




[illegible]

April 2022

Introduction	3
What is Recommendation system?	3
Data Source and Solution Approach:	4
Restaurant Exploratory Data analysis	5
Restaurants overall sales comparison	5
Feedback and Reviews Sentiment analysis	9
Sentiment Prediction Model (Classification)	11
Term Frequency-Inverse Document Frequency	13
Yelp sentiment analysis	14
Types of the Recommendation Systems	16
Food Items Network analysis	17
Recommendation based on popularity of being ordered together	18
Content Based Recommendation	19
The Most Popular Item	19
Ingredient based recommendation	19
Item Similarity Recommendation	20
Collaborative Filter Based Recommendation	25
Data limitation problem:	26
Content Based Recommendation	29

Introduction

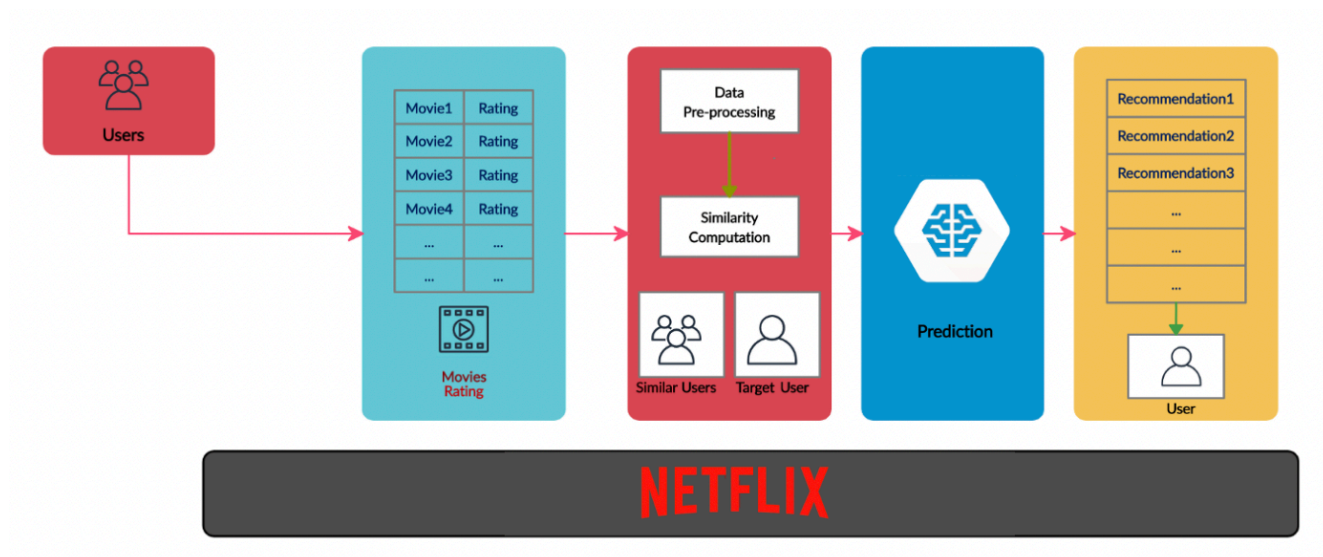
Nowadays we have many types of food and cuisine available for users to order from different online applications. With the exponential increase in the number of available food options, it is hard for the users to pick and order their desired meal especially from a new cuisine that they have never tried before. Currently, personalized recommendation systems are implemented by many online businesses. However most of them are restaurant based not food ingredient based. In this project we aim to build a personalized restaurant recommender system prototype that not only considers the users food rating and history, but also based on the food ingredient. Such an approach can increase each user's experience of exploring and finding their desired food that is closer to their tastes.

Titi Home Made 27 Km	Zeitoon Restaurant 5 Km	Gilaneh Persian Grill Hous... 4 Km
		
Koobideh Kabab With Rice \$14.85 ★★★★★ (6)	Zereshk Polo (Persian Barbe... \$3.95	Lamb Shank With Dill Rice \$13.99

What is Recommendation system?

A recommendation system generates a compiled list of items in which a user might be interested, in the reciprocity of their current selection of items, and it does not recommend items that the user already knows.

For instance, the Netflix recommendation system offers recommendations by matching and searching similar users' habits and suggesting movies that share characteristics with films that users have rated highly.



Data Source and Solution Approach:

We used datasets provided by [Fooreco](#), which is an online ordering application. The entire dataset consists of 14K orders, 5K users, 800 ratings and over 80K food items. We will begin with cleaning the dataset. Continue with some primary analysis on each dataset and then will combine them to get a combined clean data for further analysis. Then we will do deeper exploratory data analysis and some visualization. Later on, we will explore different recommendation systems like collaborative and content based and will validate and evaluate different models. After some data wrangling and data cleaning the most important features are listed below:

-
1. OrderId
 2. ItemId
 3. RestaurantId
 4. Item Description
 5. Feedback
 6. Review
 7. Rating
 8. Item Name
 9. Item Price
 10. Ordered Count
-

Restaurant Exploratory Data analysis

There are many reasons application are using recommendations. The most ones can be:

1. Help user to decide faster and easier what to order based on different approaches
2. Help restaurants to increase the sales, by recommending their customer or new customers that items they may like the most.

Therefore we start our analysis with visualizing and exploring restaurants sale through Fooreco application. Let's name the 4 top restaurants with the highest sales, A, B, C and D.

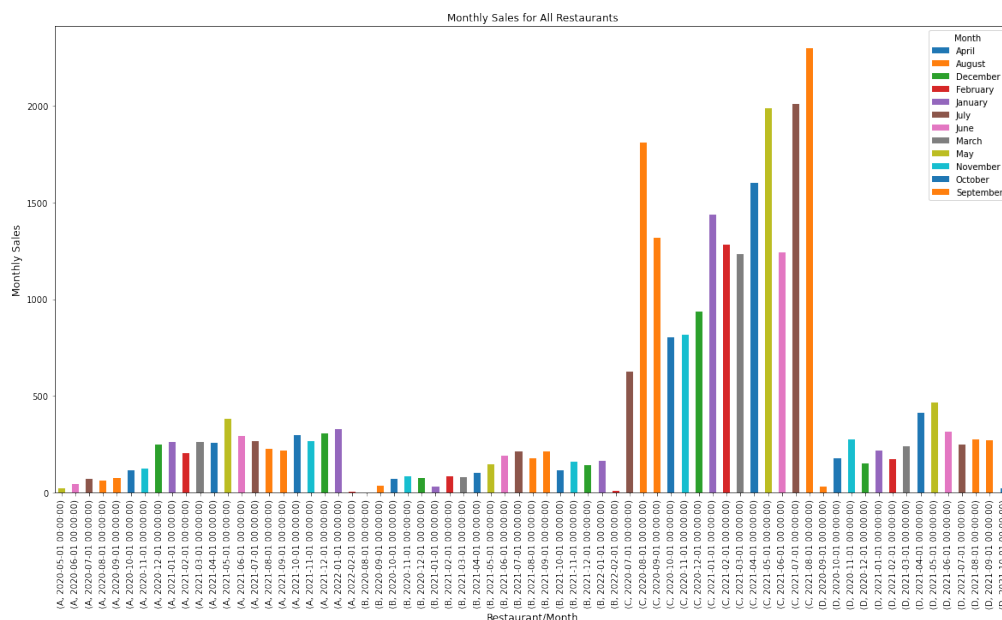
Restaurants overall sales comparison

The graph below displays the overall sales items for all the restaurants during the recent years. As seen, restaurant C has the most number of sales among the restaurants.



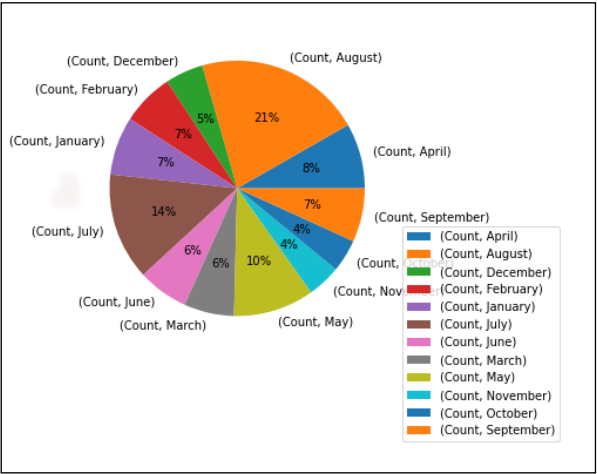
Also if we let's visualize the restaurant sales for each month during the year - just a quick reminder, due to Covid restriction this recent sales and data might be different from other restaurant historical data - As shown in the plot, for most of them May has a peak in

sales, and for restaurant C, August and July also were among the months with high number of sales.

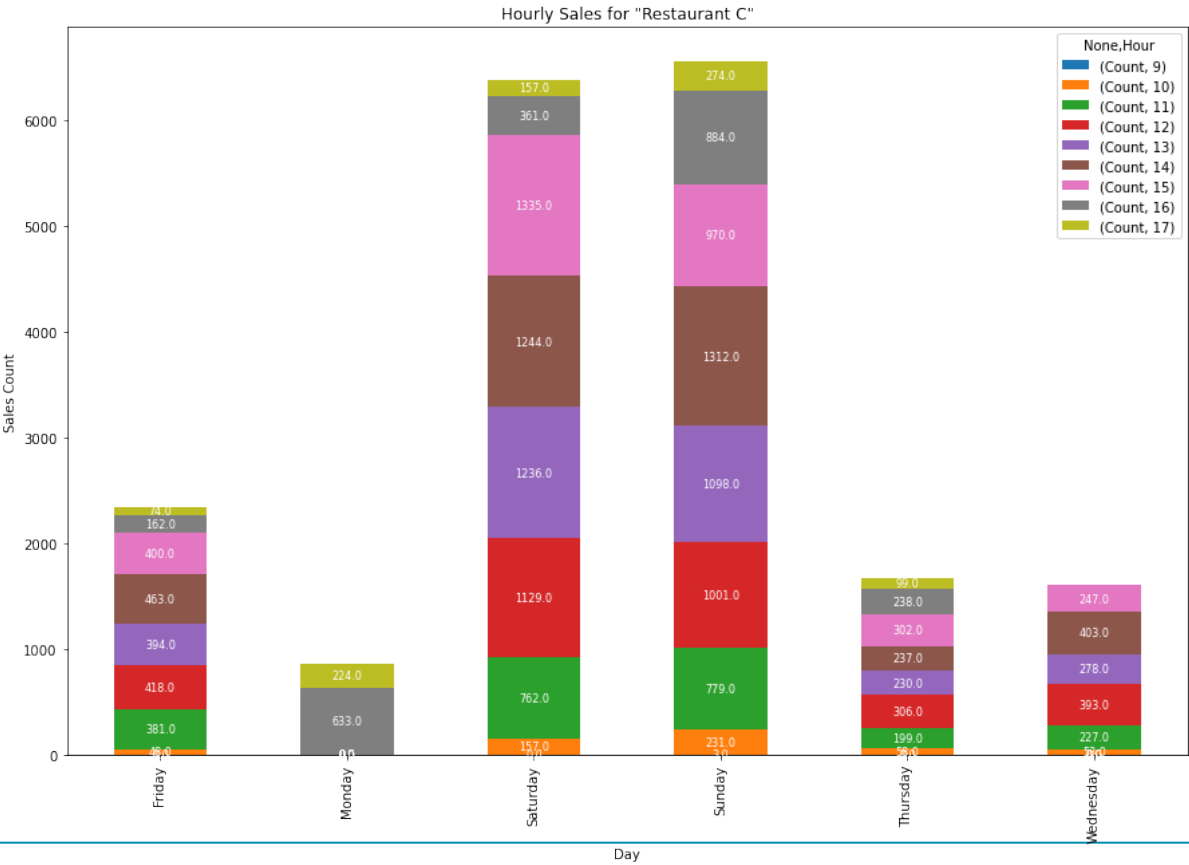


As mentioned before restaurant C has the most number of sales, therefore we will investigate a little bit deeper for this restaurant.

Plotting the monthly percentage sales approves that August and July are the top sales months.



Daily and hourly plot will show, which day of the week has the highest sales amount and also what time of the day has the sales peak.



Restaurant C operation hours is between 9 to 5pm. According to the data, Saturday and Sundays are the most busy days of the week. The sales count is pretty well distributed among the day from 11am to 5pm

In the next section of this project, we will start analyzing the food items. The first step before actually diving in to the food items, let's do some sentiment analysis on the reviews and feedback.

Feedback and Reviews Sentiment analysis

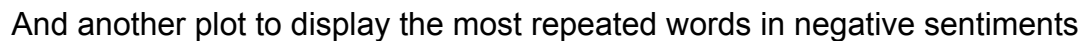
Unfortunately Fooreco dataset does not contain enough feedback and reviews for the food, as the app was launched just in few restaurants and not for so long. Therefore to complete this section we have used Yelp data at the end, and train our model with yelp data and use that model for Fooreco to predict future data.

To start the sentiment analysis, we should label our available data to be able to classify and predict new comments. The approach to do this, we labeled the comments with rating higher than 3, as positive comment and the comments combined with rating lower than 3 as a negative comment.

As expected and the distribution of rating is displayed in below histogram, the ratings are not equally distributed.



Using WordCount function that will plot the words with the size correlated to the repetition of it. Positive comments are plotted as below:



Sentiment Prediction Model (Classification)

Preprocess:

Countvectorizer makes it easy for text data to be used directly in machine learning and deep learning models such as text classification. Therefore first we convert the text to vectors using Countvectorizer.

Model:

Then we will use one of the **Naive Bayes (NB)** classifier for defining the model. Specifically, we will use **MultinomialNB classifier**. Getting help from cheat sheet given by sklearn **here** to determine the best model to use for a particular problem. It tells us to use NB classifier.

Metric:

The **accuracy_score** function is used for the accuracy, of correct predictions. Also confusion matrix is drawn to display the different values for false and correct prediction.

To complete the evaluation calculation A Classification report is a performance evaluation metric in machine learning which is used to show the precision, recall, F1 Score, and support score of your trained classification model.

Before running the model, let's divide the data to train and test and transform the data to vectors. At the end train the model on train data and predict the test data and measure the prediction score by the mentioned method.

```
y = feedback['sentiment']  
X = feedback['feedback']
```

```
X_train,X_test, y_train,y_test = train_test_split(X,y,test_size=0.33,random_state=53)  
count_vectorizer = CountVectorizer(stop_words='english')  
count_train = count_vectorizer.fit_transform(X_train.values)  
count_test = count_vectorizer.transform(X_test.values)
```

```

nb_classifier = MultinomialNB(alpha=0.5)
nb_classifier.fit(count_train,y_train)
pred = nb_classifier.predict(count_test)

score = metrics.accuracy_score(y_test,pred)
print("Naive Bayse Score: {:.2%}".format(score))

cm = metrics.confusion_matrix(y_test,pred)
print("\nConfusion Matrix:\n",cm)

print("\nClassification Report:\n",classification_report(y_test,pred,labels=np.unique(pred)))

```

```

Naive Bayse Score: 79.07%

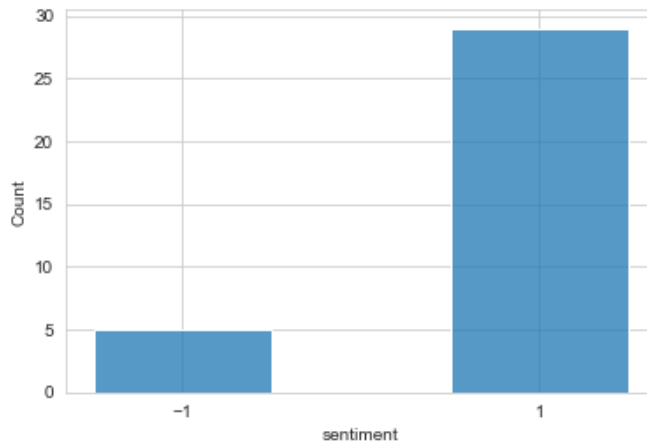
Confusion Matrix:
[[ 0  5]
 [ 4 34]]

Classification Report:

```

	precision	recall	f1-score	support
-1	0.00	0.00	0.00	5
1	0.87	0.89	0.88	38
accuracy			0.79	43
macro avg	0.44	0.45	0.44	43
weighted avg	0.77	0.79	0.78	43

The prediction score is 79% which is not bad for sentiment analysis, but as seen in the confusion matrix, and confirmed in classification report, the prediction performance for negative sentiments is awful and the reason for that is that the data is very limited and the number of negative sentiments are much lower as displayed in below histogram:



Term Frequency-Inverse Document Frequency

Let's use TF-IDF, instead of CountVector model, which takes in account the product of term frequency and inverse document frequency.

```
tfidf_vectorizer = TfidfVectorizer(stop_words="english",max_df=0.7)
tfidf_train = tfidf_vectorizer.fit_transform(X_train)
tfidf_test = tfidf_vectorizer.transform(X_test)

# fit the model on data and predict
nb_classifier.fit(tfidf_train,y_train)
pred = nb_classifier.predict(tfidf_test)

# metrics and validation of the model
score = metrics.accuracy_score(y_test,pred)
print("Naive Bayse Score: {:.2%}".format(score))

cm = metrics.confusion_matrix(y_test,pred,labels=[-1,1])
print("\nConfusion Matrix:\n",cm)

print("\nClassification Report:\n",classification_report(y_test,pred,labels=np.unique(pred)))
```

```
Naive Bayse Score: 88.37%

Confusion Matrix:
[[ 0  5]
 [ 0 38]]

Classification Report:
              precision    recall  f1-score   support

         1         0.88      1.00      0.94         38

   micro avg         0.88      1.00      0.94         38
   macro avg         0.88      1.00      0.94         38
weighted avg         0.88      1.00      0.94         38
```

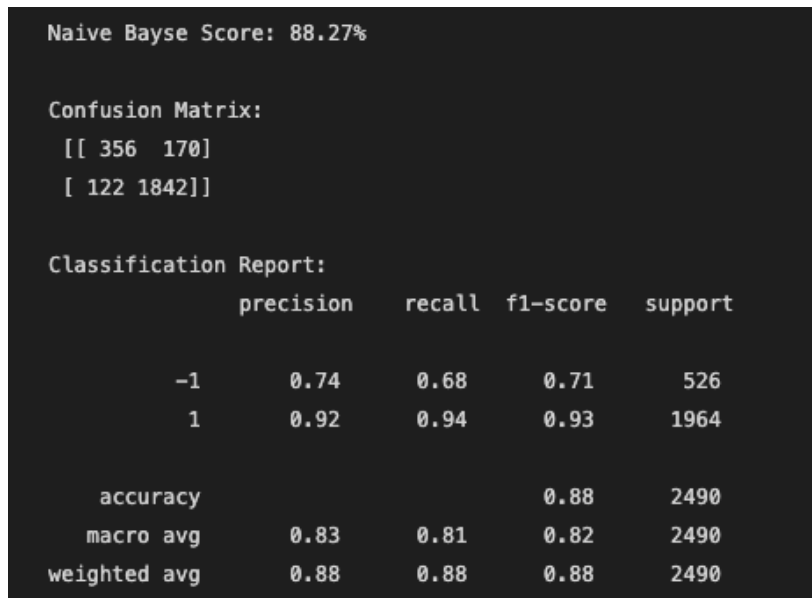
As seen in the performance report, the score is higher 88.37% for correct predictions. But again for negative, it did not predict even one correct.

Yelp sentiment analysis

As seen before, because the data is very limited for rating and comments, we want to create our model based on Yelp data.

We have done same process as before for Yelp data, and here we just want to show the final result and the performance evaluation.

As shown in the below image, the score is 88.27% for correct prediction which is really good. And by looking at the confusion matrix and the report we see a much better score for negative sentiments as well. Also its worth to mention the prediction for negative sentiment is lower as the data with lower rating is much less that the data with higher rating as seen in the following histogram.



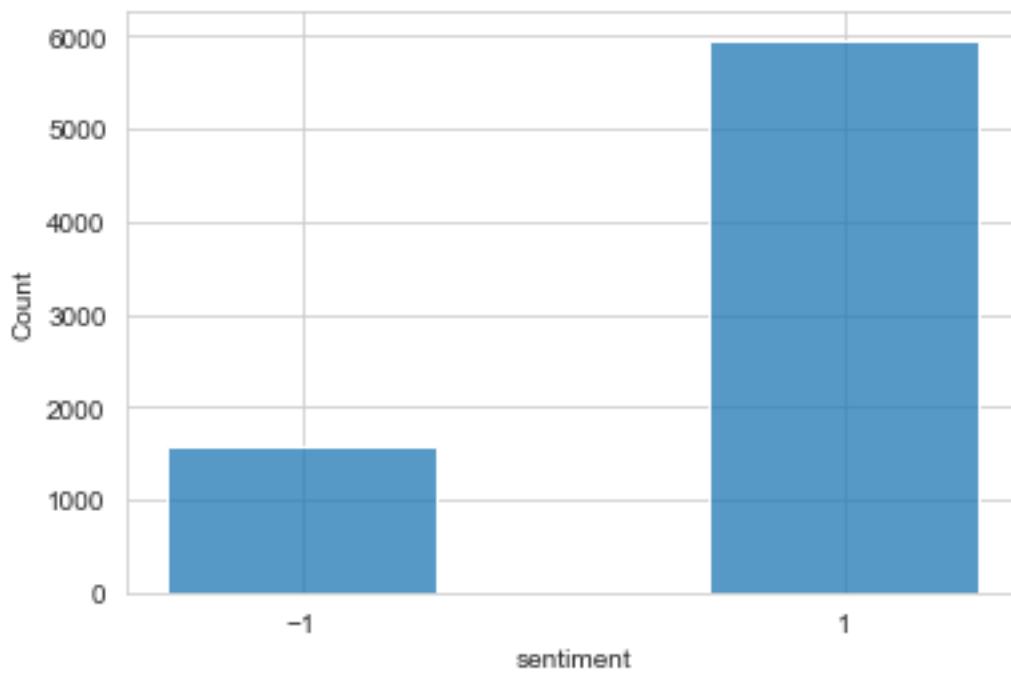
```
Naive Bayse Score: 88.27%

Confusion Matrix:
[[ 356  170]
 [ 122 1842]]

Classification Report:
              precision    recall  f1-score   support

     -1       0.74       0.68       0.71       526
      1       0.92       0.94       0.93      1964

 accuracy          0.88          0.88          0.88      2490
  macro avg       0.83       0.81       0.82      2490
weighted avg       0.88       0.88       0.88      2490
```

Types of the Recommendation Systems

There are many ways and complex algorithms used to build a recommender system. The following are fundamental approaches.

- **The Most Popular Item:** It is the simplest strategy, It works based on the assumption that the most popular item attracts most consumers or most users.
- **Association & Market Based Model:** The system makes recommendations based on the items in the consumer's basket. For instance, if the system detected that the buyer is purchasing ground coffee it would also suggest her to buy filters as well (observed association coffee - filters).
- **Content Filtering:** Uses metadata to determine the user's taste. For example, the system recommends the user movies based on their preferences of genres, actors, themes, etc. Such a system matches the user and the item based on similarity. For example, if the user watched and liked Terminator and Predator (both action movies with Arnold Schwarzenegger in the main role), it would probably recommend them to watch Commando.
- **Collaborative Filtering (CF):** It is an algorithmic architecture that recommends consumers items based on their observed behavior. There are two types of Collaborative Filtering frameworks: movie based on their gender but still focuses on the movie features the user exhibits to prefer.

Food Items Network analysis

The best approach to find the relationship between the ordered food is graphing the network, with food items as nodes, and whether they have been ordered together or not we can add an edge between nodes. Also adding the weight based on the number of times they have been ordered together can express the popularity of them being ordered together.

After creating the food network, let's see what are the top 10 foods ordered the most together:

❖ `top_weighted = sorted(X.edges(data=True), key=lambda t:t[2].get('weight'), reverse=True)[:10]`

```
items: ('Shoestring Fries (G)', 'Crispy Free Run Chicken Sandwich', {'weight': 621})
items: ('Shoestring Fries (G)', 'Grass Fed Bc Beef Cheesburger (4oz)', {'weight': 520})
items: ('Shoestring Fries (G)', 'Hammer Hot Chicken', {'weight': 354})
items: ('Shoestring Fries (G)', 'Grass Fed BC Double Cheeseburger (8oz)', {'weight': 310})
items: ('Joojeh Kabab With Rice', 'Koobideh Kabab With Rice', {'weight': 232})
items: ('Shoestring Fries (G)', 'Crispy Fish Sandwich', {'weight': 220})
items: ('Crispy Free Run Chicken Sandwich', 'Grass Fed Bc Beef Cheesburger (4oz)', {'weight': 202})
items: ('Shoestring Fries (G)', 'Coca Cola', {'weight': 201})
items: ('Shoestring Fries (G)', 'East Coast Lobster Roll', {'weight': 179})
items: ('Shoestring Fries (G)', 'House Made Falafel Sandwich (ve)', {'weight': 176})
```

As seen above, Shoestring Fries has been ordered with Crispy Free Run Chicken Sandwich, 621 times, which makes it the highest group of foods ordered together.

Also it's interesting to know which foods have been ordered mostly with other foods, which means to find the foods with the highest number of neighbours:

❖ `nodes_degrees = [(n, len(list(X.neighbors(n)))) for n in X.nodes()]`
`nodes_degrees.sort(key = lambda x:x[1], reverse=True)[:10]`

```
[('Koobideh Kabab With Rice', 93),
 ('Kashk Bademjan', 92),
 ('Skewer Of Koobideh Kabob', 77),
 ('Shirazi Salad', 76),
 ('Joojeh Kabab With Rice', 69),
 ('Guacamole ', 57),
 ('Azteca Soup', 56),
 ('Extra Bread', 55),
 ('Shoestring Fries (G)', 55),
 ('Nachos', 55)]
```

Recommendation based on popularity of being ordered together

Now is the time to make a recommendation based on the most wighted edge of the food, below is a sample example:

```
❖ food = random.choice(items)
  print('food: ',food)
  recommend_top_addon(food)
```

```
food: Crispy Free Run Chicken Sandwich
total found: 50
1 : Shoestring Fries (G) ( 621 )
2 : Grass Fed Bc Beef Cheesburger (4oz) ( 202 )
3 : Fish And Chips (G) ( 152 )
4 : Grass Fed BC Double Cheeseburger (8oz) ( 134 )
5 : Coca Cola ( 124 )
6 : Hammer Hot Chicken ( 97 )
7 : Ginger Ale ( 88 )
8 : Root Beer ( 88 )
9 : Crispy Fish Sandwich ( 85 )
10 : Sparkling Water ( 84 )
11 : Popina Bottled Water ( 79 )
12 : East Coast Lobster Roll ( 75 )
13 : Powell Street, Ode to Citra (473 ML) ( 68 )
14 : Hoi Polloi Lager ( 61 )
15 : Lager Hoi Polloi ( 53 )
16 : House Made Falafel Sandwich (ve) ( 50 )
17 : Crispy Free Run Chicken Sandwich ( 37 )
18 : Clam Chowder ( 30 )
19 : Gin & Tonic ( 25 )
20 : Popina Water ( 23 )
21 : Vodka Soda ( 22 )
22 : Dark 'n' Stormy ( 22 )
23 : Paloma ( 20 )
...
47 : Rum ( 1 )
48 : HH Cheese Burger ( 1 )
49 : Vodka ( 1 )
50 : Green Lentil Salad ( 1 )
```

Based on the recommendation algorithm, if the user already selected the Crispy Free Run chicken Sandwich, we recommend to add a Shoestring Fries to the current order.

Content Based Recommendation

Content-based filtering uses item features to recommend other items similar to what the user likes, based on their previous actions or explicit feedback.

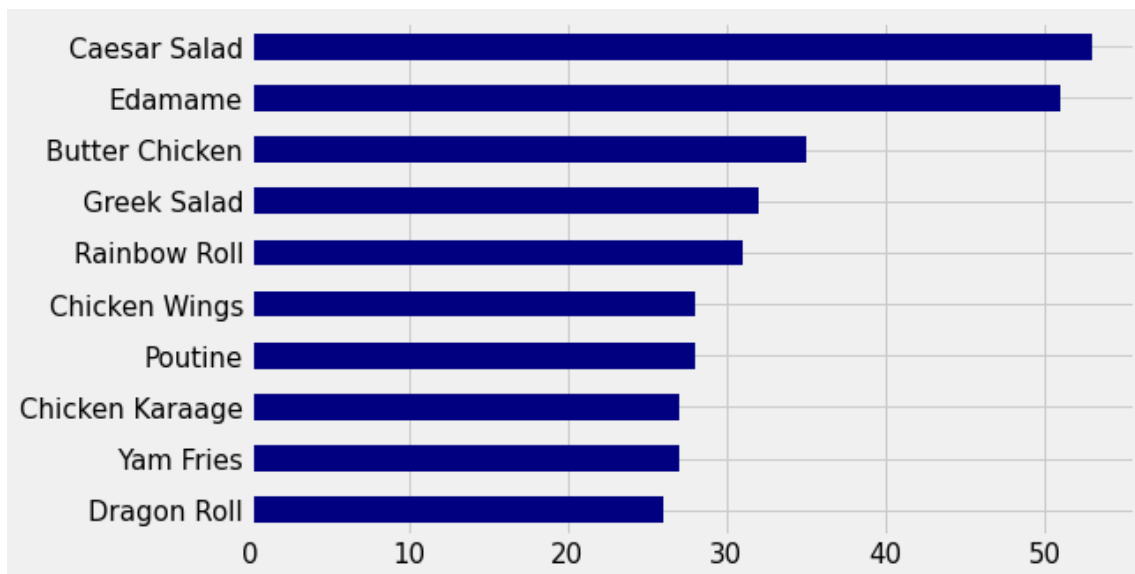
For this project context, we believe user liked the item implicitly if they ordered it.

For item features, we are using the item description text, which we first tokenize and then vectorize it to be used in our model.

The Most Popular Item

As mentioned before, one type of the recommendation is just suggest the most popular food. Let's just plot the top 10 popular foods.

```
❖ food_items['name'].value_counts()[0:10].plot(kind='barh', figsize=[8,5], fontsize=15,color='navy').invert_yaxis()
```



Ingredient based recommendation

Another type of recommendation we would like to have, is suggest the user based on the ingredients. For example:

```
ingredient1 = 'mushroom'
ingredient2 = 'onion'
ingredient3 = 'rice'
ingredient4 = 'carrots'

ingredients = [ingredient1,ingredient2,ingredient3,ingredient4]
ingredient_recommendation(food_items,ingredients)
```

```
Total food item found: 1
```

```
item name: Vegetable Fried Rice
```

```
item description is: Carrots, corn, onions, mushrooms, and peas with a fried egg.
```

Item Similarity Recommendation

This recommendation is based on the similar food to the most ordered food by the user. So first the system will calculate the cosine similarity of all the foods based on the food description. Then finds the most popular food in the customer's order history. Finally find the similar foods to the top ordered food.

Tokenize:

Combine the food name and description and then tokenize the text.

```
❖ food_items['bow'] = food_items['name'].map(str) + ',' + food_items['description'].map(str)
  food_items['tokenized'] = food_items.bow.apply(tokenize)
  food_items['tokens'] = food_items.tokenized.apply(lambda x: ','.join(map(str,x)))
```

Vectorize:

```
❖ counter = CountVectorizer()
  count_matrix = counter.fit_transform(food_items.tokens)
```

Similarity calculation:

```
❖ cosine_sim = cosine_similarity(count_matrix)
```

Recommendation:

Systems get the userId and fetch the top ordered food and based on the calculated similarity and will suggests new foods. For example:

```
recommended_items = historical_recommend(sample_userid,orders,food_items,cosine_sim)
```

```
Top order food is: Koobideh Kebab
```

```
Top order food description is: Two skewers of seasoned premium ground beef. Served with Persian saffron rice and grilled tomato.
```

```
Total similar foods count: 23
```

```
item: Koobideh Kebab (2 Pcs)
item: Beef Kebab (koobideh)
item: Koobideh Kebab (2 Pcs)
item: Loghme
item: 3. Koobideh (2 Pcs)
item: Chicken with Bone (2 Skewers)
item: Lamb Soltani
item: Koobideh With Sangak Bread (Naan)
item: Loghme (2 Pcs)
item: Platter For Four
item: 8. Loghme Kabob (2 Pcs)
item: Lamb Soltani Kabob
item: Soltani Kabob
item: Koobideh Ground Beef
item: Vaziri Kabob
item: Loghme Kebab
item: 6. Mini Koobideh (1 Pc)
item: Koobideh
item: Chelo Kabab Koobideh
item: Mini Koobideh
item: Kubideh Kebab (2 Pcs)
item: Soltani
item: Platter For Two
```

Clustering:

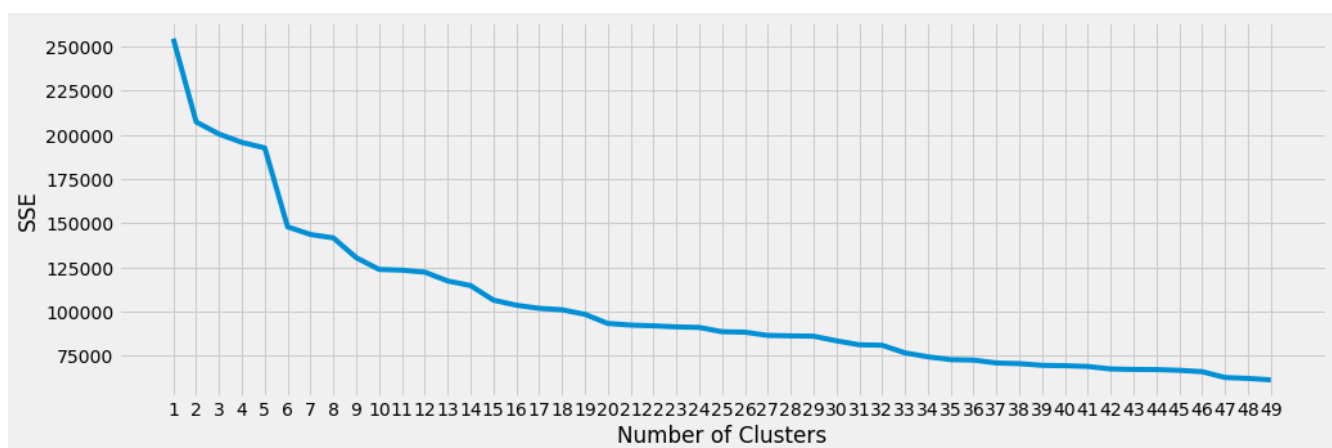
In this section we will cluster the similar foods. After combining and tokenizing the item name and description, we create the similarity matrix by calculating the cosine_similarity.

```
❖ counter = CountVectorizer()  
❖ item_count_matrix = counter.fit_transform(items.tokens)  
❖ cosine_sim = cosine_similarity(item_count_matrix)  
❖ print(cosine_sim.shape)
```

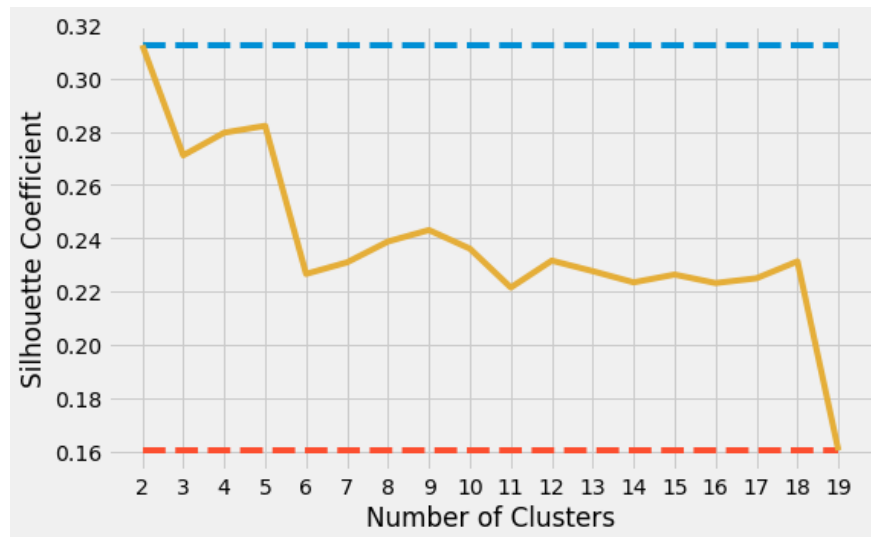
The similarity dimension is pretty high, around 41k, therefore we will reduce the dimension by fitting our cosine_sim data into a PCA model with 50 components:

```
❖ pca = PCA(n_components=50)  
❖ pca_transform = pca.fit_transform(cosine_sim)  
❖ print(pca_transform.shape)
```

Let's begin with exploring the parameters of the K Means clustering algorithm. I first predetermined a set amount of clusters. Data points which are closest to a given cluster center get grouped together. The sum of squared errors (SSE) is a metric used to measure the efficacy of clustering. The lower the SSE the more effective the parameters are. As the number of clusters increased the SSE decreased, revealing that a high number of clusters was necessary to effectively group films. Below is chart showing the SSE decreasing as number of clusters increases:



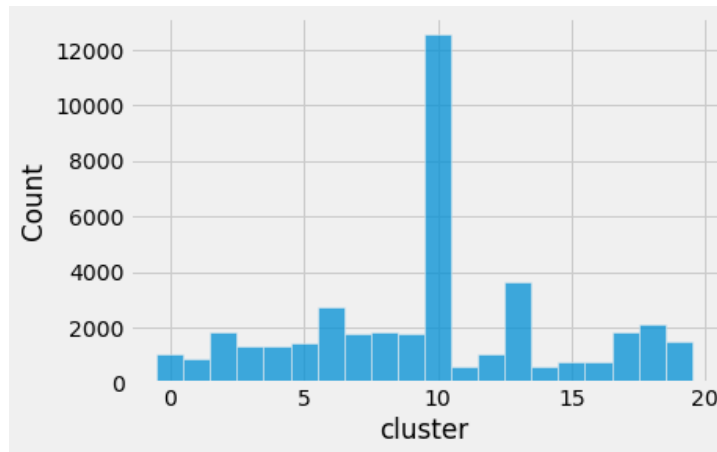
Next I employed a method that produces the Silhouette of Coefficients. This method gives a score that represents how similar a datapoint is to its cluster compared to other clusters. It evaluates the efficacy of clustering performance. The score ranges from -1 to 1, where a higher score is better. A higher score shows that data points are close/similar to their own cluster and far/different from neighbouring clusters. Below is a chart showing Silhouette scores for various cluster amounts. You can see the score decrease dramatically after the first 5 clusters are formed. This shows that after establishing 5 clusters the difference between grouped data points shrinks and they share similarities despite different cluster membership.



As seen in the above graph, the Silhouette Coefficient dropped around 18 clusters. Therefore for our model we will use 18 clusters in our KMeans.

```
❖ kmeans = KMeans(n_clusters=20)
❖ items['cluster'] = kmeans.fit_predict(pca_transform)
```

Plotting a histogram will display the distribution of the items in different clusters:



Beside the cluster “10” that seems to have huge number of items, which may caused by some outliers, other clusters seems to have reasonable amount of items.

Recommendation based on similar clusters

After clustering the foods in multiple clusters based on their cosine similarity, we can recommend new foods, based on user previous orders. For example:

- ❖ sample_food = 'Strike Signature Fried Chicken'
- ❖ recommend_clustered_food(sample_food)

```
Chicken Scramble
Chicken Strips And Fries
Chicken Hash-n-eggs
Chicken Pear Sandwich
Clubhouse Wrap
Chicken Caesar Wrap
Chicken And Waffles
Chicken Cucumber Wrap
Grilled Chicken Caesar Wrap
Chicken Burger
```

Collaborative Filter Based Recommendation

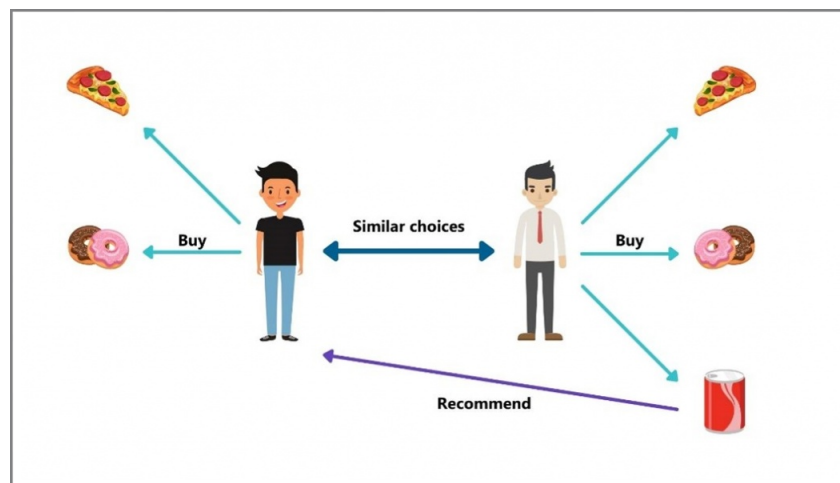
The idea behind collaborative filtering is to consider users' rating on different food items and recommend the food to each user based on the user's previous rankings and the opinion of other similar types of users.

There are basically two categories of Collaborative filtering:

- ❖ **User-based**: measure the similarity between target users and other users
- ❖ **Item-based**: measure the similarity between the items that target users rates/ interacts with and other items

Typically, the workflow of a collaborative filtering system is:

1. A user expresses his or her preferences by rating items of the system. These ratings can be viewed as an approximate representation of the user's interest in the corresponding domain.
2. The system matches this user's ratings against other users' and finds the people with most "similar" tastes



Data limitation problem:

The feedback about movies falls into one of two categories:

- ❖ **Explicit**— users specify how much they liked a particular item by providing a numerical rating.
- ❖ **Implicit**— if a user ordered a food, the system infers that the user is interested - we will add random rating from 3-5 to the foods with no rating for the foods that user has already ordered.

```
❖ data['random'] = [random.randrange(3, 6, 1) for i in range(data.shape[0])]
  data['rating'].fillna(value = data['random'], inplace= True)
```

To do Item-based Collaborative Filtering, I have used a package called [Surprise](#), which is a Python [scikit](#) for building and analyzing recommender systems that deal with explicit rating data Using cross-validation, and compute the MAE and RMSE of the [SVD](#) and [KNNWithMeans](#) algorithms.

SVD:

```
❖ param_grid = {'n_epochs': [5, 10], 'lr_all': [0.002, 0.005], 'reg_all': [0.4, 0.6]}
❖ gs_svd = GridSearchCV(SVD, param_grid, measures=['rmse', 'mae'], cv=3)
❖ gs_svd.fit(rating_data_surprise)
❖ print (gs_svd.s.best_score['rmse'])
❖ print(gs_svd.best_params['rmse'])
```

```
0.8474494796248077
{'n_epochs': 10, 'lr_all': 0.005, 'reg_all': 0.6}
```

KNN with means:

```
❖ sim_options = {
    "name": ["msd", "cosine"],
    "min_support": [3, 4, 5],
    "user_based": [False, True] }
❖ param_grid = {'sim_options': sim_options}
❖ gs_knn = GridSearchCV(KNNWithMeans, param_grid, measures=["rmse", "mae"], cv=3)
❖ gs_knn.fit(rating_data_surprise)
```

```
0.9413313583932665
{'sim_options': {'name': 'cosine', 'min_support': 4, 'user_based': False}}
```

After fitting the data to both models with the optimal parameters, apparently KNN score was better than SVD .

Lets use the models to generate a predicted rating that User X would give Item X.

KNN Prediction

```
❖ knn = gs_knn.best_estimator['rmse']  
❖ knn.fit(rating_data_surprise.build_full_trainset())  
❖ pred = knn.predict(sample_userid, sample_itemid)  
❖ print(f'The predicted rating that User {pred[0]} would give item {pred[1]} is {round(pred[3], 2)} ')
```

```
The predicted rating that User 4350 would give item 83420 is 4.03
```

SVD Prediction:

```
❖ svd = gs_svd.best_estimator['rmse']  
❖ svd.fit(rating_data_surprise.build_full_trainset())  
❖ pred = svd.predict(sample_userid, sample_itemid)  
❖ print(f'The predicted rating that User {pred[0]} would give item {pred[1]} is {round(pred[3], 2)} ')
```

```
The predicted rating that User 4350 would give item 83420 is 3.92
```

As the actual rating for the item is shown below (5), KNN as expected predicted a closer rating(4.03) to the true value comparing to SVD prediction (3.92)

	userid	itemid	rating
19785	4350	83420	5.0
21320	4637	83420	3.0
21389	4646	83420	5.0

Content Based Recommendation