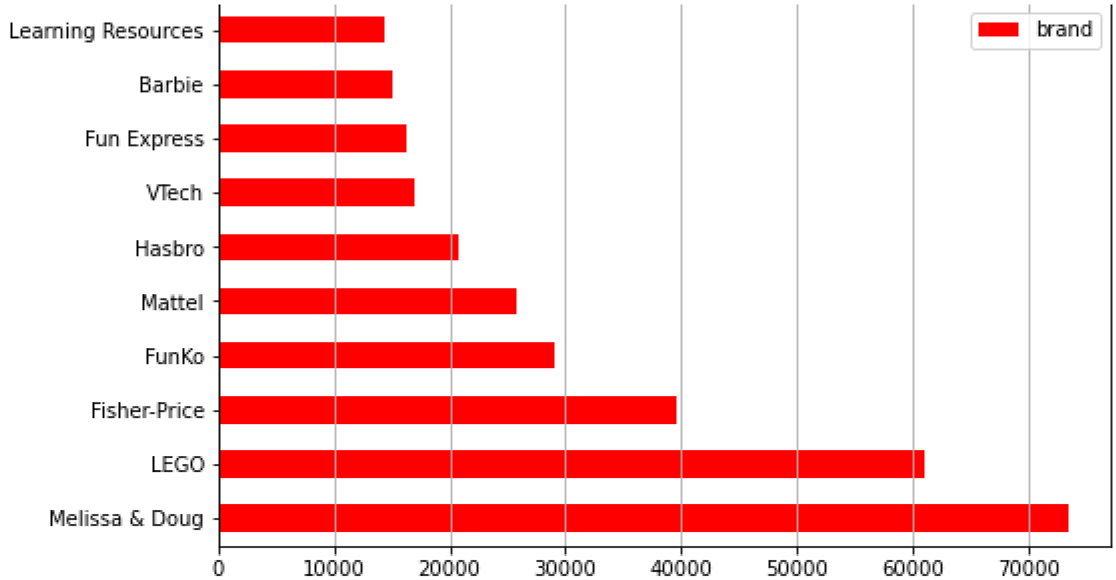


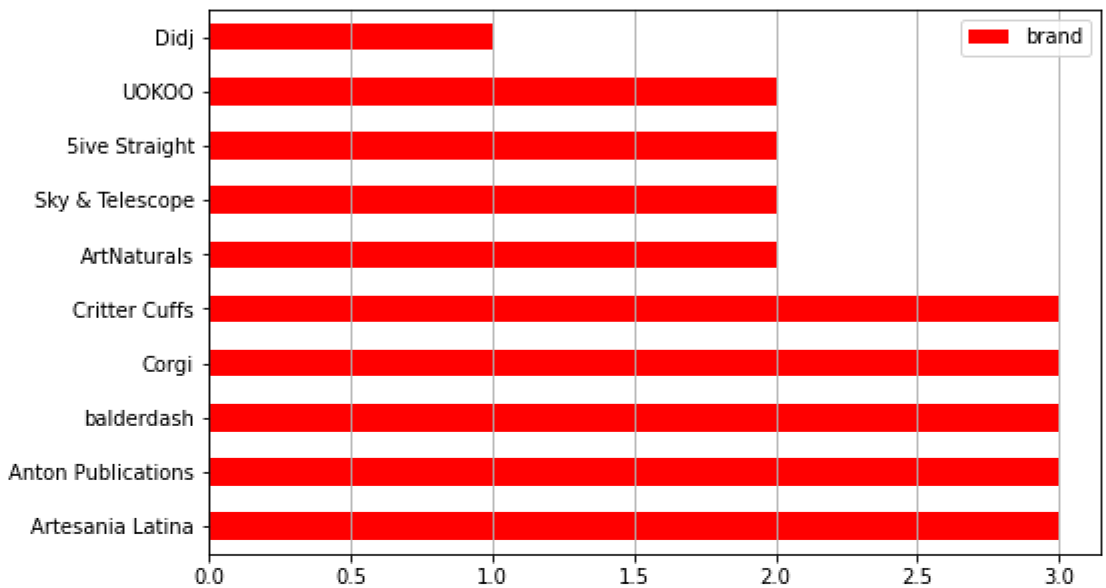
Inference
The performance of product popularity ie. The review collection has remained in the same range till 2012. The number of reviews or say the popularity of products hit its epitome between 2015-17



Inference

Trend and residuals are related- The residuals are high when the popularity is high



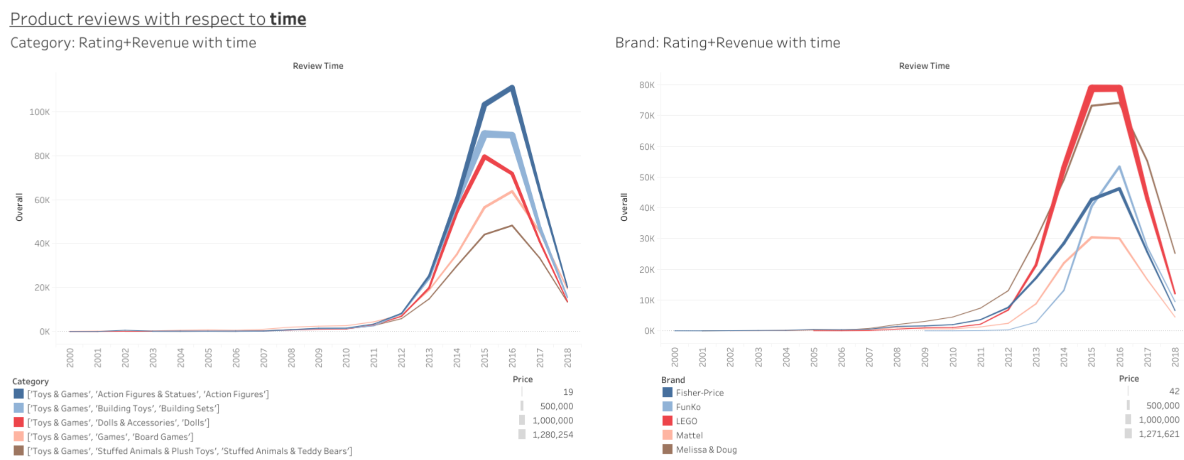


Inference

Melissa & Dough is the most popular brand and Didj is the least popular brand

2.3.2 Multivariate analysis

Multivariate analysis means the analysis of multivariate data. It is one of the simplest forms of statistical analysis, used to find out if there is a relationship between more than two sets of values.

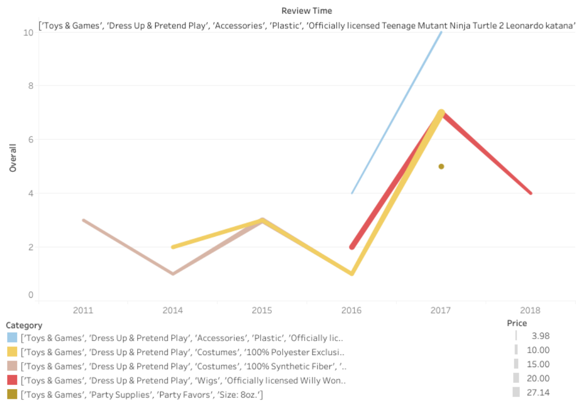


Inferences

- Most customers are pleased with the products and services
- Action figures and action statues are the most popular
- Most of the reviewed products are verified
- The popularity of products hit its epitome between 2015-17
- Trend and residuals are related- The residuals are high when the popularity is high
- Melissa & Dough is the most popular brand and Didj is the least popular brand

Product reviews with respect to **time**

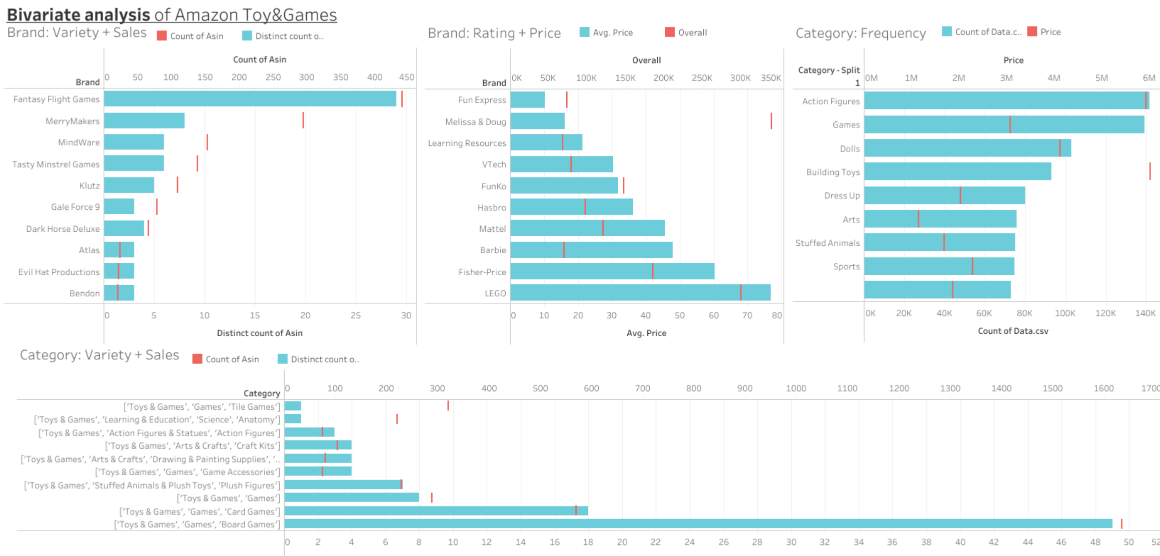
Category: Rating+Revenue with time



Brand: Rating+Revenue with time

Inferences

- Toys and Games hit its epitome between 2015-17
- Relationships can be observed between price and ratings: directly proportional.
- [Action figures] receive the highest ratings. In spite of being the most expensive of all toys [Building toys, Building sets] maintains second in total ratings throughout the time.
- Fisher-Price is the oldest brand member and has been maintaining its standing though many other younger brands have surpassed it.
- LEGO is the expensive younger brand receiving high ratings.
- Poorly rated categories and brands have inconsistent performance. They are only short lived 1-3 years.
- All the poor reviewed categories are expensive and related to [costumes].
- Brands have seen decline in performance before termination. Artesania Latina is a just born brand showing increase in ratings but slow



Inferences

- FantasyFlightGames has the highest variety and products. But MerryMakers has the highest number of products despite a lesser variety of products.
- Melissa&Dough is the most popular producer of non-expensive products
- Most products of toys are Action Figures, the average price of action figure toy is above average than the price of other toys
- Most variety and most products are in the category of board games. Category having the highest products: variety ratio is tile games

- Despite having some variety 1967 MILTON BRADLEY-MB doesn’t produce many products.
- ArtNaturals is the most non-expensive brand among least popular products and has the highest rating/price ratio.
- Art toy products are the least expensive and the least reviewed category.
- Despite having variety, Party Supplies are the least reviewed category.

3. Sentiment Analysis

3.1 Sentiment labeling

Defining a function to separate overall ratings into positive, negative and neutral:

Ratings greater than 3 are labeled as positive, lesser than 3 are labeled as negative and equal to 3 as neutral.

Ratings to sentiments

```
def predict_sentiment(rating):

    '''Returns the sentiment of the rating'''
    if rating < 3:
        sentiment = 'Negative'
    elif rating > 3:
        sentiment = 'Positive'
    else:
        sentiment = 'Neutral'

    return sentiment
```

```
data['sentiment'] = data['overall'].apply(predict_sentiment)
data.head()
```

	overall	verified	reviewTime	asin	reviewText	summary	category	brand	price	sentiment
0	3.0	True	2014-02-23	0545346193	The packaging for a draw-your-own DC Universe ...	Great Idea But Needs More Instructions & Detai...	['Toys & Games', 'Arts & Crafts', 'Craft Kits']	Klutz	12.92	Neutral
1	4.0	True	2012-11-14	0545346193	I'm 37 years old and I just took up drawing ag...	Good for beginners	['Toys & Games', 'Arts & Crafts', 'Craft Kits']	Klutz	12.92	Positive
2	5.0	True	2015-02-11	0545346193	Our little artist loves this kit. Five Stars	Five Stars	['Toys & Games', 'Arts & Crafts', 'Craft Kits']	Klutz	12.92	Positive
3	5.0	True	2015-01-04	0545346193	love this delivered on time Five Stars	Five Stars	['Toys & Games', 'Arts & Crafts', 'Craft Kits']	Klutz	12.92	Positive
4	4.0	True	2016-02-14	0545346193	Useful if you want to train your muscle memory...	Great Drawing Practice	['Toys & Games', 'Arts & Crafts', 'Craft Kits']	Klutz	12.92	Positive

3.2 Data Cleaning

i) Expanding short forms

Words like I'm, I've, etc will be converted to their expanded forms
Retaining in the short forms might leave to meaningless words when the punctuations are removed

expand shortforms

```
!pip install contractions
```

```
import contractions
```

```
data['reviewText'] = data['reviewText'].apply(lambda x:contractions.fix(x))
```

ii) Removing special characters

removing special characters

```
import re
```

```
def clean_text(text):

    '''Return clean version of the text'''

    text = text.lower()
    # Remove all non-letters and non-spaces except for hyphens and digits
    text = re.sub("[^0-9A-Za-z ]+", " ", text)
    # Remove all numbers except those attached to a word
    text = re.sub("(?<!\w)\d+", "", text)
    # Remove multiple spaces
    text = re.sub('\s+',' ',text)

    return text
```

```
data['reviewText'] = data['reviewText'].apply(clean_text)
data['reviewText'].head()
```

```
0    the packaging for a draw your own dc universe ...
1    i am years old and i just took up drawing agai...
2          our little artist loves this kit five stars
3               love this delivered on time five stars
4    useful if you want to train your muscle memory...
Name: reviewText, dtype: object
```

In the datasets in the reviewText column there are characters like '[Toys]',URL, '<span>',Punctuations,'\n','w8d' which are not required, so we are defining a function to remove them.

3.3 Lemmatization

Words describing the same action are made the same for easier understanding and quicker processing of the text.

lemmatization

```
import nltk
from nltk.stem import WordNetLemmatizer
```

```
def lemmatization(text):

    '''Returns the text after lemmatization'''

    lemmatizer = WordNetLemmatizer()
    words = text.split()
    text_lemma = ' '.join(lemmatizer.lemmatize(word) for word in words)

    return text_lemma
```

```
data['reviewText'] = data['reviewText'].apply(lemmatization)
data['reviewText'].head()
```

```
0    the packaging for a draw your own dc universe ...
1    i am year old and i just took up drawing again...
2           our little artist love this kit five star
3                love this delivered on time five star
4    useful if you want to train your muscle memory...
Name: reviewText, dtype: object
```

3.4 Stopwords removal

The stopwords are the most common words in data. They are words that aren't important to describe the subject of the content

stopwords removal

```
from spacy.lang.en.stop_words import STOP_WORDS
```

negation words are removed for appropriate vectorization and therby correct classification

```
STOP_WORDS -= {'cannot','least','less','neither','never','no','not','nor','none','nobody'}
```

```
def remove_stopwords(text):

    '''Returns text after removing stopwords'''

    new_text = []
    for word in text.split():
        if word not in STOP_WORDS:
            new_text.append(word)

    return ' '.join(new_text)
```

```
data['reviewText'] = data['reviewText'].apply(remove_stopwords)
data['reviewText'].head()
```

```
0    packaging draw dc universe look easy think twi...
1    year old took drawing time high school wa not ...
2                          little artist love kit star
3                             love delivered time star
4    useful want train muscle memory tracing improv...
Name: reviewText, dtype: object
```

3.5 Classification Modeling

The Classification algorithm is a Supervised Learning technique that is used to identify the category of new observations on the basis of training data. In Classification, a program learns from the given dataset or observations and then classifies new observations into a number of classes or groups.

Challenges

1. Large data size
2. Unbalance in data

Solution hypothesis

1. Considering data after significant transformation i.e 2012
   - Rejected - no significant reduction
2. Choose the time period that has the maximum reviews i.e 2015-17
   - Rejected - no significant reduction
3. Stratified random sampling of data
   - Accepted

```
# stratified random sampling 10% of each sentiment

grouped = data.groupby('sentiment')
group_names = data['sentiment'].unique()

sample = pd.DataFrame()

for g in group_names:
    igroup = grouped.get_group(g)
    temp_sample = igroup.sample(frac=0.05,replace=False)
    sample = sample.append(temp_sample,ignore_index=True)
```

```
sample.shape
```

```
(60286, 10)
```

```
sample['sentiment'].value_counts()
```

```
Positive    52456
Neutral      4106
Negative     3724
Name: sentiment, dtype: int64
```

To verify if the sample represents the population: Hypothesis testing

Test scenario:

Comparison of ordinal data between two groups
Comparison of rating distribution between the sample and the population

Test employed:

Mood's median test

Mood's median test

Is a non-parametric test- a special case of Pearson's Chi-squares test that compares the median of two or more groups for difference. It calculates a range of values that is likely to include the difference between population medians.

Null Hypothesis: The population Medians are all equal.
Alternate Hypothesis: The Medians are not all equal

```
p = pd.DataFrame(data['sentiment'].value_counts()).rename({'sentiment':'frequency'},axis=1)
p = p.sort_values(by='frequency',ascending=True)
p
```

	frequency
Positive	1049113
Neutral	82129
Negative	74485

```
s = pd.DataFrame(sample['sentiment'].value_counts()).rename({'sentiment':'frequency'},axis=1)
s = s.sort_values(by='frequency',ascending=True)
s
```

	frequency
Positive	52456
Neutral	4106
Negative	3724

```python
# significance level = 5%
alpha = 0.05

# df = number of sample groups - 1
df = 1
```

```python
from scipy.stats import median_test

res = median_test(p['frequency'].values,s['frequency'].values)

pvalue = res[1]

if pvalue < alpha:
    print('Reject null hypothesis')
    print('Medians of the groups are not equal')
else:
    print('Cannot reject null hypothesis')
    print('Medians of the groups are equal')
```

```
Cannot reject null hypothesis
Medians of the groups are equal
```

Implies that the sample taken follows the same population distribution.

Vectorization

Vectorization is a methodology in NLP to map words or phrases from vocabulary to a corresponding vector of real numbers based on word similarities/semantics. The process of converting words into numbers are called Vectorization
Vectorization techniques: bag of words, count vectorizer, tf-idf, word2vec, etc.

Vectorization technique used here is tf-idf: term frequency inverse document frequency
Tfidf calculates the frequency of the words in the total document and in each document and determines the significance of the word.

```python
from sklearn.feature_extraction.text import TfidfVectorizer
```

```python
x = sample['reviewText']
y = sample['sentiment']

tfidf = TfidfVectorizer()
X = tfidf.fit_transform(x)
```

```python
from sklearn.preprocessing import LabelEncoder
```

```python
encoder = LabelEncoder()
Y = encoder.fit_transform(y)
```

Data splitting

Data is splitted into training and testing datasets. The models will be trained on the training data and will be tested on the testing data. Based on the prediction of the testing data the goodness of the model will be determined.

```
from sklearn.model_selection import train_test_split
```

```
x_train,x_test,y_train,y_test = train_test_split(X,Y,test_size=0.2,random_state=777)

print('x:',x_train.shape,x_test.shape)
print('y:',y_train.shape,y_test.shape)
```

```
x: (48228, 29507) (12058, 29507)
y: (48228,) (12058,)
```

```
x_train = x_train.toarray()
x_test = x_test.toarray()
```

Modeling

for multi-classification: KNeighborsClassifier, Naive-bayes, DecisionTreeClassifier, RandomForestClassifier(Bagging), AdaBoost, Stacking

```
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score, f1_score, precision_score, recall_score
```

Evaluation of classification model are based on metrics such as:
Accuracy, precision, recall, f1_score
Accuracy is the model's ability to make correct predictions
Precision is the model's ability to classify positive values correctly
Recall is the model's ability to predict positive values
F1_score is the weighted average of precision and recall

KNeighbors

```
from sklearn.neighbors import KNeighborsClassifier
```

The k-nearest neighbors algorithm, also known as KNN or k-NN, is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point.

finding the parameters for maximum score

k = number of neighbors for the maximum score
odd number is preferred to ommit equal voting of neighbors

```
score = dict()
up = 10

for k in range(1,up,2):
    temp_knn = KNeighborsClassifier(n_neighbors=k).fit(x_train,y_train)
    temp_score = temp_knn.score(x_test,y_test)
    score[k] = temp_score

best_k = max(score,key=lambda i:score[i])
print('Best k =',best_k)
```

```
Best k = 7
```

training

```
knn = KNeighborsClassifier(n_neighbors=best_k)
knn.fit(x_train,y_train)

print('KNeighborsClassifier: Training accuracy:',knn.score(x_train,y_train))
```

```
KNeighborsClassifier: Training accuracy: 0.8753006552210334
```

testing

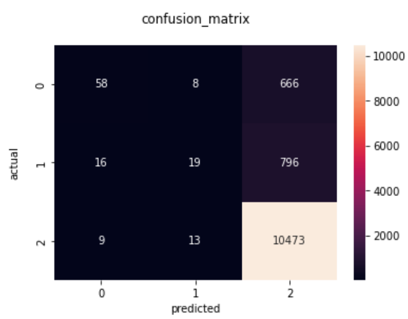
```
y_knn = knn.predict(x_test)

print('KNeighborsClassifier: Testing accuracy:',accuracy_score(y_test,y_knn))
```

```
KNeighborsClassifier: Testing accuracy: 0.8749378006302869
```

evaluation

```
sns.heatmap(confusion_matrix(y_test,y_knn),annot=True,fmt='d')
plt.title('confusion_matrix\n')
plt.xlabel('predicted\n')
plt.ylabel('actual\n')
```

```
Text(33.0, 0.5, 'actual\n')
```




```
print('Classification report:\n',classification_report(y_test,y_knn))
```

```
Classification report:
               precision    recall  f1-score   support

           0       0.70      0.08      0.14       732
           1       0.47      0.02      0.04       831
           2       0.88      1.00      0.93     10495

    accuracy                           0.87     12058
   macro avg       0.68      0.37      0.37     12058
weighted avg       0.84      0.87      0.82     12058
```

Naive-Bayes

```
from sklearn.naive_bayes import MultinomialNB
```

The Naive Bayes classification algorithm is a probabilistic classifier based on the bayes theorem. It is based on probability models that incorporate strong independence assumptions.

training

```
nb = MultinomialNB()
nb.fit(x_train,y_train)

print('MultinomialNB: Taining accuracy:',nb.score(x_train,y_train))
```

```
MultinomialNB: Taining accuracy: 0.8709256033839263
```

testing

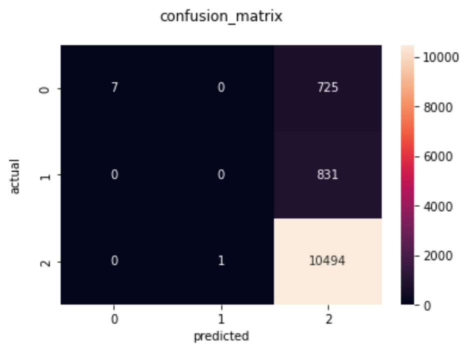
```
y_nb = nb.predict(x_test)
print('MultinomialNB: Testing accuracy:',accuracy_score(y_test,y_nb))
```

```
MultinomialNB: Testing accuracy: 0.8708741084757008
```

evaluation

```
sns.heatmap(confusion_matrix(y_test,y_nb),annot=True,fmt='d')
plt.title('confusion_matrix\n')
plt.xlabel('predicted\n')
plt.ylabel('actual\n')
```

```
Text(33.0, 0.5, 'actual\n')
```




```
print('Classification report:\n',classification_report(y_test,y_nb))
```

```
Classification report:
               precision    recall  f1-score   support

           0       1.00      0.01      0.02       732
           1       0.00      0.00      0.00       831
           2       0.87      1.00      0.93     10495

    accuracy                           0.87     12058
   macro avg       0.62      0.34      0.32     12058
weighted avg       0.82      0.87      0.81     12058
```

DecisionTreeClassifier

```
from sklearn import tree
from sklearn.tree import plot_tree
```

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import KFold, cross_val_score
```

Is a tree-structured classifier where the data is continuously split according to a certain parameter, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.

finding the parameters for maximum score

parameters: max_dept, min_samples_leaf

```
max_depth = list(range(2,20,2))
cv=KFold(n_splits=5)

score = dict()

for depth in max_depth:
    iscore = cross_val_score(tree.DecisionTreeClassifier(max_depth=depth, random_state=777),X,Y,cv=cv,scoring="accuracy")
    score[depth] = iscore

depth = max(score,key=lambda i:score[i].mean())
print('Best depth =',depth)
```

```
Best depth = 2
```

```
min_sam_leaf = list(range(10,100,10))

score = dict()

for min_sam in min_sam_leaf:
    iscore = cross_val_score(tree.DecisionTreeClassifier(min_samples_leaf=min_sam,random_state=777)
                             ,X,Y,cv=cv,scoring="accuracy")
    score[min_sam] = iscore

min_sam = max(score,key=lambda i:score[i].mean())
print('Best min_sam_leaf =',min_sam)
```

```
Best min_sam_leaf = 90
```

training

```
dtc = DecisionTreeClassifier(max_depth=depth,min_samples_leaf=min_sam)
dtc.fit(x_train,y_train)

print('DecisionTreeClassifier: Training accuracy:',dtc.score(x_train,y_train))
```

```
DecisionTreeClassifier: Training accuracy: 0.870614580741478
```

testing

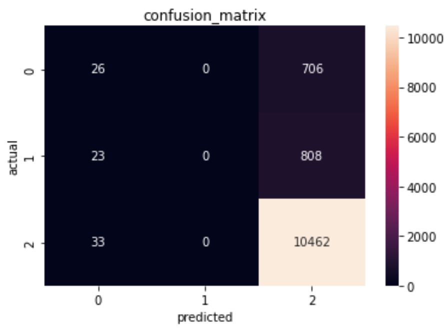
```
y_dtc = dtc.predict(x_test)
print('DecisionTreeClassifier: Testing accuracy:',accuracy_score(y_test,y_dtc))
```

```
DecisionTreeClassifier: Testing accuracy: 0.8697959860673412
```

evaluation

```
sns.heatmap(confusion_matrix(y_test,y_dtc),annot=True,fmt='d')
plt.xlabel('predicted')
plt.ylabel('actual')
plt.title('confusion_matrix')
```

```
Text(0.5, 1.0, 'confusion_matrix')
```

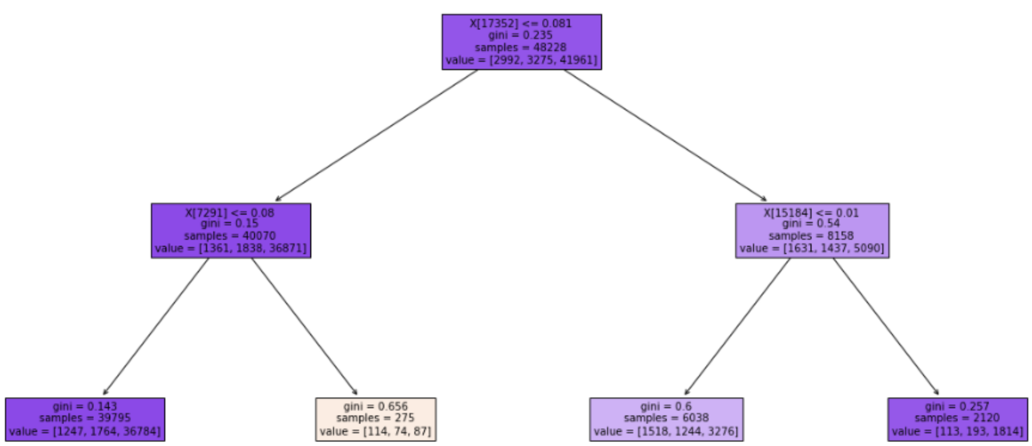



```
print('Classification report:\n',classification_report(y_test,y_dtc))
```

```
Classification report:
               precision    recall  f1-score   support

           0       0.32      0.04      0.06       732
           1       0.00      0.00      0.00       831
           2       0.87      1.00      0.93     10495

    accuracy                           0.87     12058
   macro avg       0.40      0.34      0.33     12058
weighted avg       0.78      0.87      0.81     12058
```

model representation

```
plt.figure(figsize=(20,10))
plot_tree(DecisionTreeClassifier(max_depth=depth,min_samples_leaf=min_sam,random_state=1).fit(x_train,y_train),
         fontsize=10,filled=True);
```



RandomForestClassifier

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
```

Random forest classifier is **an ensemble tree-based learning algorithm**. The random forest classifier is a set of decision trees from a randomly selected subset of the training set. It aggregates the votes from different decision trees to decide the final class of the test object.

training

```
params={'n_estimators':list(range(1,10,2))}

grid_search_rfc = GridSearchCV(estimator=RandomForestClassifier(max_depth=depth,min_samples_leaf=min_sam,random_state=777),
                               param_grid=params,cv=5,n_jobs=-1)
grid_search_rfc.fit(x_train,y_train)

print(grid_search_rfc.best_estimator_)
print('RandomForestClassifier: Training accuracy:',grid_search_rfc.best_estimator_.score(x_train,y_train))
```

```
RandomForestClassifier(max_depth=2, min_samples_leaf=90, n_estimators=1,
                       random_state=777)
RandomForestClassifier: Training accuracy: 0.8700547399850709
```

testing

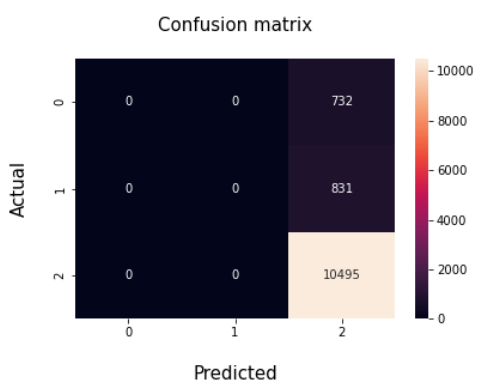
```
y_rfc = grid_search_rfc.predict(x_test)
print('RandomForestClassifier: Testing accuracy:',accuracy_score(y_test,y_rfc))
```

```
RandomForestClassifier: Testing accuracy: 0.8703765135179964
```

evaluation

```
sns.heatmap(confusion_matrix(y_test,y_rfc,),annot=True,fmt='d')
plt.xlabel('\nPredicted',fontsize=15)
plt.ylabel('Actual\n',fontsize=15)
plt.title('\nConfusion matrix\n',fontsize=15)
```

```
Text(0.5, 1.0, '\nConfusion matrix\n')
```



```
print('\nClassification report:')
print(classification_report(y_test,y_rfc))
```

```
Classification report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       732
           1       0.00      0.00      0.00       831
           2       0.87      1.00      0.93     10495

    accuracy                           0.87     12058
   macro avg       0.29      0.33      0.31     12058
weighted avg       0.76      0.87      0.81     12058
```

AdaBoost

Boosting ensemble technique: that is a combination of multiple weak learners learning happens sequential

```
from sklearn.ensemble import AdaBoostClassifier
```

training

```
ab=AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=depth, random_state=777),
                     random_state=777,n_estimators=1)
ab.fit(x_train,y_train)

print('AdaBoost: Taining accuracy:',ab.score(x_train,y_train))
```

```
AdaBoost: Taining accuracy: 0.870614580741478
```

testing

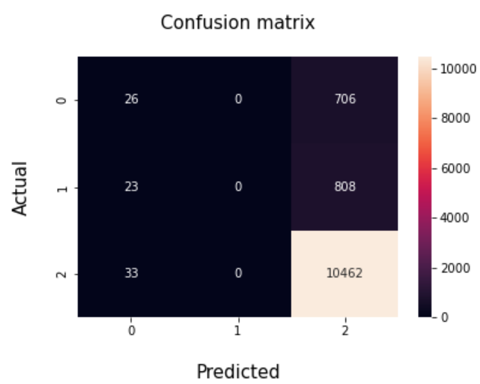
```
y_ab=ab.predict(x_test)
print('AdaBoost: Testing accuracy:',accuracy_score(y_test,y_ab))
```

```
AdaBoost: Testing accuracy: 0.8697959860673412
```

evaluation

```
sns.heatmap(confusion_matrix(y_test,y_ab,),annot=True,fmt='d')
plt.xlabel('\nPredicted',fontsize=15)
plt.ylabel('Actual\n',fontsize=15)
plt.title('\nConfusion matrix\n',fontsize=15)
```

```
Text(0.5, 1.0, '\nConfusion matrix\n')
```

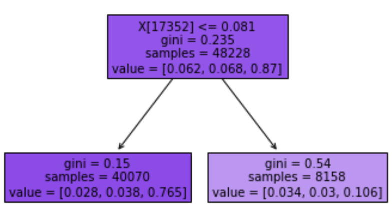



```
print('\nClassification report:')
print(classification_report(y_test,y_ab))
```

```
Classification report:
              precision    recall  f1-score   support

           0       0.32      0.04      0.06       732
           1       0.00      0.00      0.00       831
           2       0.87      1.00      0.93     10495

    accuracy                           0.87     12058
   macro avg       0.40      0.34      0.33     12058
weighted avg       0.78      0.87      0.81     12058
```

model representation

```
plot_tree(AdaBoostClassifier(random_state=777).fit(x_train,y_train).estimators_[0],fontsize=10,filled=True);
```



Stacking

```
from sklearn.ensemble import StackingClassifier
from sklearn.linear_model import LogisticRegression
```

Ensemble method in which the predictions, generated by using various machine learning algorithms, are used as inputs in a second-layer learning algorithm. This second-layer algorithm is trained to optimally combine the model predictions to form a new set of predictions

training

```
level0=[('dtree', DecisionTreeClassifier()),('nb',MultinomialNB()),('kneighbors',KNeighborsClassifier())]
level1 = LogisticRegression()

sc = StackingClassifier(estimators=level0,final_estimator=level1,cv=3).fit(x_train,y_train)
print('StackingClassifier: Training accuracy:',sc.score(x_train,y_train))
```

```
StackingClassifier: Training accuracy: 0.947520112797545
```

testing

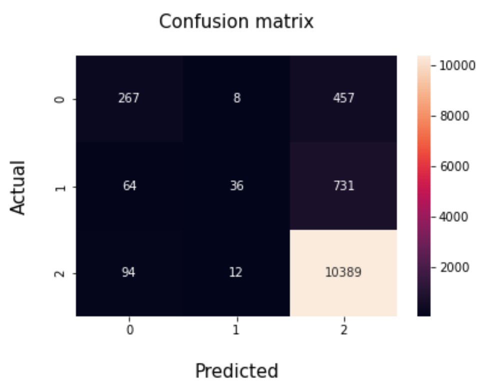
```
y_sc=sc.predict(x_test)
print('StackingClassifier: Testing accuracy:',accuracy_score(y_test,y_sc))
```

```
StackingClassifier: Testing accuracy: 0.8867142146292918
```

evaluation

```
sns.heatmap(confusion_matrix(y_test,y_sc,),annot=True,fmt='d')
plt.xlabel('\nPredicted',fontsize=15)
plt.ylabel('Actual\n',fontsize=15)
plt.title('\nConfusion matrix\n',fontsize=15)
```

```
Text(0.5, 1.0, '\nConfusion matrix\n')
```




```
print('\nClassification report:')
print(classification_report(y_test,y_sc))
```

```
Classification report:
              precision    recall  f1-score   support

           0       0.63      0.36      0.46       732
           1       0.64      0.04      0.08       831
           2       0.90      0.99      0.94     10495

    accuracy                           0.89     12058
   macro avg       0.72      0.47      0.49     12058
weighted avg       0.86      0.89      0.85     12058
```

Logistic Regression

```
from sklearn.multiclass import OneVsRestClassifier
```

training

```
score = dict()
up = 200

for i in range(100,up,50):
    temp_lr = LogisticRegression(max_iter=i)
    temp_ovr = OneVsRestClassifier(temp_lr).fit(x_train,y_train)

    temp_score = temp_ovr.score(x_test,y_test)
    score[i] = temp_score

best_i = max(score,key=lambda i:score[i])
print('Best i =',best_i)
```

```
Best i = 100
```

```
lr = LogisticRegression(max_iter=best_i)
ovr = OneVsRestClassifier(lr)
ovr.fit(x_train, y_train)

print('StackingClassifier: Training accuracy:',ovr.score(x_train,y_train))
```

```
StackingClassifier: Training accuracy: 0.9134735008708634
```

testing

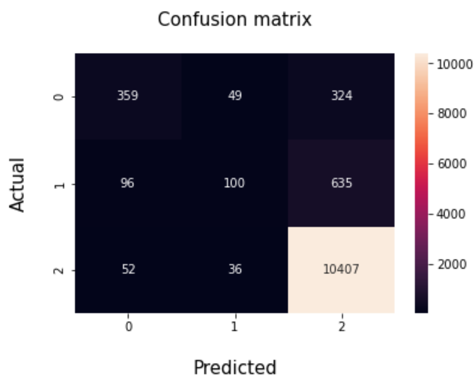
```
y_lr = ovr.predict(x_test)
print('LogisticRegression: Testing accuracy:',accuracy_score(y_test,y_lr))
```

```
LogisticRegression: Testing accuracy: 0.9011444684027202
```

evaluation

```
sns.heatmap(confusion_matrix(y_test,y_lr,),annot=True,fmt='d')
plt.xlabel('\nPredicted',fontsize=15)
plt.ylabel('Actual\n',fontsize=15)
plt.title('\nConfusion matrix\n',fontsize=15)
```

```
Text(0.5, 1.0, '\nConfusion matrix\n')
```




```
print('\nClassification report:')
print(classification_report(y_test,y_lr))
```

```
Classification report:
              precision    recall  f1-score   support

           0       0.71      0.49      0.58       732
           1       0.54      0.12      0.20       831
           2       0.92      0.99      0.95     10495

    accuracy                           0.90     12058
   macro avg       0.72      0.53      0.58     12058
weighted avg       0.88      0.90      0.88     12058
```

Comparison

	accuracy	precision	recall	f1_score
KNeighborsClassifier	0.874938	0.838915	0.874938	0.824438
MultinomialNB	0.870874	0.818693	0.870874	0.811417
DecisionTreeClassifier	0.869796	0.779592	0.869796	0.814334
RandomForestClassifier	0.870377	0.757555	0.870377	0.810056
AdaBoost	0.869796	0.779592	0.869796	0.814334
StackingClassifier	0.886714	0.863503	0.886714	0.852962
LogisticRegression	0.901144	0.877177	0.901144	0.877437

OBSERVATION

inorder to decide the best comparitive model:

1. overall metrics
2. individual f1 scores

LogisticRegression is the best comparitive model on unbalanced data

though the combined f1_score is good, the data being unbalanced the individual class f1_scores are not good

Observation

Data is imbalanced, hence the data has to be made balanced for efficient machine learning
Technique to make it balanced: SMOTE - requires numeric data

Synthetic Minority Oversampling Technique (SMOTE)

SMOTE stands for Synthetic Minority Oversampling Technique, is an oversampling technique that creates synthetic minority class data points to balance the dataset.

SMOTE works using a k-nearest neighbor algorithm to create synthetic data points.

The steps of SMOTE algorithm is:

1. Identify the minority class vector.
2. Decide the number of nearest numbers (k), to consider.
3. Compute a line between the minority data points and any of its neighbors and place a synthetic point.

Pictorial representation of SMOTE

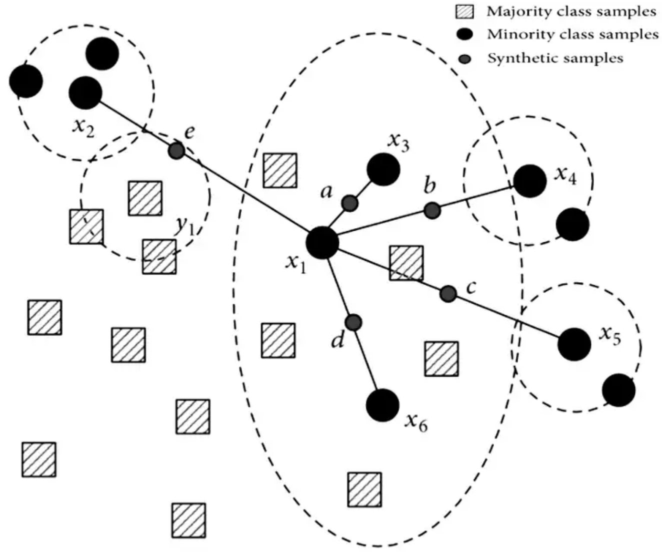


Image reference in bibliography

```
from imblearn.over_sampling import SMOTE
```

sample imbalance data

```
sample['sentiment'].value_counts()
```

```
Positive    52456
Neutral      4106
Negative     3724
Name: sentiment, dtype: int64
```

```
smote = SMOTE(random_state=777)
Xb,Yb = smote.fit_resample(X,Y)

print('x:',Xb.shape,'y:',Yb.shape)
```

```
x: (157368, 29289) y: (157368,)
```

Data splitting

```
x_trainb,x_testb,y_trainb,y_testb = train_test_split(Xb,Yb,test_size=0.2,random_state=777)

print('x:',x_trainb.shape,x_testb.shape)
print('y:',y_trainb.shape,y_testb.shape)
```

```
x: (125894, 29289) (31474, 29289)
y: (125894,) (31474,)
```

sample balanced data

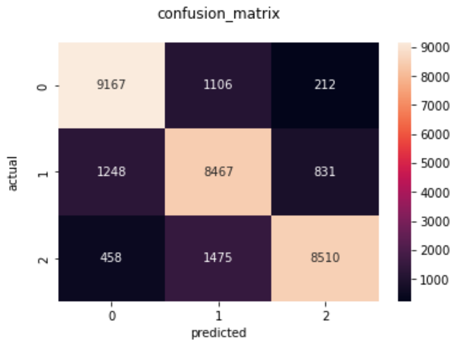
```
pd.DataFrame(y_trainb).value_counts()
```

```
2    42013
0    41971
1    41910
dtype: int64
```

Naive-Bayes

```
sns.heatmap(confusion_matrix(y_testb,y_nbb),annot=True,fmt='d')
plt.title('confusion_matrix\n')
plt.xlabel('predicted\n')
plt.ylabel('actual\n')
```

```
Text(33.0, 0.5, 'actual\n')
```



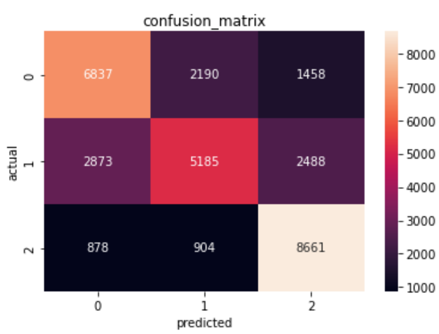

```
print('Classification report:\n',classification_report(y_testb,y_nbb))
```

```
Classification report:
               precision    recall  f1-score   support

           0       0.84      0.87      0.86     10485
           1       0.77      0.80      0.78     10546
           2       0.89      0.81      0.85     10443

    accuracy                           0.83     31474
   macro avg       0.83      0.83      0.83     31474
weighted avg       0.83      0.83      0.83     31474
```

DecisionTreeClassifier

```
sns.heatmap(confusion_matrix(y_testb,y_dtcb),annot=True,fmt='d')
plt.xlabel('predicted')
plt.ylabel('actual')
plt.title('confusion_matrix')
```

```
Text(0.5, 1.0, 'confusion_matrix')
```



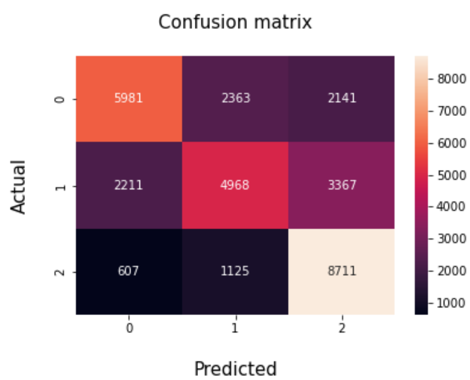

```
print('Classification report:\n',classification_report(y_testb,y_dtcb))
```

```
Classification report:
               precision    recall  f1-score   support

           0       0.65      0.65      0.65     10485
           1       0.63      0.49      0.55     10546
           2       0.69      0.83      0.75     10443

    accuracy                           0.66     31474
   macro avg       0.65      0.66      0.65     31474
weighted avg       0.65      0.66      0.65     31474
```

RandomForestClassifier

```
sns.heatmap(confusion_matrix(y_testb,y_rfcb),annot=True,fmt='d')
plt.xlabel('\nPredicted',fontsize=15)
plt.ylabel('Actual\n',fontsize=15)
plt.title('\nConfusion matrix\n',fontsize=15)
```

```
Text(0.5, 1.0, '\nConfusion matrix\n')
```

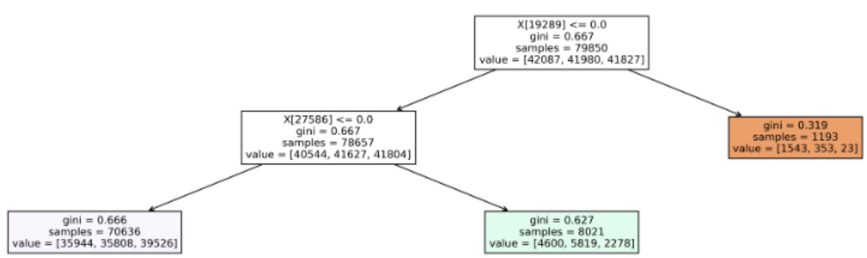



```
print('\nClassification report:')
print(classification_report(y_testb,y_rfcb))
```

```
Classification report:
              precision    recall  f1-score   support

           0       0.68      0.57      0.62     10485
           1       0.59      0.47      0.52     10546
           2       0.61      0.83      0.71     10443

    accuracy                           0.62     31474
   macro avg       0.63      0.63      0.62     31474
weighted avg       0.63      0.62      0.62     31474
```

model representation

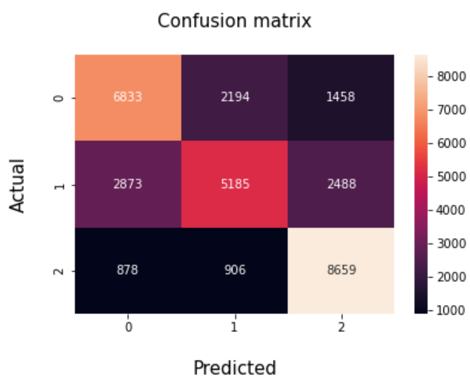
```
plt.figure(figsize=(20,5),dpi=300)
plot_tree(RandomForestClassifier(max_depth=2,min_samples_leaf=90,n_estimators=1,random_state=777).
          fit(x_trainb,y_trainb).estimators_[0],fontsize=10,filled=True);
```



AdaBoost

```
sns.heatmap(confusion_matrix(y_testb,y_abb),annot=True,fmt='d')
plt.xlabel('\nPredicted',fontsize=15)
plt.ylabel('Actual\n',fontsize=15)
plt.title('\nConfusion matrix\n',fontsize=15)
```

```
Text(0.5, 1.0, '\nConfusion matrix\n')
```



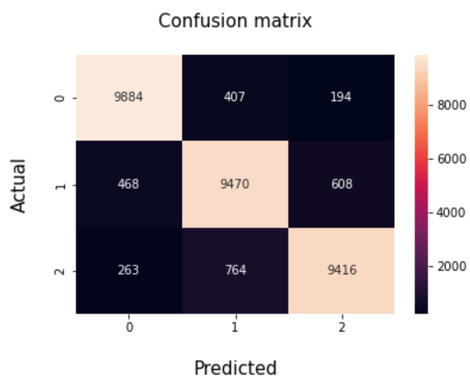

```
print('\nClassification report:')
print(classification_report(y_testb,y_abb))
```

```
Classification report:
              precision    recall  f1-score   support

           0       0.65      0.65      0.65     10485
           1       0.63      0.49      0.55     10546
           2       0.69      0.83      0.75     10443

    accuracy                           0.66     31474
   macro avg       0.65      0.66      0.65     31474
weighted avg       0.65      0.66      0.65     31474
```

Stacking

```
sns.heatmap(confusion_matrix(y_testb,y_scb),annot=True,fmt='d')
plt.xlabel('\nPredicted',fontsize=15)
plt.ylabel('Actual\n',fontsize=15)
plt.title('\nConfusion matrix\n',fontsize=15)
```

```
Text(0.5, 1.0, '\nConfusion matrix\n')
```



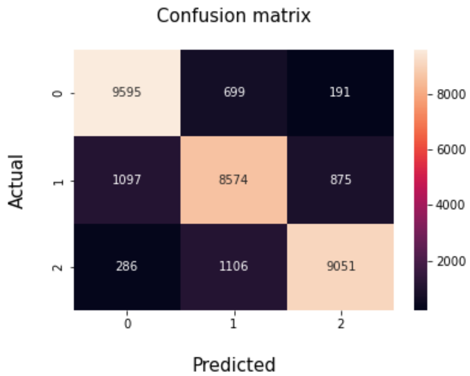

```
print('\nClassification report:')
print(classification_report(y_testb,y_scb))
```

```
Classification report:
              precision    recall  f1-score   support

           0       0.93      0.94      0.94     10485
           1       0.89      0.90      0.89     10546
           2       0.92      0.90      0.91     10443

    accuracy                           0.91     31474
   macro avg       0.91      0.91      0.91     31474
weighted avg       0.91      0.91      0.91     31474
```

Logistic Regression

```
sns.heatmap(confusion_matrix(y_testb,y_lrb),annot=True,fmt='d')
plt.xlabel('\nPredicted',fontsize=15)
plt.ylabel('Actual\n',fontsize=15)
plt.title('\nConfusion matrix\n',fontsize=15)
```

```
Text(0.5, 1.0, '\nConfusion matrix\n')
```




```
print('\nClassification report:')
print(classification_report(y_testb,y_lrb))
```

```
Classification report:
              precision    recall  f1-score   support

           0       0.87      0.92      0.89     10485
           1       0.83      0.81      0.82     10546
           2       0.89      0.87      0.88     10443

    accuracy                           0.86     31474
   macro avg       0.86      0.86      0.86     31474
weighted avg       0.86      0.86      0.86     31474
```

Comparison

on unbalanced data

```
unbal.T.style.background_gradient(cmap='Reds')
```

	accuracy	precision	recall	f1_score
KNeighborsClassifier	0.874938	0.838915	0.874938	0.824438
MultinomialNB	0.870874	0.818693	0.870874	0.811417
DecisionTreeClassifier	0.869796	0.779592	0.869796	0.814334
RandomForestClassifier	0.870377	0.757555	0.870377	0.810056
AdaBoost	0.869796	0.779592	0.869796	0.814334
StackingClassifier	0.886714	0.863503	0.886714	0.852962
LogisticRegression	0.901144	0.877177	0.901144	0.877437

on balanced data

```
bal.T.style.background_gradient(cmap='Blues')
```

	accuracy	precision	recall	f1_score
MultinomialNB	0.830654	0.833227	0.830654	0.831143
DecisionTreeClassifier	0.657146	0.652907	0.657146	0.650088
RandomForestClassifier	0.624643	0.626569	0.624643	0.616242
AdaBoost	0.656955	0.652694	0.656955	0.649908
StackingClassifier	0.914088	0.914143	0.914088	0.914062
LogisticRegression	0.864841	0.864800	0.864841	0.864572

OBSERVATION

inorder to decide the best comparitive model:

1. overall metrics
2. individual f1 scores

after balancing all the evaluations metrics have decreased for all the models except StackingClassifier
though the evaluation metrics has decreased

on comparing the confusion matrices and the evaluation metrics- **StackingClassifier** trained on balanced data is comparitively the best model

3.6 Conclusion

Based on the classification model- Positive, negative, neutral

```
print('Of AMAZON number of products with positive sentiment')
len(Positive['asin'])/len(df)*100
```

```
Of AMAZON number of products with positive sentiment
82.52465110261278
```

```
print('Of AMAZON number of categories with positive sentiment')
len(Positive['category'])/len(df)*100
```

```
Of AMAZON number of categories with positive sentiment
82.52465110261278
```

```
print('Of AMAZON number of brands with positive sentiment')
len(Positive['brand'])/len(df)*100
```

```
Of AMAZON number of brands with positive sentiment
82.52465110261278
```

```
print('Of AMAZON number of products with negative sentiment')
len(Negative['asin'])/len(df)*100
```

```
Of AMAZON number of products with negative sentiment

7.2251015362515725
```

```
print('Of AMAZON number of categories with negative sentiment')
len(Negative['category'])/len(df)*100
```

```
Of AMAZON number of categories with negative sentiment

7.2251015362515725
```

```
print('Of AMAZON number of brands with negative sentiment')
len(Negative['brand'])/len(df)*100
```

```
Of AMAZON number of brands with negative sentiment
```

```
print('Of AMAZON number of products with neutral sentiment')
len(Neutral['asin'])/len(df)*100
```

```
Of AMAZON number of products with neutral sentiment

10.250247361135646
```

```
print('Of AMAZON number of categories with neutral sentiment')
len(Neutral['category'])/len(df)*100
```

```
Of AMAZON number of categories with neutral sentiment

10.250247361135646
```

```
print('Of AMAZON number of brands with neutral sentiment')
len(Neutral['brand'])/len(df)*100
```

```
Of AMAZON number of brands with neutral sentiment

10.250247361135646
```

Positive negative neutral category

```
print('Top 5 categories with positive products:')
sorted(pos_cat_count,key=lambda i:pos_cat_count[i],reverse=True)[:5]
```

```
Top 5 categories with positive products:

["['Toys & Games', 'Action Figures & Statues', 'Action Figures']",
 "['Toys & Games', 'Building Toys', 'Building Sets']",
 "['Toys & Games', 'Dolls & Accessories', 'Dolls']",
 "['Toys & Games', 'Games', 'Board Games']",
 "['Toys & Games', 'Stuffed Animals & Plush Toys', 'Stuffed Animals & Teddy Bears']"]
```

```
print('Bottom 5 categories with positive products:')
sorted(pos_cat_count,key=lambda i:pos_cat_count[i],reverse=False)[:5]
```

```
Bottom 5 categories with positive products:

["['Toys & Games', 'Action Figures & Statues', 'Amazing shark that grows in water', 'Measures 1.50 feet long after 7 to 14 day
s', 'As it dries, it shrinks to original size', 'Kids learn various science concepts']",
 '[\'Toys & Games\', \'Dress Up & Pretend Play\', \'Costumes\', \'100% Synthetic Fiber\', \'Imported\', \'3-Piece face and body
hair kit\', \'Includes moustache, chest hair and sideburns\', "Black 70\'s Afro wig available separately from Forum", "Part of
the 70\'s Disco Fever collection", "Look to Forum Novelties for all your Halloween, Luau, Easter, Mardi Gras and St. Patrick\'s
Day supplies", \'Designed for ages 14 + adult\']',
 "['Toys & Games', 'Dress Up & Pretend Play', 'Accessories', 'Plastic', 'Officially licensed Teenage Mutant Ninja Turtle 2 Leon
ardo katana', 'Measures 26 x 2.75 x 1-inches', 'Designed to be used by children and teens', 'Costume accessory, not intended fo
r play', 'Look for TMNT costumes and accessories for babies, children, adults, and pets']",
 "['Toys & Games', 'Party Supplies', 'Party Favors', 'Size: 8oz.']",
 '[\'Toys & Games\', \'Dress Up & Pretend Play\', \'Costumes\', \'Includes one chef hat and one chef apron\', \'Child size for
3-5 year olds\', \'Apron and Hat dimensions: Apron length approx. 20.5", Apron width approx. 15", Neck opening approx. 22", Hat
circumference 21-22"\', \'Soft touch closure in the back of hat for adjustable fit\', \'Can be embellished or decorated, Poly/c
otton fabric\']']
```

```
print('Top 5 categories with negative products:')
sorted(neg_cat_count,key=lambda i:neg_cat_count[i],reverse=True)[:5]
```

```
Top 5 categories with negative products:

["['Toys & Games', 'Action Figures & Statues', 'Action Figures']",
 "['Toys & Games', 'Dolls & Accessories', 'Dolls']",
 "['Toys & Games', 'Games', 'Board Games']",
 "['Toys & Games', 'Building Toys', 'Building Sets']",
 "['Toys & Games', 'Sports & Outdoor Play', 'Blasters & Foam Play']"]
```

```
print('Bottom 5 categories with negative products:')
sorted(neg_cat_count,key=lambda i:neg_cat_count[i],reverse=False)[:5]
```

```
Bottom 5 categories with negative products:

["['Toys & Games', 'Action Figures & Statues', 'Action Figures', 'This is a necklace with a replica Save Crystal pendant from t
he Final Fantasy game series!', 'The blue pendant is about 2-inches long. It is wrapped in a silver-color frame for added styl
e. The chain is approx 19-inches.', 'This is a great item to wear and show support from the Final Fantasy series!']",
 '[\'Toys & Games\', \'Dress Up & Pretend Play\', \'Accessories\', \'Fun costumes for kids and adults\', "Whether it\'s for Hal
loween, a themed party, or even for giggles", \'Beautiful colors, hand-wash needed, excellent for dress up\']',
 "['Toys & Games', 'Dress Up & Pretend Play', 'Pretend Play', 'Plastic', 'Silver rings that slide over nose or lip to look like
piercings.Silver color.', 'Manufacturer minimum age: 48 Months']",
 "['Toys & Games', 'Dress Up & Pretend Play', 'Costumes', '100% acrylic', 'Tiger Costume for Children.', 'Four sizes. 4-6, 6-8,
8-10 and 10-12 Years.', 'Costume includes jumpsuit with tail , mittens and hood.', 'Machine washable. 100% Acrylic.', 'Conforms
to European & North American safety regulations. As a precaution keep away from fire. For more information view the product det
ails below.']",
 "['Toys & Games', 'Party Supplies', 'plastic', 'Plastic. Phoney Dynamite party accessory']"]
```

```
print('Top 5 categories with neutral products:')
sorted(neu_cat_count,key=lambda i:neu_cat_count[i],reverse=True)[:5]
```

```
Top 5 categories with neutral products:

["['Toys & Games', 'Action Figures & Statues', 'Action Figures']",
 "['Toys & Games', 'Games', 'Board Games']",
 "['Toys & Games', 'Dolls & Accessories', 'Dolls']",
 "['Toys & Games', 'Building Toys', 'Building Sets']",
 "['Toys & Games', 'Stuffed Animals & Plush Toys', 'Stuffed Animals & Teddy Bears']"]
```

```
print('Bottom 5 categories with neutral products:')
sorted(neu_cat_count,key=lambda i:neu_cat_count[i],reverse=False)[:5]
```

```
Bottom 5 categories with neutral products:

["['Toys & Games', 'Hobbies', 'Remote & App Controlled Vehicles & Parts', 'Remote & App Controlled Vehicle Parts', 'Power Plant
& Driveline Systems', 'Brakes']",
 "['Toys & Games', 'Hobbies', 'Trains & Accessories']",
 "['Toys & Games', 'Sports & Outdoor Play', 'Sports', 'Baseball & Softball', 'Batting Gloves']",
 "['Toys & Games', 'Dress Up & Pretend Play', 'Pretend Play', 'Plastic', 'Silver rings that slide over nose or lip to look like
piercings.Silver color.', 'Manufacturer minimum age: 48 Months']",
 "['Toys & Games', 'Dress Up & Pretend Play', 'Costumes', '100% acrylic', 'Tiger Costume for Children.', 'Four sizes. 4-6, 6-8,
8-10 and 10-12 Years.', 'Costume includes jumpsuit with tail , mittens and hood.', 'Machine washable. 100% Acrylic.', 'Conforms
to European & North American safety regulations. As a precaution keep away from fire. For more information view the product det
ails below.']"]
```

Positive negative neutral brand

```
print('Top 5 brands with positive products:')
sorted(posb_brand_count,key=lambda i:posb_brand_count[i],reverse=True)[:5]
```

```
Top 5 brands with positive products:

['Melissa & Doug', 'LEGO', 'Fisher-Price', 'FunKo', 'Mattel']
```

```
print('Bottom 5 brands with positive products:')
sorted(posb_brand_count,key=lambda i:posb_brand_count[i],reverse=False)[:5]
```

```
Bottom 5 brands with positive products:

['Artesania Latina',
 'Power Glide',
 'Mercurius',
 'Swordsswords',
 'FlexiBlox Fidget']
```

```python
print('Top 5 brands with negative products:')
sorted(neg_brand_count,key=lambda i:neg_brand_count[i],reverse=True)[:5]
```

```
Top 5 brands with negative products:

['Melissa & Doug',
 'Fisher-Price',
 'Fun Express',
 'Rhode Island Novelty',
 'Mattel']
```

```python
print('Bottom 5 brands with negative products:')
sorted(neg_brand_count,key=lambda i:neg_brand_count[i],reverse=False)[:5]
```

```
Bottom 5 brands with negative products:

['Therapy Game HQ',
 'Yottoy',
 'STORY EGG',
 'The Home Fusion Company',
 'Cubicle 7']
```

```python
print('Top 5 brands with neutral products:')
sorted(neu_brand_count,key=lambda i:neu_brand_count[i],reverse=True)[:5]
```

```
Top 5 brands with neutral products:

['Melissa & Doug', 'Fisher-Price', 'LEGO', 'Mattel', 'Hasbro']
```

```python
print('Bottom 5 brands with neutral products:')
sorted(neu_brand_count,key=lambda i:neu_brand_count[i],reverse=False)[:5]
```

```
Bottom 5 brands with neutral products:

['Mudpuppy',
 'Frank Schaffer',
 'Evil Hat Productions',
 'Paizo Publishing',
 'Key Education']
```

4. Clustering

It is basically a type of unsupervised learning method that is used to divide the population or data points into a number of groups such that data points in the same groups are more similar to other data points in the same group and dissimilar to the data points in other groups. It is basically a collection of objects on the basis of similarity and dissimilarity between them.

4.1 Performing clustering on categories

Performing *group by* with respect to categories and taking mean of overall ratings and sum of price (Sales)

Inventory optimization: Clustering

Clustering is grouping of like objects together
Clustering of categories on the basis of rating and Total Sales

Clustering with respect to category

```
cat_count = df['category'].value_counts().to_frame()
cat_rating = df.groupby('category')['overall'].mean().to_frame()
cat_price = df.groupby('category')['price'].mean().to_frame()
cat_sales = df.groupby('category')['price'].sum().to_frame().rename(columns={'price':'sales'})

df_cat = cat_count.join(cat_rating).join(cat_price).join(cat_sales)
df_cat.head()
```

	category	overall	price	sales
['Toys & Games', 'Action Figures & Statues', 'Action Figures']	88024	4.558666	39.436296	3471340.51
['Toys & Games', 'Building Toys', 'Building Sets']	72569	4.683846	64.466345	4678258.17
['Toys & Games', 'Dolls & Accessories', 'Dolls']	63746	4.612133	43.025772	2742720.87
['Toys & Games', 'Games', 'Board Games']	58261	4.496456	28.386070	1653800.82
['Toys & Games', 'Stuffed Animals & Plush Toys', 'Stuffed Animals & Teddy Bears']	41676	4.711081	20.733123	864073.63

```
df_cat = df_cat.reset_index().rename(columns={'index':'category','category':'Freq'})

df_cat.head(3)
```

	category	Freq	overall	price	sales
0	['Toys & Games', 'Action Figures & Statues', '...	88024	4.558666	39.436296	3471340.51
1	['Toys & Games', 'Building Toys', 'Building Se...	72569	4.683846	64.466345	4678258.17
2	['Toys & Games', 'Dolls & Accessories', 'Dolls']	63746	4.612133	43.025772	2742720.87

4.2 Feature Engineering

Transforming data to machine understandable and processable form

4.2.1 Feature Scaling

The numeric distribution is normalized/standardized to boost the model performance

Feature engineering

transforming data to machine understandable and processable form

```
df_cat.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 717 entries, 0 to 716
Data columns (total 5 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   category  717 non-null    object
 1   Freq      717 non-null    int64
 2   overall   717 non-null    float64
 3   price     717 non-null    float64
 4   sales     717 non-null    float64
dtypes: float64(3), int64(1), object(1)
memory usage: 28.1+ KB
```

Feature scaling

the numeric distribution is normalized/standardized to boost the model performance

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

df_cat_num = df_cat.select_dtypes('number')
df_cat_num=df_cat_num.drop(['Freq','price'],axis=1)
df_cat_num_scaled = scaler.fit_transform(df_cat_num)
```

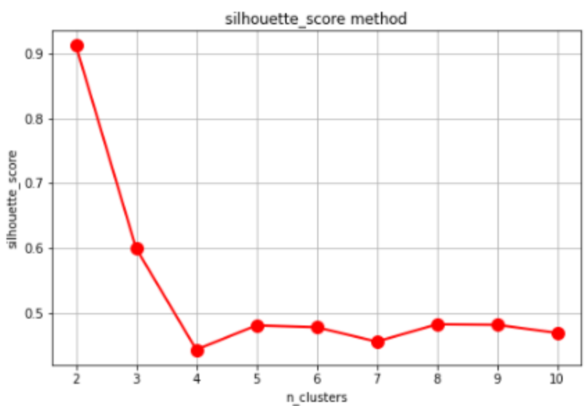
4.3 Modeling

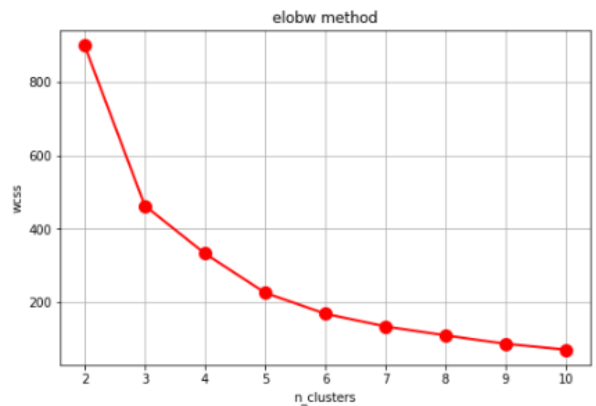
4.3.1 K Means

It is an iterative algorithm that divides the unlabeled dataset into k different clusters in such a way that each dataset belongs to only one group that has similar properties.

Selecting appropriate K Value:

1. Elbow Method
2. Silhouette score

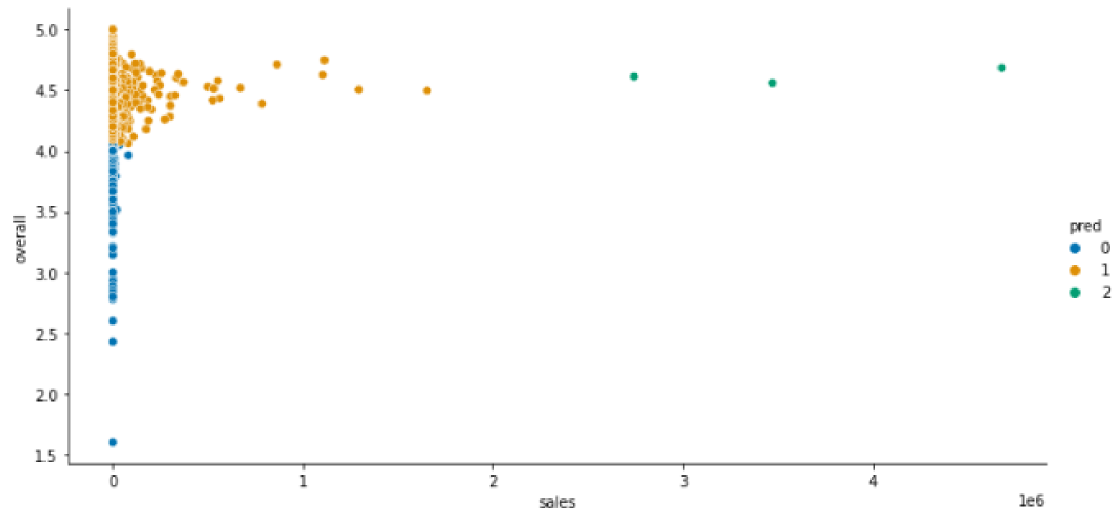




From both silhouette score and elbow method:
best K value = 3

```python
import seaborn as sns

kmeans = KMeans(n_clusters=3, random_state=777)
y_kmeans = kmeans.fit_predict(df_cat_num_scaled)

df_cat['pred']=y_kmeans
df_pred = pd.DataFrame({'y_kmeans':y_kmeans})

sns.relplot(data=df_cat,x='sales',y='overall',hue='pred',palette='colorblind',aspect=2)
```

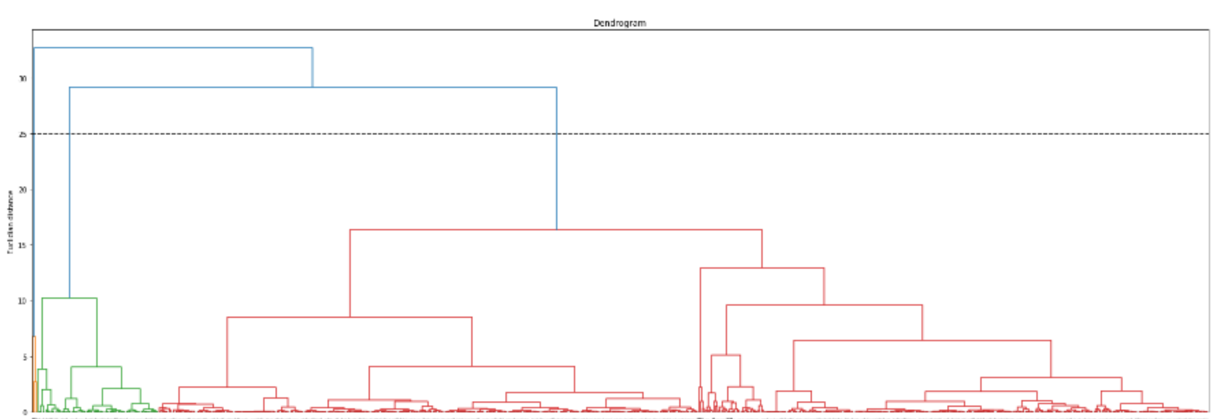
```
<seaborn.axisgrid.FacetGrid at 0x7f37e1d9ed00>
```



4.3.2 Agglomerative Clustering

Agglomerative clustering also known as bottom-up approach or hierarchical agglomerative clustering (HAC). This clustering algorithm does not require us to pre-specify the number of clusters. Bottom-up algorithms treat each data as a singleton cluster at the outset and then successively agglomerates pairs of clusters until all clusters have been merged into a single cluster that contains all data.

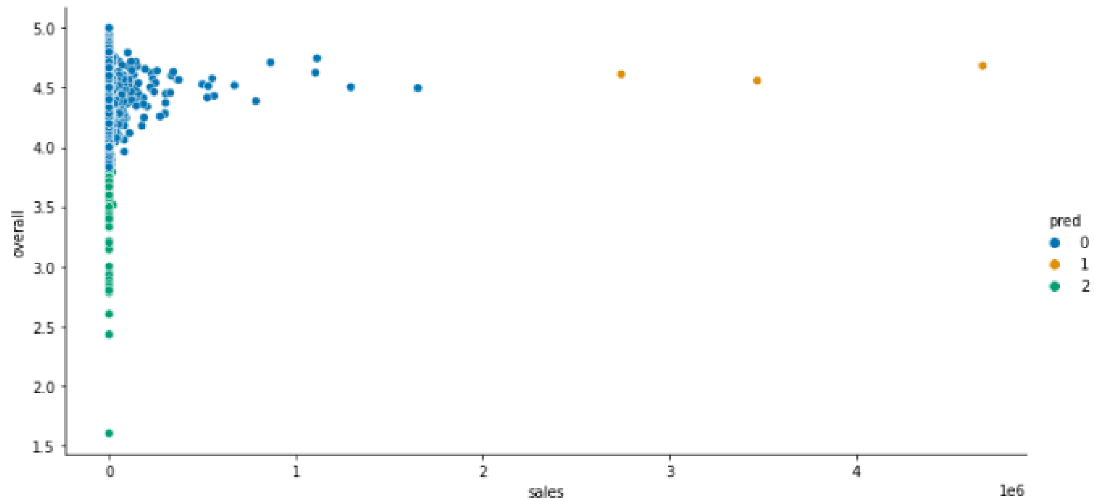
Dendrogram:



From Dendrogram k=3

```
hc = AgglomerativeClustering(n_clusters = 3, affinity = 'euclidean', linkage = 'ward')
y_hc = hc.fit_predict(df_cat_num_scaled)

df_cat['pred'] = y_hc
df_pred['y_hc'] = y_hc

sns.relplot(data=df_cat,x='sales',y='overall',hue='pred',palette='colorblind',aspect=2)
```

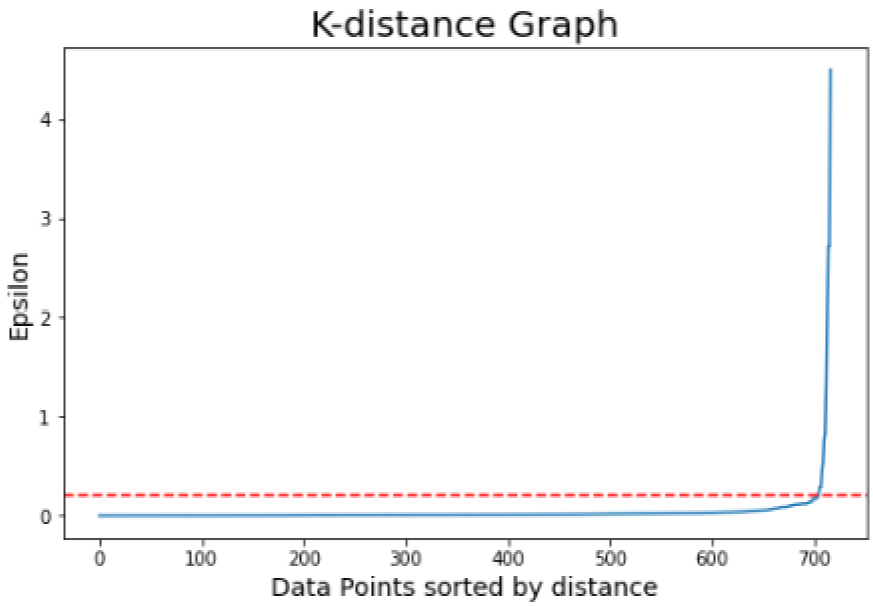
```
<seaborn.axisgrid.FacetGrid at 0x7f37e351afa0>
```



4.3.3 DBSCAN

DBSCAN stands for density-based spatial clustering of applications with noise. It is able to find arbitrary shaped clusters and clusters with noise (i.e. outliers). The main idea behind DBSCAN is that a point belongs to a cluster if it is close to many points from that cluster.

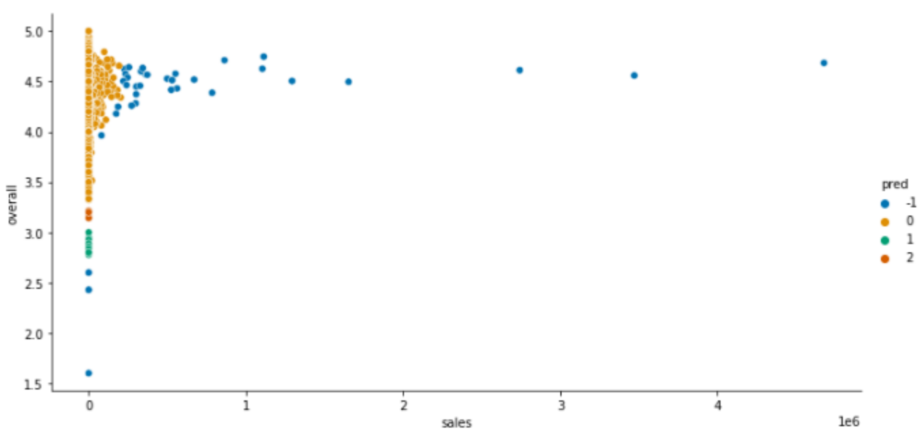
Selecting Epsilon value



breaking point at epsilon = 0.2

```
eps = 0.2; min_samples = 6

dbscan = DBSCAN(eps=eps, min_samples=min_samples)
y_dbscan = dbscan.fit_predict(df_cat_num_scaled)

df_cat['pred'] = y_dbscan
df_pred['y_dbscan'] = y_dbscan

sns.relplot(data=df_cat,x='sales',y='overall',hue='pred',palette='colorblind',aspect=2)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f37fa791670>
```



4.4 Comparison

Comparing the silhouette score of different models

Comparison

```
df_pred.head()
```

	y_kmeans	y_hc	y_dbscan
0	2	1	-1
1	2	1	-1
2	2	1	-1
3	1	0	-1
4	1	0	-1

silhouette_score is a clustering metric- measure of how well the clusters are formed
it explains how close intra-cluster points are and how far far clusters are

```
sil_kmeans = silhouette_score(df_cat_num_scaled,y_kmeans)
sil_hc = silhouette_score(df_cat_num_scaled,y_hc)
sil_dbscan = silhouette_score(df_cat_num_scaled,y_dbscan)

print('Silhouette score:')
print('Kmeans: {:.3f}'.format(sil_kmeans))
print('AgglomerativeClustering: {:.3f}'.format(sil_hc))
print('DBSCAN: {:.3f}'.format(sil_dbscan))
```

```
Silhouette score:
Kmeans: 0.600
AgglomerativeClustering: 0.652
DBSCAN: 0.570
```

INFERENCE

On comparing the silhouette scores best model: **AgglomerativeClustering**

4.5 Suggestion for inventory optimization

- First cluster (Moderate performing) has good ratings but average sales. As observed, the categories with less price are giving better sales and purchase frequency of those products is also high, so to improve the sales it is recommended to drop the price.

- Second cluster (Best performing) is having good ratings as well as sales. So, these categories should not be tampered.

- Third cluster (Needs Improvement) is having the worst response in terms of rating as well as sales. So, complete overhauling is required to improve their ratings.

5. Time series

Time series analysis is a specific way of analyzing a sequence of data points collected over an interval of time. In time series analysis, analysts record data points at consistent intervals over a set period of time rather than just recording the data points intermittently or randomly. However, this type of analysis is not merely the act of collecting data over time

5.1 Splitting Data into Sentiment

Using a filter to get positive, negative and neutral sentiments in 3 different dataframes, to forecast which sentiment has the best increase in ratings over the next few months.

Splitting data as per rating sentiment for forecast

```python
Positive = df[df['sentiment'] == 'Positive']
Positive.set_index('reviewTime',inplace=True)

Negative = df[df['sentiment'] == 'Negative']
Negative.set_index('reviewTime',inplace=True)

Neutral = df[df['sentiment'] == 'Neutral']
Neutral.set_index('reviewTime',inplace=True)

print('Positive')
display(Positive.head(3))
print('Negative')
display(Negative.head(3))
print('Neutral')
display(Neutral.head(3))
```

```
Positive
```

reviewTime	overall	verified	asin	reviewText	category	title	brand	price	sentiment	polarity	polarity_sentiment
1999-10-06	5.0	False	1572810939	ready heart meet spade whist ready play wizard...	['Toys & Games', 'Games', 'Card Games']	US Games Wizard Card Game Deluxe	US Games	12.72	Positive	0.081692	Neutral
2000-03-08	5.0	False	B00000IWFB	long ha toy yes awhile great play great stress...	['Toys & Games', 'Games']	Milton Bradley Bop It Extreme	Milton Bradley	39.63	Positive	0.157576	Positive
2000-03-08	5.0	False	B00000IWFB	long ha toy yes awhile great play great stress...	['Toys & Games', 'Games']	Milton Bradley Bop It Extreme	Milton Bradley	39.63	Positive	0.157576	Positive

Negative

reviewTime	overall	verified	asin	reviewText	category	title	brand	price	sentiment	polarity	polarity_sentiment
2001-06-07	1.0	True	B000059XI2	lugging book cartridge ziploc bag thought bag ...	['Toys & Games', "Kids' Electronics", 'Systems...	LeapPad Back Pack	LeapFrog	14.75	Negative	0.250000	Positive
2001-06-07	1.0	True	B000059XI2	lugging book cartridge ziploc bag thought bag ...	['Toys & Games', "Kids' Electronics", 'Systems...	LeapPad Back Pack	LeapFrog	14.75	Negative	0.250000	Positive
2001-08-27	1.0	False	B00000ISUK	toy wa dissapointing son love thomas toy poorl...	['Toys & Games', 'Toy Remote Control & Play Ve...	Thomas and Friends Train Play Set - Thomas Big...	TOMY	59.99	Negative	0.157143	Positive

Neutral

reviewTime	overall	verified	asin	reviewText	category	title	brand	price	sentiment	polarity	polarity_sentiment
2000-05-02	3.0	False	B00000IW4C	came price wa year ago said not getting commte...	['Toys & Games', 'Action Figures & Statues', '...	Star Wars CommTech Reader	Hasbro	19.95	Neutral	0.260000	Positive
2000-05-02	3.0	False	B00000IW4C	came price wa year ago said not getting commte...	['Toys & Games', 'Action Figures & Statues', '...	Star Wars CommTech Reader	Hasbro	19.95	Neutral	0.260000	Positive
2000-11-20	3.0	False	B00000DMFD	commercial look exciting kid fun game prepared...	['Toys & Games', 'Games', 'Board Games']	Mouse Trap Board Game For Kids Ages 6 and Up (...	Hasbro	24.00	Neutral	-0.047619	Negative

5.2 Resampling

The main purpose of resampling is to make the continuity which is the prerequisite of time series and reduces the data

Resampling of Data for positive reviews

```
Positive_resemple = Positive['price'].resample('m').sum()
Positive_resemple = Positive_resemple[Positive_resemple.index>'2002-01-01']
Positive_resemple
```

```
reviewTime
2002-01-31       537.35
2002-02-28       754.06
2002-03-31       146.48
2002-04-30       507.31
2002-05-31       635.21
                  ...
2018-06-30    132604.33
2018-07-31     95356.33
2018-08-31     49695.64
2018-09-30     11933.69
2018-10-31        69.57
Freq: M, Name: price, Length: 202, dtype: float64
```

Resampling for Negative reviews Data

```
Negative_resemple = Negative['price'].resample('m').sum()
Negative_resemple=Negative_resemple[Negative_resemple.index>='2008-01-01']
Negative_resemple
```

```
reviewTime
2008-01-31     1091.69
2008-02-29      436.20
2008-03-31      190.34
2008-04-30      468.18
2008-05-31      183.75
                 ...
2018-05-31    14228.13
2018-06-30     8385.44
2018-07-31     7018.27
2018-08-31     4218.90
2018-09-30     1785.18
Freq: M, Name: price, Length: 129, dtype: float64
```

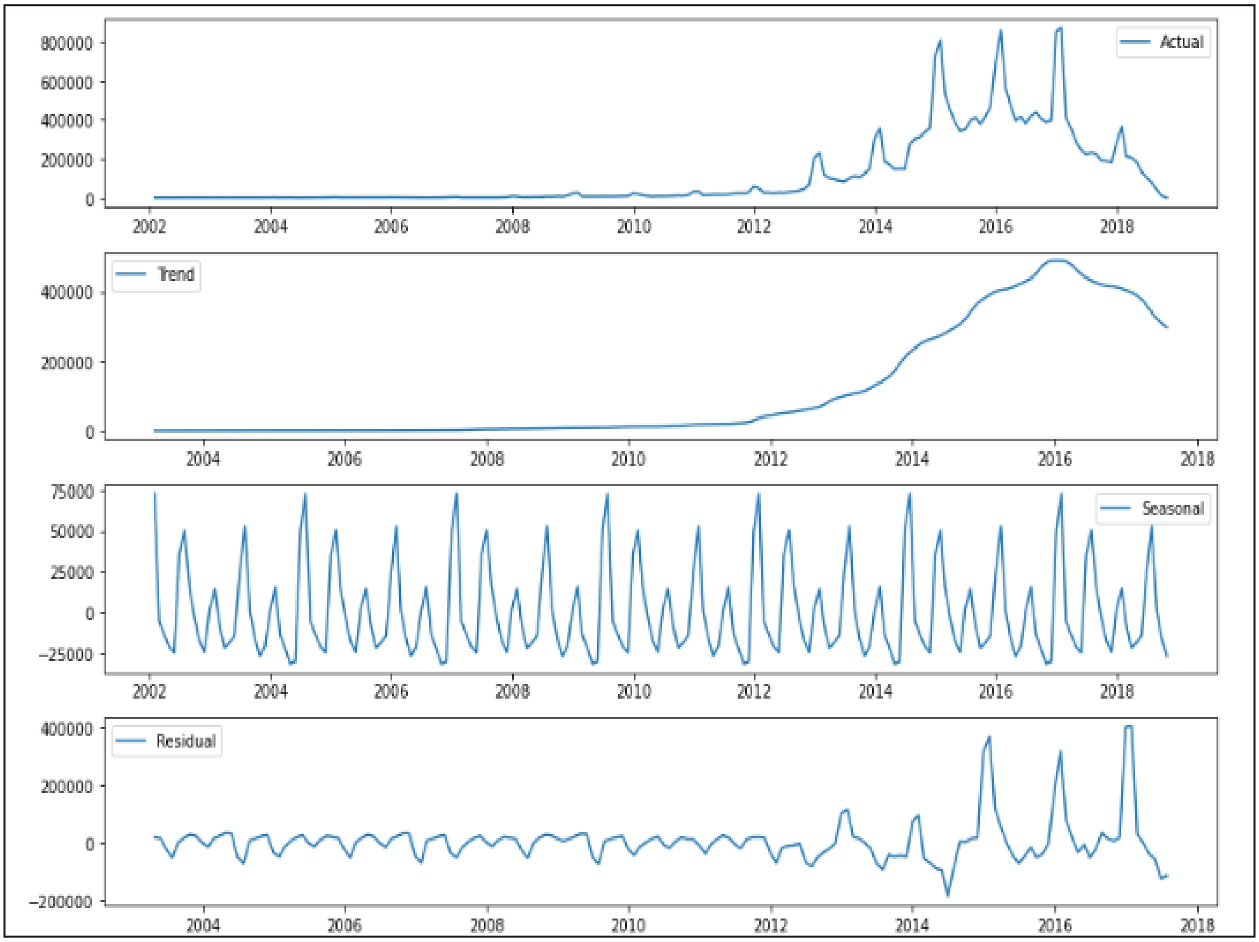
Resempling for neutral reviews

```
Neutral_resemple = Neutral['price'].resample('m').sum()

Neutral_resemple = Neutral_resemple[Neutral_resemple.index>'2007-01-01']
Neutral_resemple
```

```
reviewTime
2007-01-31     1937.58
2007-02-28      308.00
2007-03-31      518.45
2007-04-30      505.29
2007-05-31      228.37
                 ...
2018-05-31    12578.81
2018-06-30    10141.58
2018-07-31     6681.29
2018-08-31     3001.13
2018-09-30      990.58
Freq: M, Name: price, Length: 141, dtype: float64
```

5.3 Decomposition of data

Positive Trend Analysis

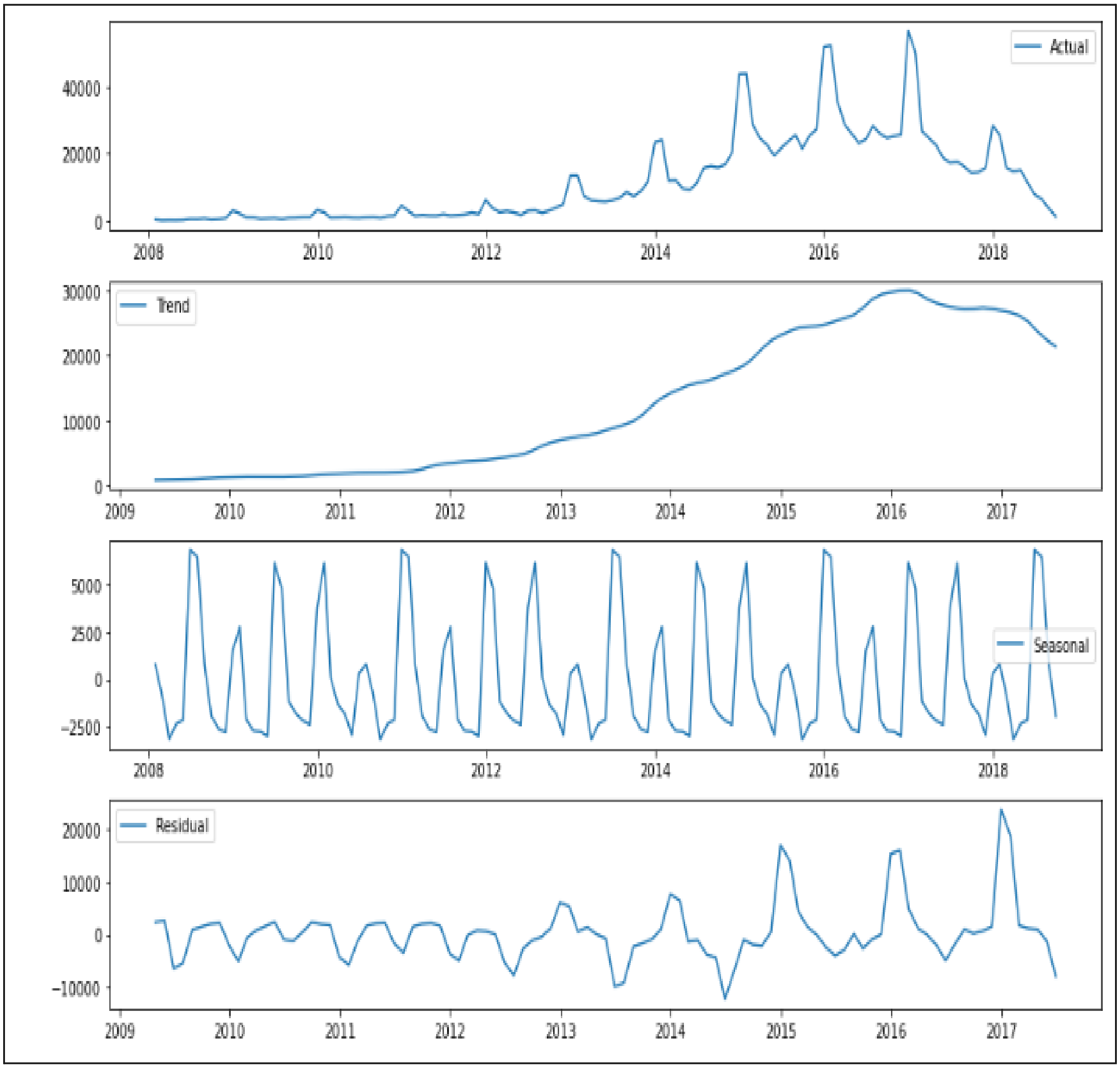


Trend: Sales increased till highest in 2016 and then started falling after 2016

Seasonality: seasonality is observed after every 2 years and the   fluctuation is between 2.5K to 75K .

Residuals: magnitude of residuals correspond to the trend

Negative Trend Analysis

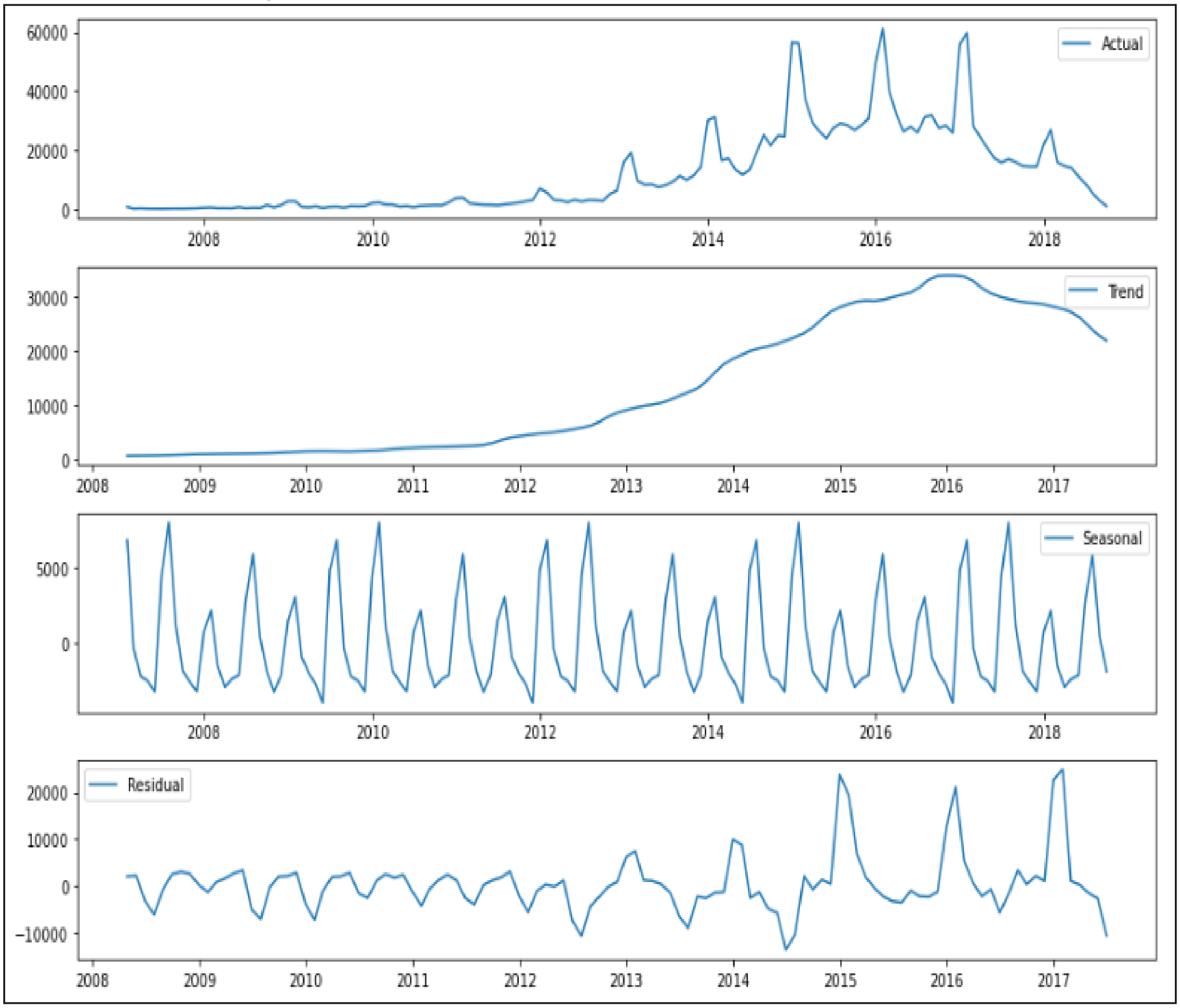


Trend: Sales increased till highest in 2016 and then started falling after 2016 .

Seasonality: seasonality is observed after every 2 years and the fluctuations are between 2.5K to 6K .

Residuals: magnitude of residuals correspond to the trend

Neutral Trend Analysis



Trend: Sales increased till highest in 2016 and then started falling after 2016

Seasonality: seasonality is observed after every 2 years and the fluctuations are between 1.5K to 6K .

Residuals: magnitude of residuals correspond to the trend

5.4 Stationary check

A time series has stationarity when the observations are not dependent on time. Statistical properties of these time series will not change with time thus they will have constant mean and variance.

Positive Trend

ADF test to check the stationarity of data

```
def checkStationarity(data):
    pvalue = adfuller(data)[1]
    if(pvalue>0.05):
        msg = 'p-value={}. Data is not stationary'.format(pvalue)
    else:
        msg='p-value={}. Data is stationary'.format(pvalue)

    return(msg)
```

```
checkStationarity(Positive_resemple)
```

```
'p-value=0.3184785690644165. Data is not stationary'
```

Here we can see that our Data is not stationary i.e. data don't have constant mean and constant variance over the time so for further processing we have to make data stationary so we can accurately do the statistical analysis.

Differencing method : subtracts the current value from the previous value .

```
shift1 = Positive_resemple - Positive_resemple.shift(8)
checkStationarity(shift1.dropna())
```

```
'p-value=0.001216688858739447. Data is stationary'
```

Here we can see that our Data is not stationary i.e. data don't have constant mean and constant variance over the time so for further processing we have made the data stationary using differencing method so we can accurately do the statistical analysis.

Negative Trend

ADF Stationary test

```
checkStationarity(Negative_resemple)
```

```
'p-value=0.659379147137551. Data is not stationary'
```

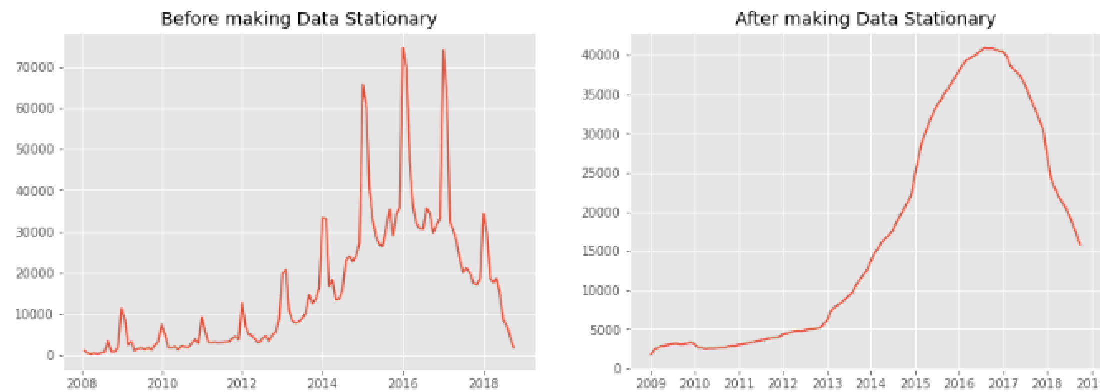
Rolling average to make data stationary

```
Stationary_negative = Negative_resemple.rolling(window=12).mean().dropna()
checkStationarity(Stationary_negative)
```

```
'p-value=0.24229853301933035. Data is not stationary'
```

Data Plot before Data stationary and after

```
fig,ax=plt.subplots(nrows=1,ncols=2,figsize=(15,5))
ax[0].plot(Negative_resemple)
ax[0].set_title('Before making Data Stationary')
ax[1].plot(Stationary_negative)
ax[1].set_title('After making Data Stationary')
plt.show()
```



Here we can see that our Data is not stationary i.e. data don't have constant mean and constant variance over the time so for further processing we have made the data stationary using rolling mean method so we can accurately do the statistical analysis.

Neutral Trend:

ADF Test for stationarity check

```
checkStationarity(Neutral_resemple)
```

```
'p-value=0.4975145669810801. Data is not stationary'
```

Rolling average for making data stationary

```
Stationary_neutral=Stationary_negative = Neutral_resemple.rolling(window=15).mean().dropna()
Stationary_neutral
```

```
reviewTime
2008-03-31      620.674667
2008-04-30      516.410000
2008-05-31      603.339333
2008-06-30      618.016667
2008-07-31      652.403333
                   ...
2018-05-31    20421.972000
2018-06-30    19102.247333
2018-07-31    17814.539333
2018-08-31    16640.882000
2018-09-30    15475.219333
Freq: M, Name: price, Length: 127, dtype: float64
```

```
checkStationarity(Stationary_neutral.dropna())
```

```
'p-value=0.010696755707564868. Data is stationary'
```

Here we can see that our Data is not stationary i.e. data don't have constant mean and constant variance over the time so for further processing we have made the data stationary using rolling mean method so we can accurately do the statistical analysis.

5.5 Time Series Modeling

5.5.1 ARMA

The name ARMA is short for Autoregressive Moving Average. It comes from merging two simpler models - the Autoregressive, or AR, and the Moving Average, or MA. In analysis, we tend to put the residuals at the end of the model equation, so that’s why the “MA” part comes second. Of course, this will become apparent once we examine the equation.

Positive trend:

Training of model :

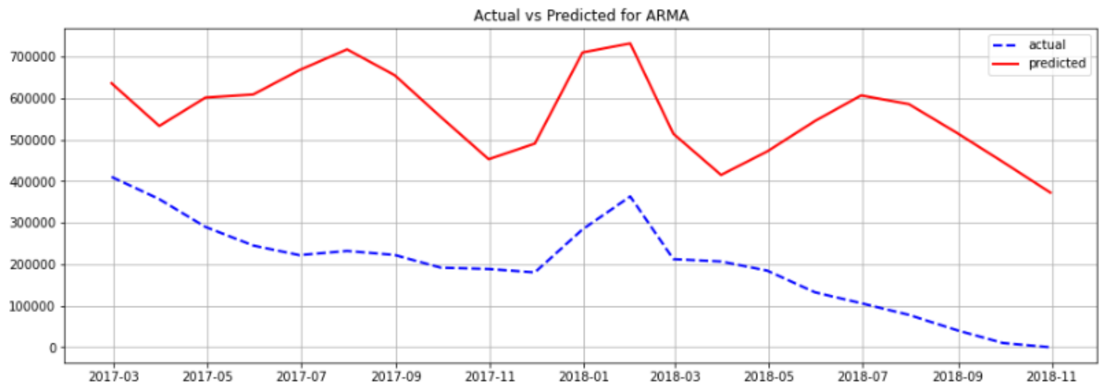
```
p,d,q = 8,0,7
arma_model1 = ARIMA(train,order=(p,d,q)).fit()

print(arma_model1.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                  price   No. Observations:                  181
Model:                 ARIMA(8, 0, 7)   Log Likelihood               -2217.429
Date:                Fri, 20 Jan 2023   AIC                           4468.857
Time:                        10:18:20   BIC                           4523.232
Sample:                    01-31-2002   HQIC                          4490.902
                         - 01-31-2017
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const       1.043e+05    1.4e-06   7.46e+10      0.000    1.04e+05    1.04e+05
ar.L1          0.0103      0.163      0.063      0.950      -0.310       0.330
ar.L2         -0.9789      0.179     -5.463      0.000      -1.330      -0.628
ar.L3         -0.0533      0.228     -0.234      0.815      -0.500       0.393
ar.L4         -0.0735      0.258     -0.284      0.776      -0.580       0.433
ar.L5          0.1529      0.297      0.515      0.606      -0.429       0.735
ar.L6          0.7714      0.299      2.582      0.010       0.186       1.357
ar.L7          0.1437      0.218      0.659      0.510      -0.284       0.571
ar.L8          0.8186      0.169      4.849      0.000       0.488       1.149
ma.L1          1.2048      3.149      0.383      0.702      -4.968       7.377
ma.L2          2.0887      0.304      6.872      0.000       1.493       2.684
ma.L3          2.4515      2.242      1.093      0.274      -1.944       6.846
ma.L4          2.4352      2.545      0.957      0.339      -2.552       7.423
ma.L5          2.0523      0.316      6.502      0.000       1.434       2.671
ma.L6          1.0979      2.902      0.378      0.705      -4.590       6.785
ma.L7          0.9121      0.113      8.050      0.000       0.690       1.134
sigma2       3.64e+09   4.69e-10   7.76e+18      0.000    3.64e+09    3.64e+09
===================================================================================
Ljung-Box (L1) (Q):                   0.79   Jarque-Bera (JB):              1153.79
Prob(Q):                              0.38   Prob(JB):                         0.00
Heteroskedasticity (H):             744.80   Skew:                             1.29
Prob(H) (two-sided):                  0.00   Kurtosis:                        15.10
===================================================================================
```

Evaluation:

Actual vs predicted :

```
plt.figure(figsize=(15,5))
plt.plot(test,'b--',linewidth=2,label='actual')
plt.plot(pred,'r-',linewidth=2,label='predicted')
plt.title(f'Actual vs Predicted for ARMA')
plt.legend()
plt.grid()
```



Key to read the graph:
red corresponds to the model values and blue corresponds to the actual \

--- Model is predicting near to the actual values but not the exact values and predicting values following the pattern of actual values

Evaluation :

```
mse = mean_squared_error(pred,test)
rmse = np.sqrt(mse)
print('\nRoot Mean squarred error for ARMA model for positive sentiments  is : ',round(rmse,2))
print('\n')
```

```
Root Mean squarred error for ARMA model for positive sentiments  is :  377659.07
```

Ljung box test :

The test examines autocorrelations of the residuals to check the goodness of the model. If the autocorrelations are very small, it is a good model

```
pvalue = sm.stats.acorr_ljungbox(model.resid,lags=[1],return_df=True)['lb_pvalue'].values
if pvalue < 0.05:
    print("Reject H0. Bad model")
else:
    print("Fail-to-Reject H0. Good model")
```

```
Fail-to-Reject H0. Good model
```

Negative trend:

Traing the model :

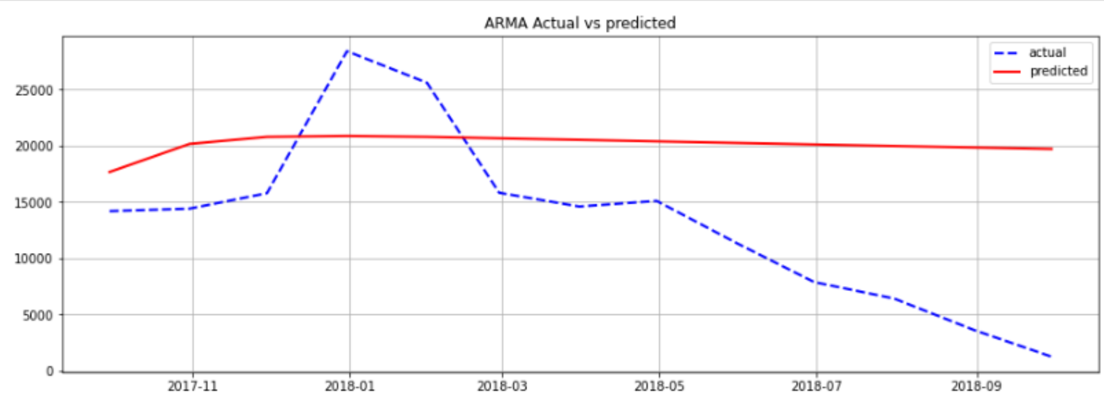
```
p,d,q = 2,0,2
arma_model2 = ARIMA(train,order=(p,d,q)).fit()
print(model.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                  price   No. Observations:                  116
Model:                 ARIMA(2, 0, 2)   Log Likelihood               -1156.950
Date:                Fri, 20 Jan 2023   AIC                           2325.899
Time:                        11:50:00   BIC                           2342.421
Sample:                    01-31-2008   HQIC                          2332.606
                         - 08-31-2017
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const        1.15e+04   1.49e+04      0.771      0.441   -1.77e+04    4.07e+04
ar.L1          1.2793      0.319      4.015      0.000       0.655       1.904
ar.L2         -0.2903      0.311     -0.933      0.351      -0.900       0.320
ma.L1         -0.2169      0.295     -0.736      0.462      -0.795       0.361
ma.L2         -0.5127      0.093     -5.505      0.000      -0.695      -0.330
sigma2      2.884e+07     32.424    8.9e+05      0.000    2.88e+07    2.88e+07
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):              2066.10
Prob(Q):                              0.99   Prob(JB):                         0.00
Heteroskedasticity (H):              47.78   Skew:                             3.98
Prob(H) (two-sided):                  0.00   Kurtosis:                        22.08
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
[2] Covariance matrix is singular or near-singular, with condition number 4.9e+20. Standard errors may be unstable.
```

Evaluation:

Actual vs predicted :

```
plt.figure(figsize=(15,5))


plt.plot(test,'b--',linewidth=2,label='actual')
plt.plot(pred,'r-',linewidth=2,label='predicted')

plt.title(f'ARMA Actual vs predicted')
plt.legend()
plt.grid()
```




```
mse = mean_squared_error(pred,test)
rmse = np.sqrt(mse)
print('\nRoot Mean squarred error for ARMA model for negative sentiments  is : ',round(rmse,2))
print('\n')
```

```
Root Mean squarred error for ARMA model for positive sentiments  is :  9829.12
```

Evaluation :

```
mse = mean_squared_error(pred,test)
rmse = np.sqrt(mse)
print('\nRoot Mean squarred error for ARMA model for negative sentiments  is : ',round(rmse,2))
print('\n')
```

```
Root Mean squarred error for ARMA model for positive sentiments  is :  9829.12
```

Ljung box test :

```
pvalue = sm.stats.acorr_ljungbox(arma_model2.resid,lags=[1],return_df=True)['lb_pvalue'].values
if pvalue < 0.05:
    print("Reject H0. Bad model")
else:
    print("Fail-to-Reject H0. Good model")
```

```
Fail-to-Reject H0. Good model
```

Neutral trend:

```
p=7;q=6
model = ARIMA(train,order=(p,0,q))
arma_model3 = model.fit()
result.summary()
```

SARIMAX Results

Dep. Variable:	price	**No. Observations:**	112
Model:	ARIMA(7, 0, 6)	**Log Likelihood**	-1098.282
Date:	Fri, 20 Jan 2023	**AIC**	2226.565
Time:	12:48:47	**BIC**	2267.342
Sample:	01-31-2007	**HQIC**	2243.109
	- 04-30-2016		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
const	9820.5635	1.46e+04	0.671	0.502	-1.89e+04	3.85e+04
ar.L1	0.0653	0.810	0.081	0.936	-1.522	1.652
ar.L2	0.0050	0.680	0.007	0.994	-1.329	1.339
ar.L3	-0.4127	0.665	-0.620	0.535	-1.717	0.891
ar.L4	0.4067	0.376	1.082	0.279	-0.330	1.143
ar.L5	-0.0795	0.173	-0.458	0.647	-0.419	0.260
ar.L6	0.4015	0.196	2.048	0.041	0.017	0.786
ar.L7	0.2936	0.355	0.828	0.408	-0.402	0.989

ma.L1	0.9565	1.087	0.880	0.379	-1.175	3.087
ma.L2	0.7754	0.850	0.912	0.362	-0.891	2.442
ma.L3	1.0990	0.620	1.774	0.076	-0.115	2.313
ma.L4	0.6664	0.775	0.860	0.390	-0.853	2.186
ma.L5	1.1355	0.346	3.277	0.001	0.456	1.815
ma.L6	0.7551	0.628	1.203	0.229	-0.475	1.985
sigma2	1.767e+07	3.614	4.89e+06	0.000	1.77e+07	1.77e+07

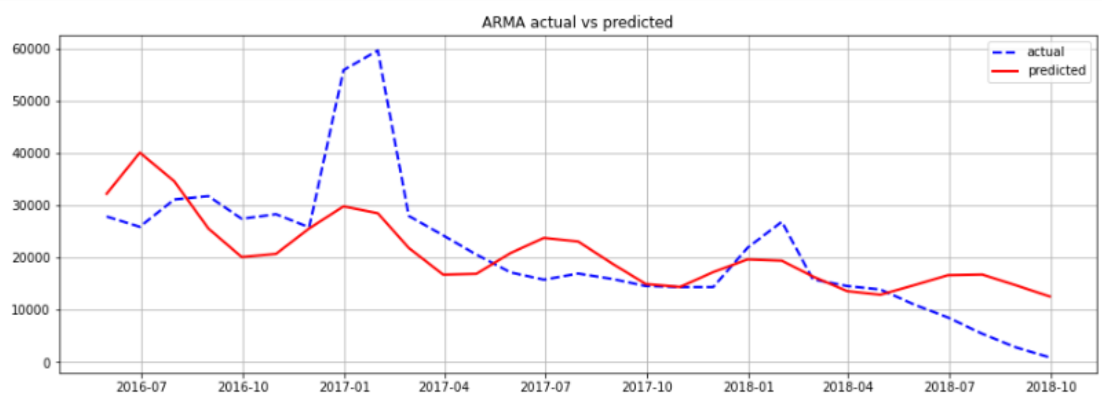
Ljung-Box (L1) (Q):	0.00	**Jarque-Bera (JB):**	223.12
Prob(Q):	0.97	**Prob(JB):**	0.00
Heteroskedasticity (H):	67.99	**Skew:**	1.43
Prob(H) (two-sided):	0.00	**Kurtosis:**	9.30

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
[2] Covariance matrix is singular or near-singular, with condition number 3.41e+22. Standard errors may be unstable.

Evaluation:

Actual vs predicted :

```
plt.figure(figsize=(15,5))


plt.plot(test,'b--',linewidth=2,label='actual')
plt.plot(pred,'r-',linewidth=2,label='predicted')

plt.title(f'ARMA actual vs predicted')
plt.legend()
plt.grid()
```



Evaluation :

```
mse = mean_squared_error(pred,test)
rmse = np.sqrt(mse)
print('\nRoot Mean squarred error for ARMA model for Neutral sentiments  is : ',round(rmse,2))
print('\n')
```

```
Root Mean squarred error for ARMA model for Neutral sentiments  is :  9856.0
```

Ljung box :

```
pvalue = sm.stats.acorr_ljungbox(arma_model3.resid,lags=[1],return_df=True)['lb_pvalue'].values
if pvalue < 0.05:
    print("Reject H0. Bad model")
else:
    print("Fail-to-Reject H0. Good model")
```

```
Fail-to-Reject H0. Good model
```

5.5.2 ARIMA

It is the combination of AR or Autoregressive I or integrations that is nothing but the lag used to make the data stationary and MA or Moving Average.

Positive Trend

```
p,d,q = 9,8,9

arima_model1 = ARIMA(train,order=(p,d,q)).fit()

print(arima_model1.summary())
```

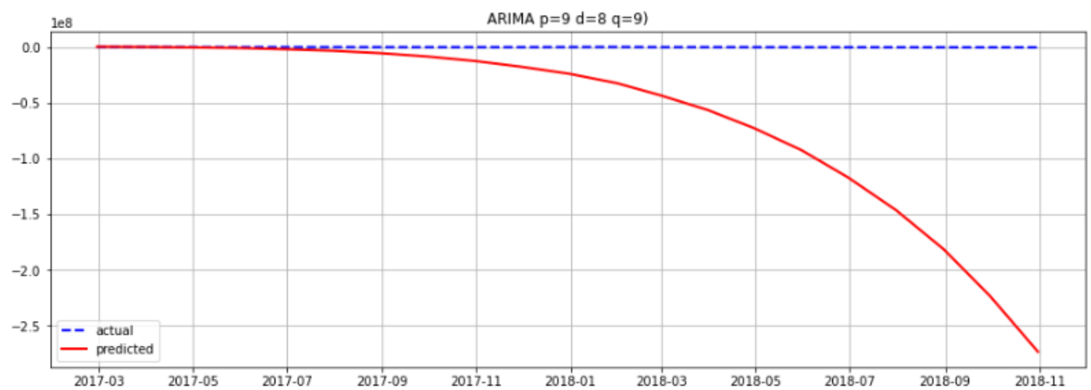
```
                               SARIMAX Results
==============================================================================
Dep. Variable:                  price   No. Observations:                  181
Model:                 ARIMA(9, 8, 9)   Log Likelihood               -2250.973
Date:                Fri, 20 Jan 2023   AIC                           4539.947
Time:                        10:28:05   BIC                           4599.859
Sample:                    01-31-2002   HQIC                          4564.253
                         - 01-31-2017
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -2.7947      0.182    -15.338      0.000      -3.152      -2.438
ar.L2         -4.8153      0.458    -10.504      0.000      -5.714      -3.917
ar.L3         -6.6020      0.746     -8.848      0.000      -8.064      -5.140
ar.L4         -7.5747      0.946     -8.005      0.000      -9.429      -5.720
ar.L5         -7.5370      0.957     -7.877      0.000      -9.412      -5.662
ar.L6         -6.4964      0.782     -8.309      0.000      -8.029      -4.964
ar.L7         -4.7088      0.530     -8.877      0.000      -5.749      -3.669
ar.L8         -2.7258      0.302     -9.031      0.000      -3.317      -2.134
ar.L9         -0.9695      0.144     -6.724      0.000      -1.252      -0.687
```

```
ma.L1        -2.8283      0.476     -5.938      0.000      -3.762      -1.895
ma.L2         2.5508      1.579      1.615      0.106      -0.544       5.646
ma.L3        -0.3351      2.323     -0.144      0.885      -4.889       4.218
ma.L4        -1.0318      2.372     -0.435      0.664      -5.681       3.618
ma.L5         1.3363      2.329      0.574      0.566      -3.229       5.902
ma.L6        -1.3610      2.095     -0.650      0.516      -5.467       2.746
ma.L7         1.4292      2.035      0.702      0.482      -2.559       5.418
ma.L8        -1.1466      1.516     -0.756      0.449      -4.117       1.824
ma.L9         0.3893      0.524      0.742      0.458      -0.639       1.417
sigma2     2.239e+10   3.38e-09   6.63e+18      0.000    2.24e+10    2.24e+10
===================================================================================
Ljung-Box (L1) (Q):                   0.59   Jarque-Bera (JB):               309.14
Prob(Q):                              0.44   Prob(JB):                         0.00
Heteroskedasticity (H):           10306.44   Skew:                            -0.95
Prob(H) (two-sided):                  0.00   Kurtosis:                         9.27
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
[2] Covariance matrix is singular or near-singular, with condition number 1.13e+35. Standard errors may be unstable.
```

Evaluation

Predicted vs actual :

```python
plt.figure(figsize=(15,5))

plt.plot(test,'b--',linewidth=2,label='actual')
plt.plot(pred,'r-',linewidth=2,label='predicted')
plt.title(f'ARIMA p={p} d={d} q={q})')
plt.legend()
plt.grid()
```



Key to read the graph:
red corresponds to the model values and blue corresponds to the actual \

Here we can see as per the time increasing error in predictions is also increasing that means model have `heteroscedasticity` factor .

Evalustion :

```
mse = mean_squared_error(pred,test)
rmse = np.sqrt(mse)
print('\nRoot Mean squarred error for ARMA model for positive sentiments  is : ',round(rmse,2))
print('\n')
```

```
Root Mean squarred error for ARMA model for positive sentiments  is :  100875050.22
```

Ljung box test :

```
pvalue = sm.stats.acorr_ljungbox(arima_model1.resid,lags=[1],return_df=True)['lb_pvalue'].values
if pvalue < 0.05:
    print("Reject H0. Bad model")
else:
    print("Fail-to-Reject H0. Good model")
```

```
Fail-to-Reject H0. Good model
```

Negative Trend

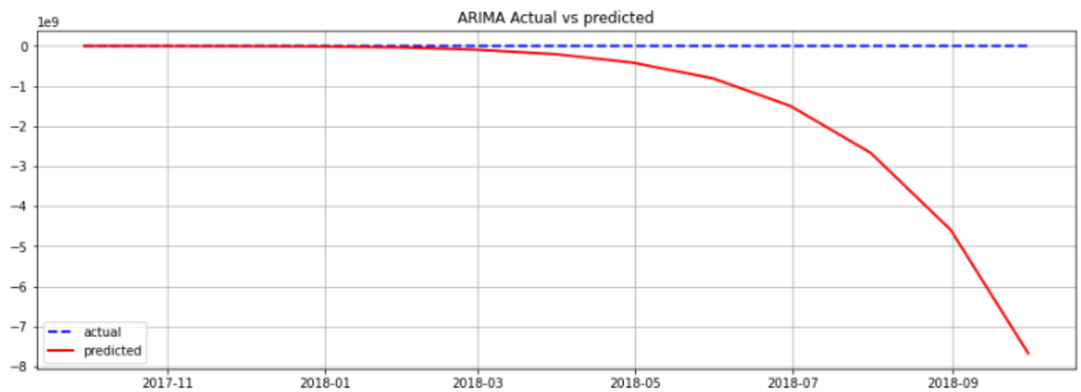
```
p,d,q = 2,12,4

arima_model2 = ARIMA(train,order = (p,d,q)).fit()
print(model.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                  price   No. Observations:                  116
Model:                 ARIMA(2, 0, 2)   Log Likelihood               -1156.950
Date:                Fri, 20 Jan 2023   AIC                           2325.899
Time:                        12:22:56   BIC                           2342.421
Sample:                    01-31-2008   HQIC                          2332.606
                         - 08-31-2017
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const        1.15e+04   1.49e+04      0.771      0.441   -1.77e+04    4.07e+04
ar.L1          1.2793      0.319      4.015      0.000       0.655       1.904
ar.L2         -0.2903      0.311     -0.933      0.351      -0.900       0.320
ma.L1         -0.2169      0.295     -0.736      0.462      -0.795       0.361
ma.L2         -0.5127      0.093     -5.505      0.000      -0.695      -0.330
sigma2      2.884e+07     32.424    8.9e+05      0.000    2.88e+07    2.88e+07
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):              2066.10
Prob(Q):                              0.99   Prob(JB):                         0.00
Heteroskedasticity (H):              47.78   Skew:                             3.98
Prob(H) (two-sided):                  0.00   Kurtosis:                        22.08
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
[2] Covariance matrix is singular or near-singular, with condition number 4.9e+20. Standard errors may be unstable.
```

Evaluation

Actual vs predicted :

```
plt.figure(figsize=(15,5))


plt.plot(test,'b--',linewidth=2,label='actual')
plt.plot(pred,'r-',linewidth=2,label='predicted')

plt.title(f'ARIMA Actual vs predicted')
plt.legend()
plt.grid()
```



Here we can see as per the time increasing error in predictions is also increasing that means model have `heteroscedasticity` factor .

Evaluation :

```
mse = mean_squared_error(pred,test)
rmse = np.sqrt(mse)
print('\nRoot Mean squarred error for ARMA model for negative sentiments  is : ',round(rmse,2))
print('\n')
```

```
Root Mean squarred error for ARMA model for positive sentiments  is :  2631881347.7
```

Ljung box test :

```
pvalue = sm.stats.acorr_ljungbox(arima_model2.resid,lags=[1],return_df=True)['lb_pvalue'].values
if pvalue < 0.05:
    print("Reject H0. Bad model")
else:
    print("Fail-to-Reject H0. Good model")
```

```
Reject H0. Bad model
```

Neutral Trend

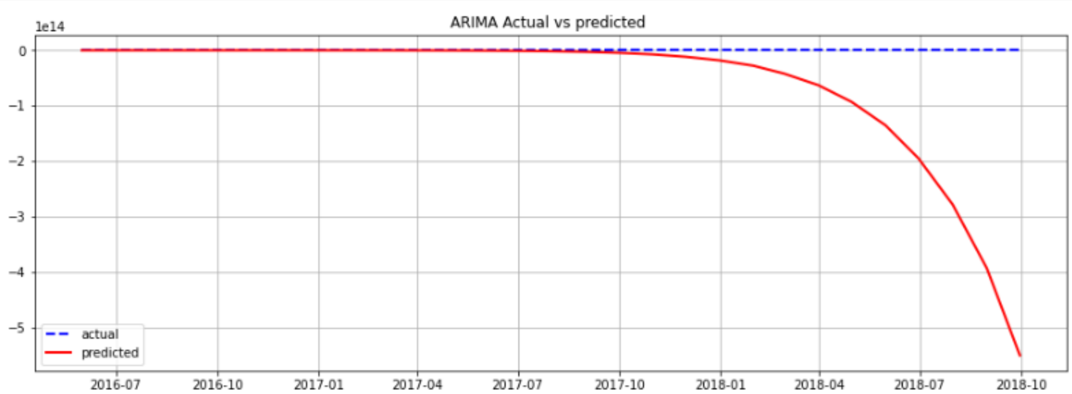
```python
p,d,q = 2,15,3

arima_model3 = ARIMA(train,order=(p,d,q)).fit()
print(arima_model3.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                  price   No. Observations:                  112
Model:                ARIMA(2, 15, 3)   Log Likelihood               -1488.026
Date:                Fri, 20 Jan 2023   AIC                           2988.052
Time:                        12:55:12   BIC                           3003.500
Sample:                    01-31-2007   HQIC                          2994.299
                         - 04-30-2016
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -1.6463      0.104    -15.773      0.000      -1.851      -1.442
ar.L2         -0.8526      0.107     -7.974      0.000      -1.062      -0.643
ma.L1         -2.8419      0.332     -8.548      0.000      -3.493      -2.190
ma.L2          2.7642      0.589      4.695      0.000       1.610       3.918
ma.L3         -0.9202      0.439     -2.096      0.036      -1.781      -0.060
sigma2        2.6e+12   4.55e-13   5.72e+24      0.000     2.6e+12     2.6e+12
===================================================================================
Ljung-Box (L1) (Q):                  36.30   Jarque-Bera (JB):                93.25
Prob(Q):                              0.00   Prob(JB):                         0.00
Heteroskedasticity (H):             113.22   Skew:                            -0.59
Prob(H) (two-sided):                  0.00   Kurtosis:                         7.66
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
[2] Covariance matrix is singular or near-singular, with condition number 7.68e+40. Standard errors may be unstable.
```

Evaluation

Actual vs Predicted :

```python
plt.figure(figsize=(15,5))



plt.plot(test,'b--',linewidth=2,label='actual')
plt.plot(pred,'r-',linewidth=2,label='predicted')


plt.title(f'ARIMA Actual vs predicted ')
plt.legend()
plt.grid()
```



Evaluation :

```
mse = mean_squared_error(pred,test)
rmse = np.sqrt(mse)
print('\nRoot Mean squarred error for ARIMA model for Neutral sentiments  is : ',round(rmse,2))
print('\n')
```

```
Root Mean squarred error for ARIMA model for Neutral sentiments  is :  144736989984770.2
```

Ljung box test :

```
pvalue = sm.stats.acorr_ljungbox(arima_model3.resid,lags=[1],return_df=True)['lb_pvalue'].values
if pvalue < 0.05:
    print("Reject H0. Bad model")
else:
    print("Fail-to-Reject H0. Good model")
```

```
Reject H0. Bad model
```

5.5.2 SARIMAX

SARIMAX (Seasonal Auto-Regressive Integrated Moving Average with eXogenous factors) is an updated version of the ARIMA model. we can say SARIMAX is a seasonal equivalent model like SARIMA and Auto ARIMA. It can also deal with external effects

Positive Trend:

SARIMA model for positive sentiments :

```
p,d,q = (8, 0, 7)

sarima_model1 = SARIMAX(train, order=(p,d,q), seasonal_order=(p,d,q,24)).fit()
print(sarima_model1.summary())
```

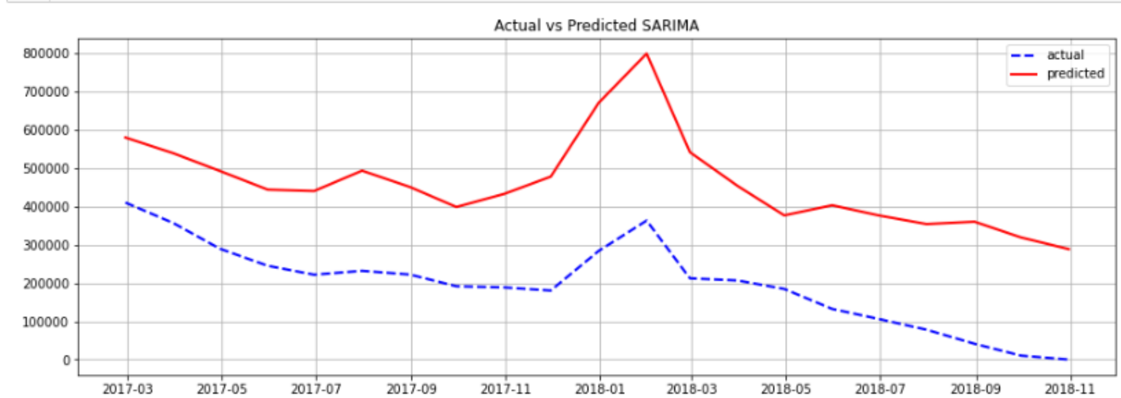
```
                                     SARIMAX Results                                      
==========================================================================================
Dep. Variable:                                 price   No. Observations:                  181
Model:             SARIMAX(8, 0, 7)x(8, 0, 7, 24)   Log Likelihood               -2195.490
Date:                            Fri, 20 Jan 2023   AIC                           4452.981
Time:                                    11:20:54   BIC                           4552.134
Sample:                                01-31-2002   HQIC                          4493.180
                                     - 01-31-2017                                         
Covariance Type:                                opg                                         
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.7851      0.856      0.917      0.359      -0.893       2.464
ar.L2         -0.5440      0.859     -0.633      0.526      -2.227       1.139
ar.L3          0.1538      0.936      0.164      0.870      -1.681       1.989
ar.L4         -0.2901      1.410     -0.206      0.837      -3.053       2.473
ar.L5          0.6326      1.409      0.449      0.653      -2.129       3.394
ar.L6         -0.2319      1.127     -0.206      0.837      -2.440       1.977
ar.L7          0.0992      0.817      0.121      0.903      -1.502       1.701
ar.L8          0.1245      0.634      0.196      0.844      -1.118       1.367
ma.L1          0.3157      0.635      0.497      0.619      -0.928       1.560
ma.L2          0.5427      0.480      1.131      0.258      -0.398       1.483
ma.L3          0.8463      0.775      1.092      0.275      -0.673       2.365
ma.L4          0.9300      0.763      1.218      0.223      -0.566       2.426
ma.L5          0.3649      0.867      0.421      0.674      -1.335       2.065
ma.L6          0.4380      0.673      0.651      0.515      -0.881       1.757
ma.L7          0.7519      0.672      1.119      0.263      -0.565       2.069
```

```
ar.S.L24        0.4389   6.21e+04   7.06e-06      1.000   -1.22e+05    1.22e+05
ar.S.L48       -0.0153   5.44e+04  -2.82e-07      1.000   -1.07e+05    1.07e+05
ar.S.L72       -0.0195   2.45e+04  -7.98e-07      1.000    -4.8e+04     4.8e+04
ar.S.L96        0.0109   3.73e+04   2.92e-07      1.000   -7.31e+04    7.31e+04
ar.S.L120    9.098e-05   3.31e+04   2.75e-09      1.000   -6.49e+04    6.49e+04
ar.S.L144       0.0011    3.3e+04   3.37e-08      1.000   -6.47e+04    6.47e+04
ar.S.L168      -0.0019   2.37e+04  -7.98e-08      1.000   -4.64e+04    4.64e+04
ar.S.L192      -0.0002   6201.389   -2.8e-08      1.000   -1.22e+04    1.22e+04
ma.S.L24        0.5462   6.21e+04   8.79e-06      1.000   -1.22e+05    1.22e+05
ma.S.L48        0.1925    5.2e+04   3.71e-06      1.000   -1.02e+05    1.02e+05
ma.S.L72        0.0753   3.81e+04   1.98e-06      1.000   -7.46e+04    7.46e+04
ma.S.L96        0.0186   2.68e+04   6.92e-07      1.000   -5.26e+04    5.26e+04
ma.S.L120   -6.361e-05   2.14e+04  -2.98e-09      1.000   -4.18e+04    4.18e+04
ma.S.L144      -0.0019   2.95e+04  -6.37e-08      1.000   -5.79e+04    5.79e+04
ma.S.L168       0.0006   4617.968   1.34e-07      1.000   -9051.050    9051.051
sigma2       3.624e+09        nan        nan        nan         nan         nan
===================================================================================
Ljung-Box (L1) (Q):                   0.03   Jarque-Bera (JB):              1206.59
Prob(Q):                              0.85   Prob(JB):                         0.00
Heteroskedasticity (H):           15114.93   Skew:                             1.96
Prob(H) (two-sided):                  0.00   Kurtosis:                        15.03
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
[2] Covariance matrix is singular or near-singular, with condition number 7.55e+27. Standard errors may be unstable.
```

Evalution:

Actual vs Predicted :

```
plt.figure(figsize=(15,5))

plt.plot(test,'b--',linewidth=2,label='actual')
plt.plot(pred,'r-',linewidth=2,label='predicted')
plt.title(f'Actual vs Predicted SARIMA')
plt.legend()
plt.grid()
```



Key to read the graph:
red corresponds to the model values and blue corresponds to the actual \

Evaluation :

```
mse = mean_squared_error(pred,test)
rmse = np.sqrt(mse)
print('\nRoot Mean squarred error for ARMA model for positive sentiments  is : ',round(rmse,2))
print('\n')
```

```
Root Mean squarred error for ARMA model for positive sentiments  is :  271848.26
```

Ljung box test :

```
pvalue = sm.stats.acorr_ljungbox(sarima_model1.resid,lags=[1],return_df=True)['lb_pvalue'].values
if pvalue < 0.05:
    print("Reject H0. Bad model")
else:
    print("Fail-to-Reject H0. Good model")
```

```
Fail-to-Reject H0. Good model
```

-- Model is predicting near to the actual values but not the exact values and predicted values following the pattern of actual values

-- SARIMA model giving the best predictions with compare to the other models as it have low RMSE value and the predicted values are following the actual values path and closed to it

Negative Trend:

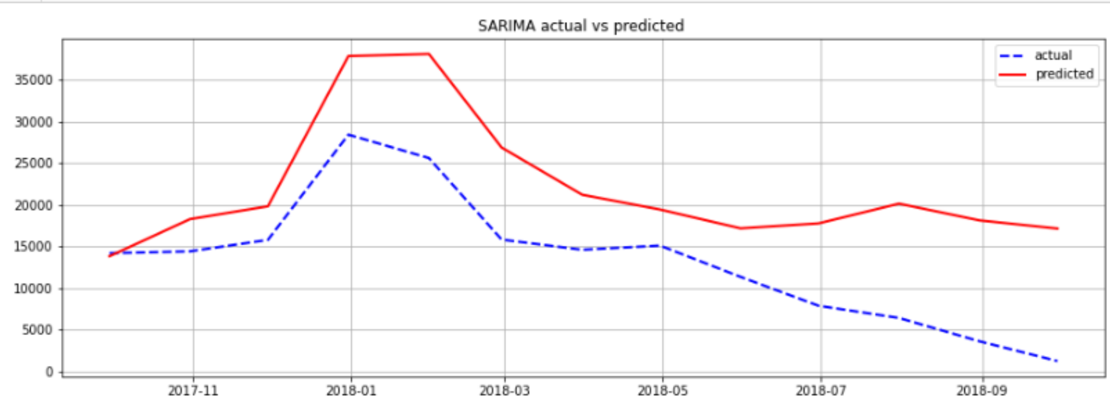
SARIMA for negative sentiments :

```
sarima_model2 = SARIMAX(train, order=(2,0,2), seasonal_order=(2,0,2,24)).fit()
pred=m.predict(len(train),((len(train)+len(test)-1)))
```

Evaluation:

Actual vs predicted :

```
plt.figure(figsize=(15,5))

plt.plot(test,'b--',linewidth=2,label='actual')
plt.plot(pred,'r-',linewidth=2,label='predicted')
plt.title(f'SARIMA actual vs predicted ')
plt.legend()
plt.grid()
```



Evaluation :

```
mse = mean_squared_error(pred,test)
rmse = np.sqrt(mse)
print('\nRoot Mean squarred error for ARMA model for negative sentiments  is : ',round(rmse,2))
print('\n')
```

```
Root Mean squarred error for ARMA model for positive sentiments  is :  9784.9
```

Ljung box test:

```
pvalue = sm.stats.acorr_ljungbox(sarima_model2.resid,lags=[1],return_df=True)['lb_pvalue'].values
if pvalue < 0.05:
    print("Reject H0. Bad model")
else:
    print("Fail-to-Reject H0. Good model")
```

```
Fail-to-Reject H0. Good model
```

------- SARIMA model giving the best predictions with compare to the other models as it have low RMSE value and the predicted values are following the actual values path and closed to it

Neutral Trend:

SARIMA model for neutral sentiments :

```
p,d,q = (7, 0, 9)

sarima_model3 = SARIMAX(train,order=(p,d,q),seasonal_order=(p,d,q,24)).fit()
print(sarima_model3.summary())
```

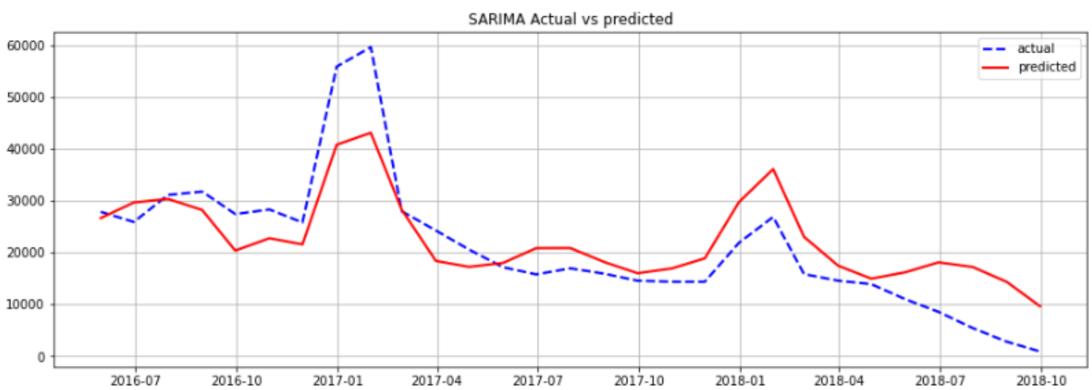
```
                                     SARIMAX Results                                      
==========================================================================================
Dep. Variable:                                price   No. Observations:                  112
Model:             SARIMAX(7, 0, 9)x(7, 0, 9, 24)   Log Likelihood               -1087.778
Date:                            Fri, 20 Jan 2023   AIC                           2241.556
Time:                                    13:40:25   BIC                           2331.267
Sample:                                01-31-2007   HQIC                          2277.955
                                     - 04-30-2016                                         
Covariance Type:                              opg                                         
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.4614      1.590      0.290      0.772      -2.654       3.577
ar.L2         -0.9861      1.748     -0.564      0.573      -4.412       2.440
ar.L3          0.5766      1.591      0.363      0.717      -2.541       3.694
ar.L4         -0.5467      1.740     -0.314      0.753      -3.957       2.864
ar.L5          0.5672      2.137      0.265      0.791      -3.622       4.756
ar.L6         -0.0836      1.829     -0.046      0.964      -3.669       3.501
ar.L7          0.3216      0.734      0.438      0.661      -1.117       1.760
ma.L1          0.6298      1.439      0.438      0.662      -2.191       3.450
ma.L2          1.5855      2.135      0.743      0.458      -2.598       5.769
ma.L3          1.1002      1.658      0.663      0.507      -2.150       4.350
ma.L4          1.7951      1.656      1.084      0.278      -1.450       5.040
ma.L5          1.2989      2.806      0.463      0.643      -4.200       6.798
ma.L6          1.4403      1.452      0.992      0.321      -1.406       4.286
ma.L7          1.0471      1.439      0.728      0.467      -1.772       3.867
ma.L8          0.7435      1.308      0.568      0.570      -1.820       3.307
ma.L9          0.6400      0.955      0.670      0.503      -1.232       2.512
```

```
ar.S.L24       0.2864   6.46e+04   4.44e-06      1.000   -1.27e+05    1.27e+05
ar.S.L48       0.0389   2.71e+04   1.44e-06      1.000   -5.31e+04    5.31e+04
ar.S.L72      -0.0223   2.58e+04  -8.65e-07      1.000   -5.05e+04    5.05e+04
ar.S.L96       0.0059   1.76e+04   3.36e-07      1.000   -3.44e+04    3.44e+04
ar.S.L120      0.0002   2.58e+04   6.73e-09      1.000   -5.05e+04    5.05e+04
ar.S.L144     -0.0002   2.06e+04  -1.18e-08      1.000   -4.04e+04    4.04e+04
ar.S.L168      0.0011   1.55e+04   7.43e-08      1.000   -3.03e+04    3.03e+04
ma.S.L24       0.3399   6.46e+04   5.26e-06      1.000   -1.27e+05    1.27e+05
ma.S.L48       0.0390   3.26e+04    1.2e-06      1.000    -6.4e+04     6.4e+04
ma.S.L72       0.0289    2.6e+04   1.11e-06      1.000    -5.1e+04     5.1e+04
ma.S.L96       0.0029   2.17e+04   1.33e-07      1.000   -4.26e+04    4.26e+04
ma.S.L120     -0.0076   2.23e+04  -3.41e-07      1.000   -4.36e+04    4.36e+04
ma.S.L144     -0.0053   2.49e+04  -2.13e-07      1.000   -4.88e+04    4.88e+04
ma.S.L168     -0.0039   1.75e+04  -2.22e-07      1.000   -3.42e+04    3.42e+04
ma.S.L192     -0.0020   1.74e+04  -1.15e-07      1.000   -3.41e+04    3.41e+04
ma.S.L216     -0.0007   2.17e+04  -3.25e-08      1.000   -4.25e+04    4.25e+04
sigma2      2.633e+07     34.110   7.72e+05      0.000    2.63e+07    2.63e+07
===================================================================================
Ljung-Box (L1) (Q):                   0.25   Jarque-Bera (JB):               306.73
Prob(Q):                              0.62   Prob(JB):                         0.00
Heteroskedasticity (H):             106.63   Skew:                             1.52
Prob(H) (two-sided):                  0.00   Kurtosis:                        10.51
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
[2] Covariance matrix is singular or near-singular, with condition number 2.46e+22. Standard errors may be unstable.
```

Evaluation:

Actual vs predicted :

```python
plt.figure(figsize=(15,5))

plt.plot(test,'b--',linewidth=2,label='actual')
plt.plot(pred,'r-',linewidth=2,label='predicted')

plt.title(f'SARIMA Actual vs predicted')
plt.legend()
plt.grid()
```



Evaluation :

```
mse = mean_squared_error(pred,test)
rmse = np.sqrt(mse)
print('\nRoot Mean squarred error for SARIMA model for Neutral sentiments  is : ',round(rmse,2))
print('\n')
```

```
Root Mean squarred error for SARIMA model for Neutral sentiments  is :  7003.46
```

Ljung box test : ¶

```
pvalue = sm.stats.acorr_ljungbox(sarima_model3.resid,lags=[1],return_df=True)['lb_pvalue'].values
if pvalue < 0.05:
    print("Reject H0. Bad model")
else:
    print("Fail-to-Reject H0. Good model")
```

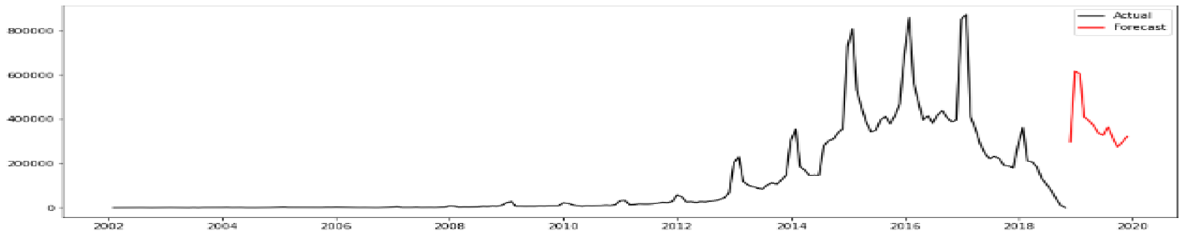
```
Fail-to-Reject H0. Good model
```

5.6 Comparison table

Models	Positive (RMSE)	Negative (RMSE)	Neutral (RMSE)
SARIMAX	271848	9584	9015
ARMA	377659	9829	9856
ARIMA	100875050	2631881348	14473698998

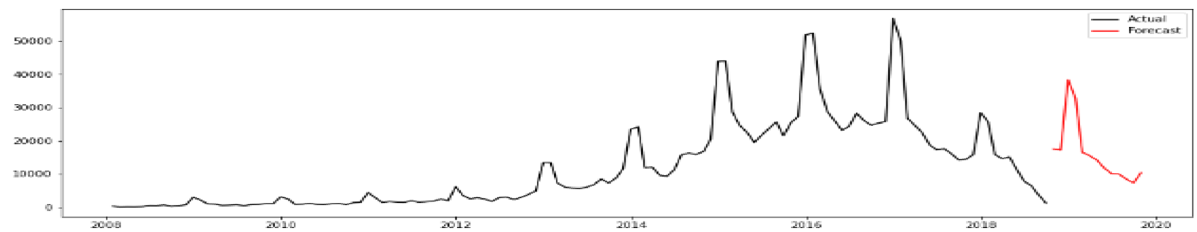
5.7 Demand Forecasting

Positive Trend



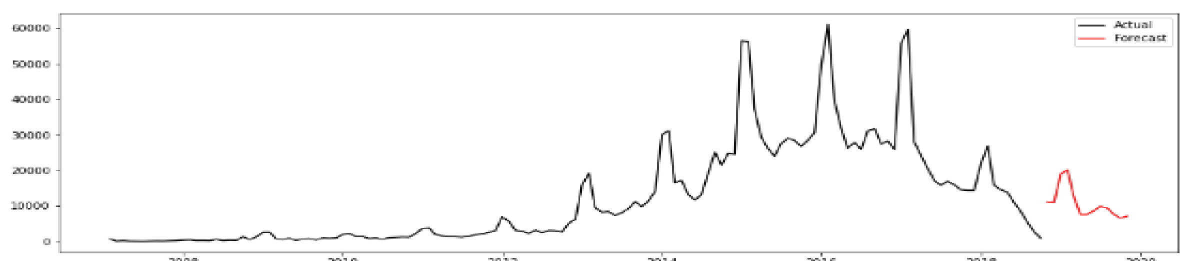
Sales for the products with positive reviews in 2019 will be highest in the end of first quarter around 6 lac and lowest in November around 3 lac.

Negative Trend



Sales for negative reviewed products will hit peak at the end of 1st quarter around 38 K and decline as the year passes and will reach lowest sales in November 2019 around 10 K and will again increase in December 2019.

Neutral Trend



Sales for neural reviewed products will be highest around 20K at the end of first quarter and then start decreasing and will again increase in 3rd quarter and then again start to decrease and lowest in November around 8K.

5.8 Suggestions:

- The total sales for the positively reviewed products will increase in 2019 in comparison to 2018 so the production should be high in 2019 but as the year passes the sales will decrease that can be the effect of seasonal sales and high prices . so we can optimize the inventory by increasing the production because the demand of the products will be high throughout the year with comparison to the previous year.

- The total sales of the negatively reviewed products will be high for the first quarter of 2019 and mostly the same after as the previous year so we can increase the production for the first quarter of 2019 rather than the first quarter of 2018 and optimize the inventory. Although these products got negative reviews their sales will still be more than the neutral reviewed products because these can include some essential products. High prices and quality of products can be the reason for these reviews so we can also consider these concerns and give some discounts and quality products.

- The total sales of the neutral reviewed products will decrease in the first three quarters of 2019 compared to 2018 but in the last quarter of 2019 there will be an increase in the sales of products so we can do the production accordingly.

- As we can observe that if the sales can increase in 2019 we can increase the production and when the sales can decrease we can give some discounts & offers , and do more advertisements related to the products.

6. Conclusions

- First cluster (Moderate performing) has good ratings but average sales. As observed the categories with less price are giving better sales and purchase frequency of those products is also high, so to improve the sales it is recommended to drop the price.

- Second cluster (Best performing) is having good ratings as well as sales. So, these categories should not be tampered.

- Third cluster (Needs Improvement) is having the worst response in terms of rating as well as sales. So, complete overhauling is required to improve their ratings.

- The total sales for the positively reviewed products will increase in 2019 in comparison to 2018 so the production should be high in 2019 but as the year passes the sales will decrease, which can be the effect of seasonal sales and high prices . so we can optimize the inventory by increasing the production because the demand of the products will be high throughout the year with comparison to the previous year .

- The total sales of the negatively reviewed products will be high for the first quarter of 2019 and mostly the same after as the previous year so we can increase the production for the first quarter of 2019 rather than the first quarter of 2018 and optimize the inventory. Although these products got negative reviews , their sales will be more than the neutral reviewed products because these can include some essential products. High prices, quality of products can be the reason for these reviews so we can also consider these concerns and give some discounts and quality products .

- The total sales of the neutral reviewed products will decrease in the first three quarters of 2019 compared to 2018 but in the last quarter of 2019 there will be an increase in the sales of products so we can do the production accordingly .

- As we can observe that if the sales can increase in 2019 we can increase the production and when the sales can decrease we can give some discounts & offers , do more advertisements related to the products.