meatball and noodle are often bought together since people are likely to make Italian noodle with those two items. Besides, interesting, beer and diapers are usually both together because they are often purchased by dads who are more likely to perform the shopping and the selection while the moms are with the toddler. There are main application of association rule mining such as basket data analysis with finding association of purchased items, cross marketing and catalog design.

When it comes to the coding implementation for SoftMax, I import necessary library then define SoftMax function in accordance to the formula and then define the input in the format of vector. I then input the vector as the input into the SoftMax function. The function will return list of probabilities distribution of vector. The sum of the probabilities would be equal to 1. I also test that in the output.

```
#Define the softmax function that takes input as a vector
def softmax(vector_input):
    e = exp(vector_input)
    return e / e.sum()
print("Define softmax function: ", softmax)
#Define the vector
input = [4,8,5,6,2,9,7,6,3]
print("\nDefine input in array format as follow:",input)
# Let's convert list of numbers to a list of probabilities using softmax function
output = softmax(input)
```

With the similar approach, this time, I define another function that could take input as a matrix and then apply that function into matrix. It will then return the output as a list of probabilities in a corresponding size of matrix. In this case, it is 3x3.

```
# Define softmax function that takes input as a matrix
def softmax2(matrix_input):
    X_exp = np.exp(matrix_input)
    partition = X_exp.sum(1, keepdims = True)
    return X_exp/ partition
print(softmax2)
# Apply function on the matrix
X_prob = softmax2(X)
```

As far as the implementation of coding for customer sentiment analysis is regarded, I utilize dataset found in Kaggle about Women's Clothing E-Commerce dataset including all the reviews written by customers. The data will be used most for this analysis is "Text" and "Rating" column. Text column will include all the review about clothing products from customers while Rating column represents the rating rated by customers. I first implement necessary library and package of nltk and stopwords to conduct sentiment analysis to generate word cloud of most popular words used in the reviews. In accordance with the output, the popular words are mainly positive ones, demonstrating the fact that most reviews in the dataset showcases positive feedback sentiment. Next step, I create an additional column attached to the dataset, naming it "sentiment". This column will be generated base on the rating score with + 1for positive reviews if rating above 3 and -1 for negative reviews if rating below 3. All reviews with Rating = 3 will be dropped since my model will only classify positive and negative reviews. In the following steps, I split the dataframe based

on its positive and negative sentiment in the sentiment column and then utilize wordcloud function again along with stopwords package of nltk to generate most frequent world cloud, excluding those that appears in both negative and positive reviews. We could notice from the output that most of the positive reviews include words love, perfect, etc while negative reviews contain negative texts regarding to size, fit, and even, etc. I then generate the distribution of reviews with sentiment across the dataset as showed below in the output section. In the last steps, I built the sentiment analysis model after cleaning the dataframe with removing punctuation. I then split the dataframe and subset only the "Text" and "sentiment" column. The new dataset will then be splitted into train set with 70% and test set with 30% of the dataset. Additionally, a bag of words will be created using count vectorizer from Scikit-learn library, which will transform the text in the dataframe into a bag of words and include a sparse matrix of integers. I then import and utilize logistic regression mode to train and test the data frame with X as Text and y as sentiment. The model will be utilized to fit on the data and predictions will be applied with the X_test. Finally, I test the accuracy of the model using confusion matrix and classification report. The final output is the classification report of the model with accuracy rate being 93% which is considered great model tin term of classifying the reviews data.

```
####### WORDCLOUD - Positive Sentiment ###########
' good and great will be removed because they were included in negative sentiment.
stopwords = set(stopwords.words('english'))
topwords.update(["br", "href","good","great","dress","top","one","like","color","look"])
os = " ".join(review for review in positive.Text)
ordcloud2 = WordCloud(stopwords=stopwords).generate(pos)
rint("\nMost frequently used words in the positive reviews:\n")
lt.imshow(wordcloud2, interpolation='bilinear')
lt.axis("off")
lt.savefig('wordcloud02.png')
lt.show()
```

```
# New dataframe will have two columns - Title of review and sentiment.
dfNew = df[['Text','sentiment']]
print("\nTake a look at the heading of new subset dataset\n")
print(dfNew.head())
#random split train and test data 70% train and 30% test
index = df.index
df['random_number'] = np.random.randn(len(index))
train = df[df['random_number'] <= 0.7]
test = df[df['random_number'] > 0.7]

# Create a bag of words using count vectorizer from Scikit-learn library.

# count vectorizer:
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(token_pattern=r'\b\w+\b')
train_matrix = vectorizer.fit_transform(train['Text'])
test_matrix = vectorizer.transform(test['Text'])

## Import Logistic Regression.
# Logistic Regression
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()

### Split target and independent variables
X_train = train_matrix
X_test = test_matrix
y_train = train['sentiment']
y_test = test['sentiment']


### Fit model on data
lr.fit(X_train,y_train)


### Make prediction
predictions = lr.predict(X_test)

# find accuracy, precision, recall
# Test the accuracy of the model
from sklearn.metrics import confusion_matrix,classification_report
new = np.asarray(y_test)
print("Confusion matrix:",confusion_matrix(predictions,y_test))
```

# CODING IMPLEMENTATION - OUTPUT

## 1.Write code in python to implement softmax function

1.a. Apply softmax function in a predefined vector

Define softmax function:  <function softmax at 0x1a3357d560>

Define input in array format as follow: [4, 8, 5, 6, 2, 9, 7, 6, 3]

As a result,applying softmax function,the propabilities of the vector is calculated as follow:
 [0.004 0.226 0.011 0.031 0.001 0.613 0.083 0.031 0.002]

Sum of the probabilities 1.0

***
1.b. Apply softmax function in a predefined matrix

Given a matrix X 3x3 we could sum over all elements in the same axis.

[[ 3.83682479 -1.99815641  3.01522871 -0.23815476  2.41484459 -1.3969099 ]
 [ 3.6698077  -2.06695995 -4.40939249  3.83926995  3.3833769   1.1537748 ]
 [-1.5559371   2.98495289  3.2456137   1.00243817 -0.00507989 -0.79398461]]

Define softmax function that take input as matrix:
<function softmax2 at 0x1a355d3710>

As a result,applying softmax function,the propabilities of the matrix is calculated as follow:

[[5.8609e-01 1.7100e-03 2.5772e-01 9.9600e-03 1.4139e-01 3.1300e-03]
 [3.3113e-01 1.0700e-03 1.0000e-04 3.9228e-01 2.4866e-01 2.6750e-02]
 [4.2300e-03 3.9693e-01 5.1514e-01 5.4670e-02 1.9960e-02 9.0700e-03]]

Sum of the probabilities [1. 1. 1.]

**********

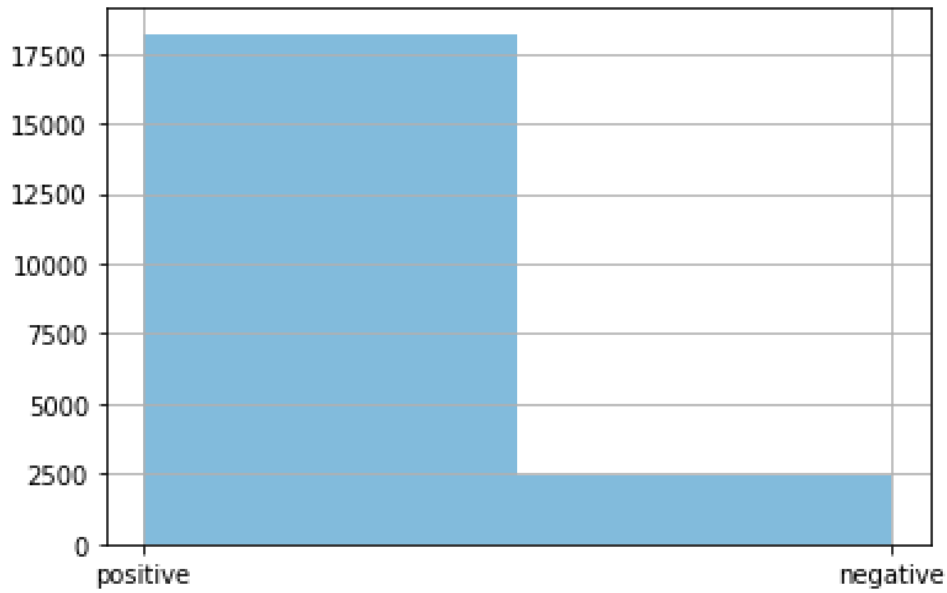
## 2.Write code in python to implement customer sentiment analytics:

**Most frequently used words in the reviews:**


**Most frequently used words in the positive reviews:**


**Most frequently used words in the negative reviews:**


**Distribution of reviews with sentiment across dataset:**




**Confusion matrix**: [[ 355  125]
[ 216 4325]]

**Finally, below is the classification report of my customer review sentiment model**

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| -1 | 0.62 | 0.74 | 0.68 | 480 |
| 1 | 0.97 | 0.95 | 0.96 | 4541 |
| accuracy | | | 0.93 | 5021 |
| macro avg | 0.80 | 0.85 | 0.82 | 5021 |
| weighted avg | 0.94 | 0.93 | 0.93 | 5021 |