Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ

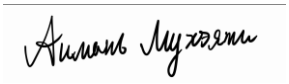
КАФЕДРА СИСТЕМЫ ОБРАБОТКИ ИНФОРМАЦИИ И УПРАВЛЕНИЯ

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

НА ТЕМУ:

РЕКОМЕНДАТЕЛЬНЫЙ АЛГОРИТМ ДЛЯ ПРИЛОЖЕНИЙ РЕСТОРАНОВ НА ВЫНОС

Студент ИУ5-33М  30.11.2021 Аимань Мухэяти
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель _____ _____
(Подпись, дата) (И.О.Фамилия)

Консультант _____ _____
(Подпись, дата) (И.О.Фамилия)

2021 г.

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ
Заведующий кафедрой __ИУ5_____
(Индекс)
_____ _____
(И.О.Фамилия)
« 12 » 20 2020 г.

З А Д А Н И Е
на выполнение научно-исследовательской работы

по теме Рекомендательный алгоритм для приложений ресторанов на вынос
Студент группы ИУ5-33М
Аимань Мухэяти
(Фамилия, имя, отчество)

Направленность НИР (учебная, исследовательская, практическая, производственная, др.)
Учебная, практическая
Источник тематики (кафедра, предприятие, НИР) Кафедра

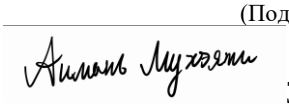
График выполнения НИР: 25% к ___ нед., 50% к ___ нед., 75% к __ нед., 100% к ___ нед.

Техническое задание
отслеживание объектов_на основе питона
Оформление научно-исследовательской работы:

Расчетно-пояснительная записка на __50__ листах формата А4.
Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)
_____
_____
_____
_____

Дата выдачи задания « __30_ » _____11_____ 20_21_ г.

Руководитель НИР _____ _____
(Подпись, дата) (И.О.Фамилия)

Студент  30.11.2021 Аимань Мухэяти
(Подпись, дата) (И.О.Фамилия)

Примечание: Задание оформляется в двух экземплярах: один выдается студенту, второй хранится на кафедре.

Аннотация

Набор данных, использованный в работе, содержит 5,8 миллиона данных о заказах, в которых участвуют более 18 000 пользователей и 100 ресторанов. В работе использованы три популярные рекоментательные прототипы для разработки алгоритмов персонализированной рекомендации ресторанов на вынос для пользователей.

Содержание

1. Два режима традиционного рекомендательного алгоритма

1.1 Анализ с помощью количественной обратной связи (оценок)

Первый основан на рейтинге клиентов. Почти на всех веб-сайтах или приложениях есть система обратной связи с клиентами после использования товаров или услуг. Клиенты выставляют оценки в соответствии с их удовлетворенностью. И компания может использовать эту информацию.

С помощью совместной фильтрации (CF) алгоритм находит других клиентов, которые показывают схожий рейтинг с определенным клиентом. Если товары или услуги, купленные аналогичным клиентом, еще не были куплены, алгоритм рекомендует товар для клиента. Это была базовая логика в раннюю эпоху системы рекомендаций. На самом деле, в настоящее время он в основном не используется.

А матричная фарторизация (MF) - это метод, который находит скрытый фактор. Путем разложения матрицы, которая содержит информацию о клиенте и товаре, на латентную матрицу пользователя и латентную матрицу товара, алгоритм прогнозирует рейтинговую матрицу. То есть он также прогнозирует рейтинг для ожидаемого клиента. Но использование всей информации - это другая точка зрения по сравнению с совместной фильтрацией, поэтому ее способность к предварительной обработке более сильна. В частности, эта модель может быть построена с помощью глубокого обучения (DL). С помощью многослойного анализа точность модели может быть повышена.

1.2 Анализ по содержанию

Другой подход основан на функциях, называемых основанными на "содержимом". Другими словами, алгоритм находит предметы, которые имеют схожие характеристики.

Вычисляя Частоту термина - Обратную частоте документа (TF-IDF), можно получить сходство, подобное косинусному сходству. TF означает значение частоты появления определенных слов в документе. TF-IDF состоит из TF и IDF. Если этот балл высокий, это слово означает, что оно часто встречается в этом документе, а не в других документах. Косинусное сходство - это мера сходства между двумя ненулевыми векторами пространства внутреннего произведения. Когда клиент купил определенный товар, компания должна порекомендовать ему аналогичный товар. И в этом процессе вычисляется частота каждого слова, исключая слова, которые оказывают низкое влияние.

Doc2Vec (Внедрение документа с векторами абзацев) - это расширенный алгоритм в Word2Vec, который предсказывает слово путем последовательного анализа абзаца. Вектор абзаца - это вектор существующего вектора слов (с учетом слов в пределах размера окна) плюс матрица абзаца. Путем переноса документов в вектор документов (встраивание) и вычисления сходства между документами этот алгоритм рекомендует похожие элементы.

Мы выбираем две функции в качестве содержимого. Один из них - "тег (название)", в котором есть десять параметров, описывающих каждого поставщика, другой - время открытия, которое может описать целевой временной интервал ресторана. Это "содержимое" анализируется с помощью описанного выше алгоритма. То есть мы рассматриваем это содержимое как один документ и применяем метод для анализа документа.

```
import pandas as pd
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
import missingno as msno
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Embedding, Dot, Add, Flatten
from tensorflow.keras.regularizers import l2
from tensorflow.keras.optimizers import SGD, Adam, Adamax
from difflib import SequenceMatcher
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from gensim.models import Word2Vec
```

2. Исследовательский анализ данных(EDA)

2.1 Датасет «train_full»

Во-первых, просмотрите исходные наборы данных и просто визуализируйте данные

```
#загрузим дата
data1=pd.read_csv("/kaggle/input/restaurant-recommendation-challenge/train_full
data1.head()
```

	customer_id	gender	status_x	verified_x	created_at_x	updated_at_x	location_number	location_type	latitude
0	TCHWPBT	Male	1	1	2018-02-07 19:16:23	2018-02-07 19:16:23	0	Work	-96.
1	TCHWPBT	Male	1	1	2018-02-07 19:16:23	2018-02-07 19:16:23	0	Work	-96.
2	TCHWPBT	Male	1	1	2018-02-07 19:16:23	2018-02-07 19:16:23	0	Work	-96.
3	TCHWPBT	Male	1	1	2018-02-07 19:16:23	2018-02-07 19:16:23	0	Work	-96.
4	TCHWPBT	Male	1	1	2018-02-07 19:16:23	2018-02-07 19:16:23	0	Work	-96.

5 rows × 73 columns

```
data1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5802400 entries, 0 to 5802399
Data columns (total 73 columns):
 #   Column                  Dtype
---  ------                  -----
 0   customer_id             object
 1   gender                  object
 2   status_x                int64
 3   verified_x              int64
 4   created_at_x            object
 5   updated_at_x            object
 6   location_number         int64
 7   location_type           object
 8   latitude_x              float64
 9   longitude_x             float64
 10  id                      int64
 11  authentication_id       float64
 12  latitude_y              float64
 13  longitude_y             float64
 14  vendor_category_en      object
 15  vendor_category_id      float64
 16  delivery_charge         float64
 17  serving_distance        float64
 18  is_open                 float64
 19  OpeningTime             object
 20  OpeningTime2            object
 21  prepration_time         int64
 22  commission              float64
 23  is_akeed_delivering     object
 24  discount_percentage     float64
 25  status_y                float64
 26  verified_y              int64
 27  rank                    int64
 28  language                object
 29  vendor_rating           float64
 30  sunday_from_time1       object
 31  sunday_to_time1         object
 32  sunday_from_time2       object
 33  sunday_to_time2         object
 34  monday_from_time1       object
 35  monday_to_time1         object
 36  monday_from_time2       object
 37  monday_to_time2         object
 38  tuesday_from_time1      object
```

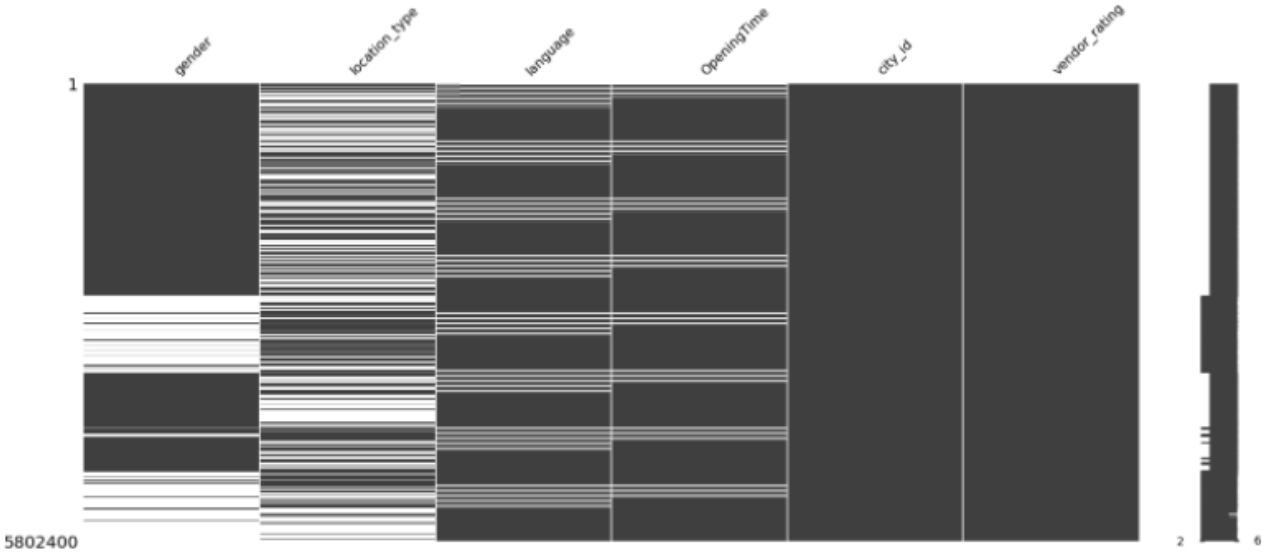
```
 39  tuesday_to_time1        object
 40  tuesday_from_time2      object
 41  tuesday_to_time2        object
 42  wednesday_from_time1    object
 43  wednesday_to_time1      object
 44  wednesday_from_time2    object
 45  wednesday_to_time2      object
 46  thursday_from_time1     object
 47  thursday_to_time1       object
 48  thursday_from_time2     object
 49  thursday_to_time2       object
 50  friday_from_time1       object
 51  friday_to_time1         object
 52  friday_from_time2       object
 53  friday_to_time2         object
 54  saturday_from_time1     object
 55  saturday_to_time1       object
 56  saturday_from_time2     object
 57  saturday_to_time2       object
 58  primary_tags            object
 59  open_close_flags        float64
 60  vendor_tag              object
 61  vendor_tag_name         object
 62  one_click_vendor        object
 63  country_id              float64
 64  city_id                 float64
 65  created_at_y            object
 66  updated_at_y            object
 67  device_type             int64
 68  display_orders          int64
 69  location_number_obj     int64
 70  id_obj                  int64
 71  CID X LOC_NUM X VENDOR  object
 72  target                  int64
dtypes: float64(16), int64(12), object(45)
memory usage: 3.2+ GB
```

Набор данных содержит 73 столбца и 5802400 строк. Поэтому мы просто просматриваем примерно 8-10 столбцов:

- gender: пол клиента
- location type: заказы клиентов из одного или нескольких местоположений.
- language: Выбранный язык
- openinig Time: рабочее время поставщика
- city_id: Идентификатор города
- vendor_rating: средний рейтинг поставщика.

```
# Визуализируем пропущенные значения в виде матрицы
main_df=data1[['gender','location_type','language','OpeningTime','city_id','ven
msno.matrix(main_df)
```

```
<AxesSubplot:>
```

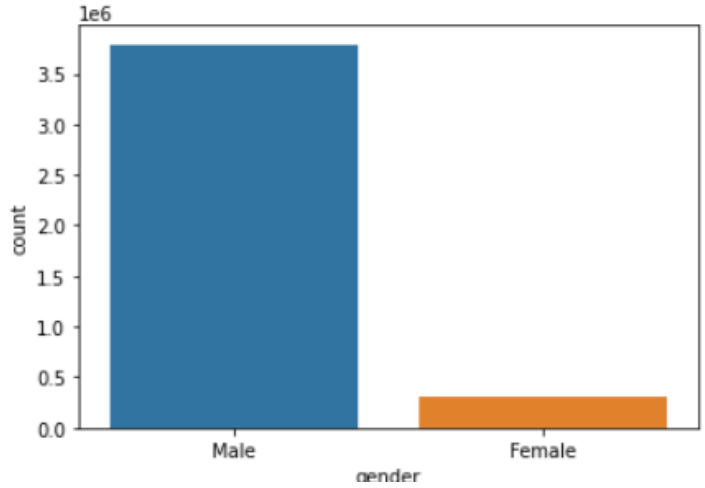



```
#пол

print(data1['gender'].value_counts())
gender_null = np.count_nonzero(data1['gender'].isnull())
print(gender_null)

gender_null/data1.shape[0]
# Значения : Male / Female
# Процентов пропущенных значений: 30%
```

```
Male      3789100
Female     308200
Name: gender, dtype: int64
1705100
0.2938611608989384
```

```
sns.countplot('gender',data=data1)
```

```
<AxesSubplot:xlabel='gender', ylabel='count'>
```

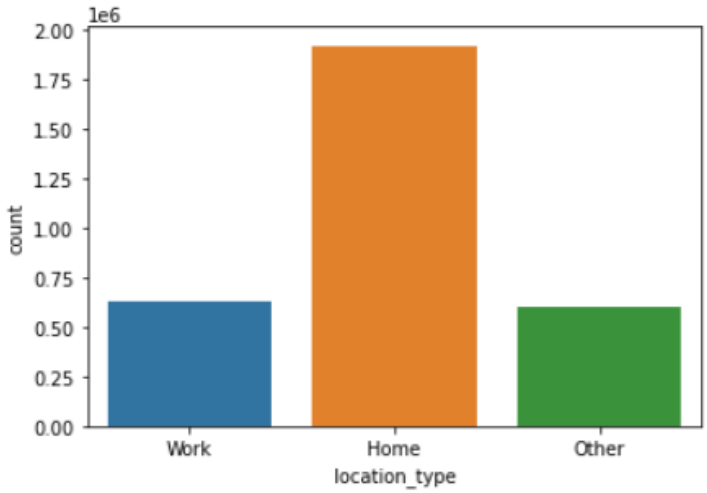


```
#location type
print(data1['location_type'].value_counts())
location_null = np.count_nonzero(data1['location_type'].isnull())
print(location_null)

print("null Ratio : ", location_null/data1.shape[0])
# Значения: Home / Work / Other  but, we don't know the meaning of 'other'
# Процентов пропущенных значений: 45%
```

```
Home     1921700
Work      627400
Other     599100
Name: location_type, dtype: int64
2654200
null Ratio :  0.45743140769336826
```

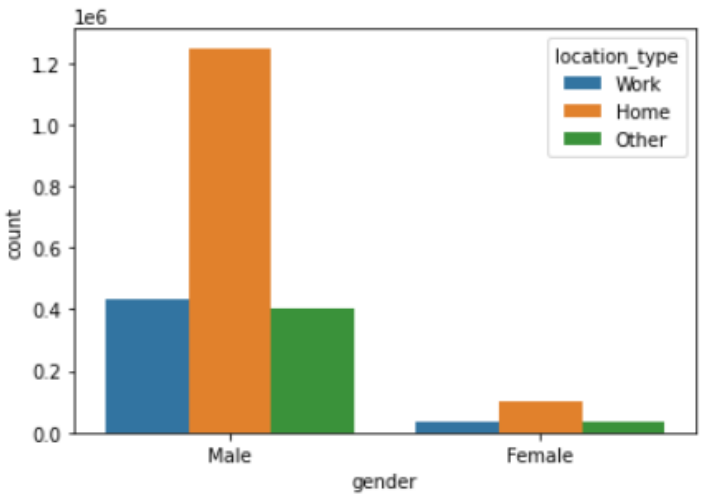
```
sns.countplot('location_type',data=data1)
```

```
<AxesSubplot:xlabel='location_type', ylabel='count'>
```




```
# Разница в типе местоположения по полу
sns.countplot(data1['gender'],hue=data1['location_type'])
```

```
<AxesSubplot:xlabel='gender', ylabel='count'>
```



```
#Language
print(data1['language'].value_counts())
null = np.count_nonzero(data1['language'].isnull())
print(null)

null/data1.shape[0]
# Значения: it has only EN(english)
# Процентов пропущенных значений: 15%
```

```
EN    4932040
Name: language, dtype: int64
870360
0.15
```

```
#Opening Time
null = np.count_nonzero(data1['OpeningTime'].isnull())
print(null)

null/data1.shape[0]
# Процентов пропущенных значений: 9%
```

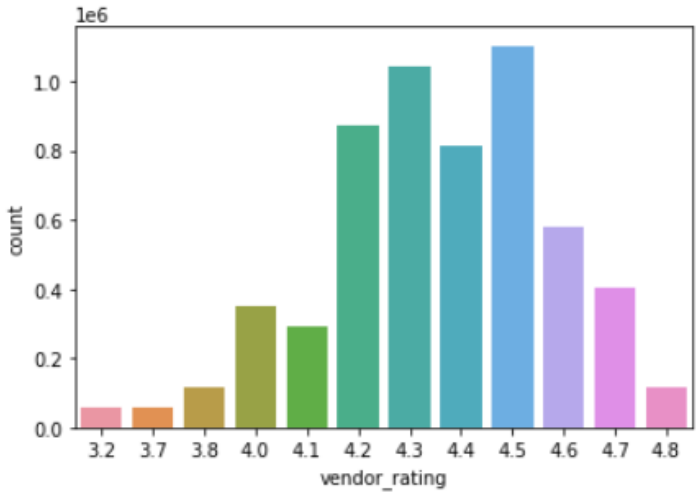
```
522216
0.09
```

```
# Средний рейтинг поставщика

print(data1['vendor_rating'].value_counts())
null = np.count_nonzero(data1['vendor_rating'].isnull())
print(null)

null/data1.shape[0]
# Здесь нет NaN
```

```
4.5    1102456
4.3    1044432
4.2     870360
4.4     812336
4.6     580240
4.7     406168
4.0     348144
4.1     290120
4.8     116048
3.8     116048
3.2      58024
3.7      58024
Name: vendor_rating, dtype: int64
0
0.0
```

```
sns.countplot('vendor_rating',data=data1)
```

```
<AxesSubplot:xlabel='vendor_rating', ylabel='count'>
```



Распределение Vendor_rating довольно переполнен между 4.2 и 4.5.

2.2 Датасет «orders»

```
data2 = pd.read_csv("/kaggle/input/restaurant-recommendation-challenge/orders.c
data2.head()
```

	akeed_order_id	customer_id	item_count	grand_total	payment_mode	promo_code	vendor_discount_amount
0	163238.0	92PEE24	1.0	7.6	2	NaN	0.0
1	163240.0	QS68UD8	1.0	8.7	1	NaN	0.0
2	163241.0	MB7VY5F	2.0	14.4	1	NaN	0.0
3	163244.0	KDJ951Y	1.0	7.1	1	NaN	0.0
4	163245.0	BAL0RVT	4.0	27.2	1	NaN	0.0

5 rows × 26 columns

```
data2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 135303 entries, 0 to 135302
Data columns (total 26 columns):
 #   Column                          Non-Null Count   Dtype  
---  ------                          --------------   -----  
 0   akeed_order_id                  135233 non-null  float64
 1   customer_id                     135303 non-null  object 
 2   item_count                      128378 non-null  float64
 3   grand_total                     135303 non-null  float64
 4   payment_mode                    135303 non-null  int64  
 5   promo_code                      4305 non-null    object 
 6   vendor_discount_amount          135303 non-null  float64
 7   promo_code_discount_percentage  65880 non-null   float64
 8   is_favorite                     100108 non-null  object 
 9   is_rated                        135303 non-null  object 
 10  vendor_rating                   45220 non-null   float64
 11  driver_rating                   135303 non-null  float64
 12  deliverydistance                135303 non-null  float64
 13  preparationtime                 79743 non-null   float64
 14  delivery_time                   5123 non-null    object 
 15  order_accepted_time             86955 non-null   object 
 16  driver_accepted_time            46458 non-null   object 
 17  ready_for_pickup_time           84249 non-null   object 
 18  picked_up_time                  83865 non-null   object 
 19  delivered_time                  85741 non-null   object 
 20  delivery_date                   35544 non-null   object 
 21  vendor_id                       135303 non-null  int64  
 22  created_at                      135303 non-null  object 
 23  LOCATION_NUMBER                 135303 non-null  int64  
 24  LOCATION_TYPE                   86410 non-null   object 
 25  CID X LOC_NUM X VENDOR          135303 non-null  object 
dtypes: float64(9), int64(3), object(14)
memory usage: 26.8+ MB
```

Это набор данных содержит 26 столбцов и 135303 строки. Объясним некоторые столбцы

- akeed_order_id: Уникальный идентификатор клиента, используемый в train_locations и train_orders
- vendor_rating: Рейтинги оцениваются клиентами, которые используют поставщика
- vendor_id: Уникальный идентификатор поставщика
- customer_id: Уникальный идентификатор клиента

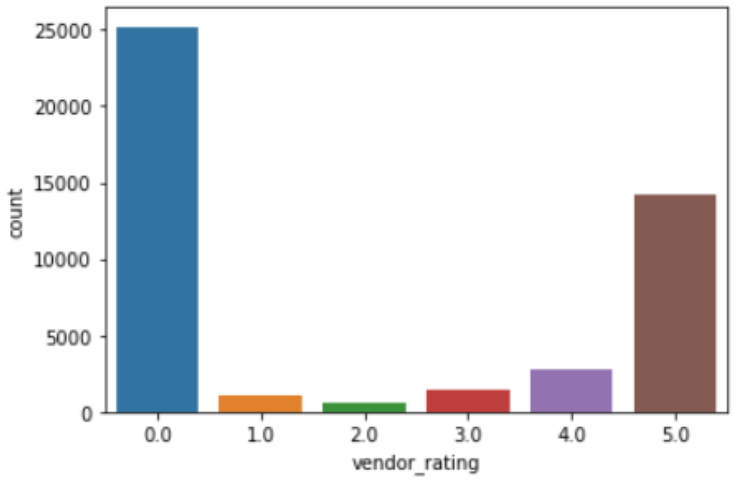
```
#vendor_rating

print(data2['vendor_rating'].value_counts())
null = np.count_nonzero(data2['vendor_rating'].isnull())
print(null)

null/data2.shape[0]
#Процентов пропущенных значений: 66%
```

```
0.0    25175
5.0    14212
4.0     2748
3.0     1426
1.0     1029
2.0      630
Name: vendor_rating, dtype: int64
90083
```

```
0.6657871591908531
```

```
sns.countplot('vendor_rating',data=data2)
```

```
<AxesSubplot:xlabel='vendor_rating', ylabel='count'>
```



Оценки, за исключением 0 баллов, составляют почти 5 баллов.

```
#vendor_id
print(data2['vendor_id'].value_counts())
# Количество уникальных поставщиков равно 100
```

```
113    7807
105    5562
79     5117
84     5001
78     4643
       ...
304     562
271     559
196     512
250     503
295     474
Name: vendor_id, Length: 100, dtype: int64
```

```
#customer_id
print(data2['customer_id'].value_counts())
# Количество клиентов: 27445
```

```
XW90EAP    262
TL7Z2DM    151
VDEJEMP    146
HJFTTGW    128
B28LJKM    119
          ...
UU3ZMK4      1
Z1RFX9S      1
439W3X1      1
JGAKTC7      1
6PWKEPD      1
Name: customer_id, Length: 27445, dtype: int64
```

3. Предварительная обработка

3.1 Объединение наборов данных

3.1.1 Для совместной фильтрации

```
# Выбираем несколько столбцов из датасет 1(train_full.csv)
# vendor_rating - > mean_rating
dataset1 = data1[['customer_id','gender','location_type','id','OpeningTime','la
dataset1.rename(columns = {"vendor_rating": "mean_rating"}, inplace = True)

# Сделаем производные переменные "all" со столбцами id и customer_id
cols = ['customer_id', 'id']
dataset1['all'] = dataset1[cols].apply(lambda row: '_'.join(row.values.astype(s

dataset1.head()
```

	customer_id	gender	location_type	id	OpeningTime	language	mean_rating	serving_distance	
0	TCHWPBT	Male	Work	4	11:00AM-11:30PM	EN	4.4	6.0	Arabic
1	TCHWPBT	Male	Work	13	08:30AM-10:30PM	EN	4.7	5.0	Breakfast,Cake
2	TCHWPBT	Male	Work	20	08:00AM-10:45PM	EN	4.5	8.0	Breakf
3	TCHWPBT	Male	Work	23	10:59AM-10:30PM	EN	4.5	5.0	
4	TCHWPBT	Male	Work	28	11:00AM-11:45PM	EN	4.4	15.0	

```
# Удалим дубликаты на основе "all" производных переменных
dataset1.drop_duplicates(['all'],inplace=True)
```

```
# Выбираем несколько столбцов из датасет 2(orders.csv)
# vendor_id - > id
dataset2 = data2[['akeed_order_id','customer_id','vendor_id', 'item_count', 'gr
dataset2.rename(columns = {"vendor_id": "id"}, inplace = True)

# Сделаем производные переменные "all" со столбцами id и customer_id
cols = ['customer_id', 'id']
dataset2['all'] = dataset2[cols].apply(lambda row: '_'.join(row.values.astype(s

dataset2.head()
```

	akeed_order_id	customer_id	id	item_count	grand_total	vendor_rating	all
0	163238.0	92PEE24	105	1.0	7.6	NaN	92PEE24_105
1	163240.0	QS68UD8	294	1.0	8.7	NaN	QS68UD8_294
2	163241.0	MB7VY5F	83	2.0	14.4	NaN	MB7VY5F_83
3	163244.0	KDJ951Y	90	1.0	7.1	NaN	KDJ951Y_90
4	163245.0	BAL0RVT	83	4.0	27.2	NaN	BAL0RVT_83

```
# Количество строк отбрасывается повторяющиеся строки
print(dataset1.shape)
print(dataset2.shape)
```

```
(3452300, 11)
(135303, 7)
```

Соединяем датасет 1 и датасет 2.

```
df1=pd.merge(dataset1,dataset2,on='all',how='inner')
df1.head()
```

	customer_id_x	gender	location_type	id_x	OpeningTime	language	mean_rating	serving_distance	
0	TCHWPBT	Male	Work	113	10:59AM-10:59PM	EN	4.7	15.0	
1	TCHWPBT	Male	Work	237	08:30PM-11:59PM	EN	4.6	15.0	American,I
2	ZGFSYCZ	Male	Home	4	11:00AM-11:30PM	EN	4.4	6.0	Arabic,Br
3	ZGFSYCZ	Male	Home	28	11:00AM-11:45PM	EN	4.4	15.0	
4	ZGFSYCZ	Male	Home	28	11:00AM-11:45PM	EN	4.4	15.0	

```
df1.shape
```

```
(132027, 17)
```

```
# меняем названия некоторых столбцов и удаляем один и тот же столбец
df1.rename(columns = {"customer_id_x": "customer_id"}, inplace = True)
df1.rename(columns = {"id_x": "vendor_id"}, inplace = True)
df1.drop(['customer_id_y','id_y'],axis=1,inplace=True)
```

Видим, что количество строк уменьшается при объединении наборов данных

3.1.2 Для алгоритма на основе содержения

```
# Выберим несколько столбцов из датасет1 (очищенные данные)
df2=dataset1[['customer_id','id','vendor_tag_name']]
df2.rename(columns={'id':'vendor_id'},inplace=True)
df2.head()
```

	customer_id	vendor_id	vendor_tag_name
0	TCHWPBT	4	Arabic,Breakfast,Burgers,Desserts,Free Deliver...
1	TCHWPBT	13	Breakfast,Cakes,Crepes,Italian,Pasta,Pizzas,Sa...
2	TCHWPBT	20	Breakfast,Desserts,Free Delivery,Indian
3	TCHWPBT	23	Burgers,Desserts,Fries,Salads
4	TCHWPBT	28	Burgers

3.2 Очистка данных

```
cols=[ 'serving_distance', 'delivery_charge','item_count', 'grand_total', 'vend

def null_check(x):
  null = np.count_nonzero(df1[x].isnull())
  print(null)

  return null/df1.shape[0]

for i in cols:
  print(i,'null ratio :', null_check(i))
```

```
0
serving_distance null ratio : 0.0
0
delivery_charge null ratio : 0.0
6750
item_count null ratio : 0.05112590606466859
0
grand_total null ratio : 0.0
87951
vendor_rating null ratio : 0.6661591947101729
```

```
# Удалим столбцы "language"
df1.drop(['language'],axis=1,inplace=True)

# Удалим попрощунные значения из столбцов "gender"
df1 = df1[df1['gender'].notnull()].reset_index(drop=True)
```

Изменим столбец "пол".

```
#gender - > one-hot encoding (int)
sex=pd.get_dummies(df1["gender"], columns = ['gender'],prefix="sex",drop_first=

df1=pd.concat([df1,sex],axis=1)

# Удалим необработанный столбец "gender" (char)
df1.drop(['gender'],axis=1,inplace=True)
```

```
df1.rename(columns={'vendor_rating': 'rating'}, inplace=True)
```

```
print(df1.shape)
df1.head()
```

```
(104549, 14)
```

	customer_id	location_type	vendor_id	OpeningTime	mean_rating	serving_distance	
0	TCHWPBT	Work	113	10:59AM-10:59PM	4.7	15.0	Arabic,Desserts,
1	TCHWPBT	Work	237	08:30PM-11:59PM	4.6	15.0	American,Burgers,Desserts,D
2	ZGFSYCZ	Home	4	11:00AM-11:30PM	4.4	6.0	Arabic,Breakfast,Burgers,De
3	ZGFSYCZ	Home	28	11:00AM-11:45PM	4.4	15.0	
4	ZGFSYCZ	Home	28	11:00AM-11:45PM	4.4	15.0	

```
df1_train_for_anal = df1[:]
```

3.2.1 Для алгоритма на основе содержения

```
train_for=df1[:]
```

```
train_contents = train_for[['customer_id','vendor_id','OpeningTime','vendor_tag
train_contents.head()
```

	customer_id	vendor_id	OpeningTime	vendor_tag_name
0	TCHWPBT	113	10:59AM-10:59PM	Arabic,Desserts,Free Delivery,Indian
1	TCHWPBT	237	08:30PM-11:59PM	American,Burgers,Desserts,Donuts,Fries,Pasta,S...
2	ZGFSYCZ	4	11:00AM-11:30PM	Arabic,Breakfast,Burgers,Desserts,Free Deliver...
3	ZGFSYCZ	28	11:00AM-11:45PM	Burgers
4	ZGFSYCZ	28	11:00AM-11:45PM	Burgers

Предварительная обработка столбца "OpeningTime":

```
# Разделим столбец "OpeningTime" на 2 столбца (открыть /закрыть).
train_contents['OpeningTime'].fillna('-',inplace=True)

time_split= train_contents.OpeningTime.str.split('-')
open=time_split.str.get(0)
close=time_split.str.get(1)

train_contents['Open']=open
train_contents['Close']=close

# Заполним попрощуные значения пробелом
train_contents['Open'].fillna('',inplace=True)
train_contents['Close'].fillna('',inplace=True)

print(train_contents['Open'].unique())
print(train_contents['Close'].unique())
```

```
['10:59AM' '08:30PM' '11:00AM' '11.30am' '09:00AM' '11AM' '10:00AM'
 '08:00AM' '12:00PM' '9' '' '11:59AM' '09:59AM' '05:00PM' '09:00 AM'
 '01.00PM' '05:30PM' '08:30AM' '8:00AM' '11:15AM' '9:00AM' '7:58AM'
 '04:59PM' '00:01AM' '9am' '04:00PM' '06:15AM' '06:00PM' '11:30AM']
['10:59PM' '11:59PM' '11:30PM' '11:45PM' '10:30PM' '3:50pm' '11:00PM'
 '11PM' '3:30PM' '07:00PM' '111:00PM' '22' '' '2:15 am' '12:30PM'
 '10:45PM' '11:15PM' '11:45 PM' '02:.00AM' '09:30PM' '09:45PM' '08:00pm '
 '03:00PM' '04:00PM' '10:00PM' '11:59PM ' '09:01AM' '11:45' '11:01PM'
 '01:00AM' '10pm' '1:45PM' '10:45PM ' '12:00PM' '11.59 PM']
```

Создаем новые столбцы "утро, день, вечер" на основе времени открытия и закрытия.

```
def morning_func(x) :
  if x == "" :
    return None
  else :
    x1 = int(x[:2].replace(":", "").replace("a", ""))
    x2 = x[-2:]
    if (x1>=7 and x1 <= 10) and x2 == ("AM" or "am"):
      return 1
    elif x1 <=10 and len(x) <= 2 :
      return 1
    else :
      return 0
```

```
def afternoon_func(x) :
  if x == "" :
    return None
  else :
    x1 = int(x[:2].replace(":", "").replace("a", ""))
    x2 = x[-2:]
    if x1 <= 1 and x2 == "PM":
      return 1
    elif x1 == 12 and x2 == "PM":
      return 1
    elif x2 == ("AM" or "am"):
      return 1
    elif x1 <=10 and len(x) <= 2 :
      return 1
    else :
      return 0
def evening_func(x) :
  if x == "" :
    return None
  else :
    x1 = int(x[:2].replace(":",""))
    x2 = x[-2:]
    if (x1 >= 6 and x2 == "PM") or x2 == ("Am" or "am") :
      return 1
    elif x1 >= 22 and len(x)<=2:
      return 1
    else :
      return 0
train_contents["morning"] = train_contents["Open"].apply(morning_func)
train_contents["afternoon"] = train_contents["Open"].apply(afternoon_func)
train_contents["evening"] = train_contents["Close"].apply(evening_func)
train_contents[:2]
```

r_id	OpeningTime	vendor_tag_name	Open	Close	morning	afternoon	evening
113	10:59AM-10:59PM	Arabic,Desserts,Free Delivery,Indian	10:59AM	10:59PM	1.0	1.0	1.0
237	08:30PM-11:59PM	American,Burgers,Desserts,Donuts,Fries,Pasta,S...	08:30PM	11:59PM	0.0	0.0	1.0

Если "breakfast" находится в «vendor_tag_name», а утреннее значение равно нулю (пустое), заполним утренние и дневные значения равными 1.

```
# Проверим процент пропущенных значений 'vendor_tag_name'
null = np.count_nonzero(train_contents['vendor_tag_name'].isnull())
print(null)
print(null/train_contents.shape[0]) #1%

# Удалим пропущенные значения
train_contents= train_contents[train_contents['vendor_tag_name'].notnull()].res
null = np.count_nonzero(train_contents['vendor_tag_name'].isnull())
print(null)
```

```
1611
0.015409042649857962
0
```

```
# Очистка 'vendor_tag_name'

# Меняем все "chars" до "lower char"
train_contents['vendor_tag_name']=train_contents['vendor_tag_name'].apply(lambd

# str -> list
train_contents['vendor_tag']= train_contents['vendor_tag_name'].str.split(',')
train_contents['vendor_tag'].head()
```

```
0           [arabic, desserts, free delivery, indian]
1    [american, burgers, desserts, donuts, fries, p...
2    [arabic, breakfast, burgers, desserts, free de...
3                                           [burgers]
4                                           [burgers]
Name: vendor_tag, dtype: object
```

```
# Если 'breakfast' в tag, то добавим "morning"
def breakfast1(tag,x2):
  if any('breakfast' in i for i in tag) and np.isnan(x2)==True :
    return 1
  else:
    return x2
train_contents['mor2']=train_contents.apply(lambda x:  breakfast1(x['vendor_tag
for i in range(len(train_contents['afternoon'])):
  if (np.isnan(train_contents['morning'][i])==True) and (train_contents['mor2']
    train_contents['afternoon'][i]=1
  else:
    pass
for i in range(len(train_contents['evening'])):
  if (np.isnan(train_contents['morning'][i])==True) and (train_contents['mor2']
    train_contents['evening'][i]=0
  else:
    pass
```

```
# Проверим и удалим пропущенные значения
null = np.count_nonzero(train_contents['mor2'].isnull())
print(null)
print(null/train_contents.shape[0]) #0.4 %

train_contents= train_contents[train_contents['mor2'].notnull()].reset_index(dr
train_contents.drop(['morning'],axis=1,inplace=True)

train_contents.rename(columns={'mor2':'morning'},inplace=True)
```

```
487
0.004731003128096524
```

```
train_contents[:3]
```

	customer_id	vendor_id	OpeningTime	vendor_tag_name	Open	Close	afternoc
0	TCHWPBT	113	10:59AM-10:59PM	arabic,desserts,free delivery,indian	10:59AM	10:59PM	1
1	TCHWPBT	237	08:30PM-11:59PM	american,burgers,desserts,donuts,fries,pasta,s...	08:30PM	11:59PM	0
2	ZGFSYCZ	4	11:00AM-11:30PM	arabic,breakfast,burgers,desserts,free deliver...	11:00AM	11:30PM	1

4. Модель, основанная на рейтинге

4.1 Совместная фильтрация (CF)

4.1.1 Создание Полной Матрицы

Извлекаем требуемую переменную для CF:

```
# Извлекаем требуемую переменную для CF:
cus_ven_ratings = df1_train_for_anal[['customer_id', 'vendor_id', 'rating']]
cus_ven_ratings
```

	customer_id	vendor_id	rating
0	TCHWPBT	113	5.0
1	TCHWPBT	237	NaN
2	ZGFSYCZ	4	NaN
3	ZGFSYCZ	28	NaN
4	ZGFSYCZ	28	NaN
...	...	...	...
104544	U2OTA4O	573	NaN
104545	Z7RQ368	160	5.0
104546	WIIU12E	573	NaN
104547	LE63M0S	84	NaN
104548	8PSO92C	237	NaN

104549 rows × 3 columns

```
# Рассчитаем средний рейтинг только по действительным
# рейтингам (за исключением отсутствующих и нулевых оценок)
ratings_not_none = []

for i in range(0, cus_ven_ratings.shape[0]-1) :
  if pd.isnull(cus_ven_ratings.iloc[i][2]) == False and cus_ven_ratings.iloc[i]
    ratings_not_none.append(cus_ven_ratings.iloc[i][2])

valid_rating_mean = np.mean(np.array(ratings_not_none))
```

```
# Заменяем отсутствующий и нулевой рейтинг на действительный рейтинг
def rating_missing_func(x) :
  if pd.isnull(x) == True :
    return valid_rating_mean
  elif x == 0 :
    return valid_rating_mean
  else :
    return x

cus_ven_ratings["rating2"] = cus_ven_ratings["rating"].apply(rating_missing_fun
cus_ven_ratings
```

	customer_id	vendor_id	rating	rating2
0	TCHWPBT	113	5.0	5.000000
1	TCHWPBT	237	NaN	4.435256
2	ZGFSYCZ	4	NaN	4.435256
3	ZGFSYCZ	28	NaN	4.435256
4	ZGFSYCZ	28	NaN	4.435256
...	...	...	...	...
104544	U2OTA4O	573	NaN	4.435256
104545	Z7RQ368	160	5.0	5.000000
104546	WIIU12E	573	NaN	4.435256
104547	LE63M0S	84	NaN	4.435256
104548	8PSO92C	237	NaN	4.435256

104549 rows × 4 columns

```
# Реорганизация фрейма данных (переименование столбцов)
cus_ven_ratings = cus_ven_ratings[['customer_id', 'vendor_id', 'rating2']]
cus_ven_ratings.rename(columns={'rating2':'rating', 1:'customer_id_num'}, inpla
cus_ven_ratings
```

	customer_id	vendor_id	rating
0	TCHWPBT	113	5.000000
1	TCHWPBT	237	4.435256
2	ZGFSYCZ	4	4.435256
3	ZGFSYCZ	28	4.435256
4	ZGFSYCZ	28	4.435256
...	...	...	...
104544	U2OTA4O	573	4.435256
104545	Z7RQ368	160	5.000000
104546	WIIU12E	573	4.435256
104547	LE63M0S	84	4.435256
104548	8PSO92C	237	4.435256

104549 rows × 3 columns

```
# Интеграция в индивидуальные рейтинги по группам означает
cus_ven_ratings_mean = cus_ven_ratings.groupby(['customer_id', 'vendor_id']).me
cus_ven_ratings_mean
```

		rating
customer_id	**vendor_id**	
000THBA	**148**	4.435256
005ECL6	**237**	4.435256
009UFS1	**83**	4.435256
	84	4.435256
	193	4.435256
...	**...**	...
ZZY3N0D	**225**	3.000000
	356	4.435256
	419	4.435256
	459	5.000000
	537	5.000000

54144 rows × 1 columns

```
df_cus_ven_ratings_mean = cus_ven_ratings_mean.reset_index()
df_cus_ven_ratings_mean
```

	customer_id	vendor_id	rating
0	000THBA	148	4.435256
1	005ECL6	237	4.435256
2	009UFS1	83	4.435256
3	009UFS1	84	4.435256
4	009UFS1	193	4.435256
...	...	...	...
54139	ZZY3N0D	225	3.000000
54140	ZZY3N0D	356	4.435256
54141	ZZY3N0D	419	4.435256
54142	ZZY3N0D	459	5.000000
54143	ZZY3N0D	537	5.000000

54144 rows × 3 columns

```
# Создание Полной матрицы (Разреженная матрица)
rating_full_matrix = df_cus_ven_ratings_mean.pivot(index='customer_id', columns
rating_full_matrix
```

vendor_id	4	13	20	23	28	33	43	44	55	66	...	681	841	843	845	846
customer_id																
000THBA	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
005ECL6	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
009UFS1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
00F8I3F	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
00FQ1U9	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
ZZP7JCZ	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
ZZV76GY	NaN	NaN	NaN	NaN	4.435256	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
ZZVLIB5	NaN	NaN	NaN	4.435256	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
ZZWKMGG	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
ZZY3N0D	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN

18550 rows × 100 columns

```
# Вычисляем сходство всей пары клиентов по Полной матрице
from sklearn.metrics.pairwise import cosine_similarity
rating_matrix_dummy = rating_full_matrix.copy().fillna(0)

customer_similarity = cosine_similarity(rating_matrix_dummy, rating_matrix_dumm

customer_similarity = pd.DataFrame(customer_similarity, index = rating_full_mat
customer_similarity
```

customer_id	000THBA	005ECL6	009UFS1	00F8I3F	00FQ1U9	00GV4J4	00HWUU3	00M7NA5	00OT8JX	00Q(
customer_id										
000THBA	1.0	0.0	0.0	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	
005ECL6	0.0	1.0	0.0	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	
009UFS1	0.0	0.0	1.0	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	
00F8I3F	0.0	0.0	0.0	1.000000	0.000000	0.000000	0.000000	0.0	0.257810	
00FQ1U9	0.0	0.0	0.0	0.000000	1.000000	0.000000	0.000000	0.0	0.000000	
...	...	...	...	...	...	...	...	...	...	
ZZP7JCZ	0.0	0.0	0.0	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	
ZZV76GY	0.0	0.0	0.0	0.353553	0.000000	0.204124	0.000000	0.0	0.344008	
ZZVLIB5	0.0	0.0	0.0	0.000000	0.000000	0.000000	0.000000	0.0	0.114345	
ZZWKMGG	0.0	0.0	0.0	0.000000	0.000000	0.000000	0.000000	0.0	0.228691	
ZZY3N0D	0.0	0.0	0.0	0.000000	0.347731	0.200763	0.295403	0.0	0.000000	

18550 rows × 18550 columns

4.1.2 Создание Модель Совместной Фильтраций

```
# Функция, которая вычисляет точность (Среднеквадратичная ошибка)
def RMSE(y_true, y_pred):
    return np.sqrt(np.mean((np.array(y_true) - np.array(y_pred))**2))
```

```
# Функция, которая применяет RMSE к модели CF
def knn_score(model, neigbor_size=0) :
  id_pairs = zip(df_cus_ven_ratings_mean['customer_id'], df_cus_ven_ratings_mea
  y_pred = np.array([model(customer, vendor, neigbor_size) for (customer, vendo
  y_true = np.array(df_cus_ven_ratings_mean['rating'])
  return RMSE(y_true, y_pred)
```

```
# Модель CF (ограничение количества соседних размеров)
def cf_knn(customer_id, vendor_id, neighbor_size=0):
    if vendor_id in rating_full_matrix:
        # Сходство введенного клиента и другого клиента
        sim_scores = customer_similarity[customer_id].copy()
        # Оценки всех клиентов для введенного поставщика (ресторана)
        vendor_ratings = rating_full_matrix[vendor_id].copy()
        # Индекс клиентов, которые не являются поставщиком, введенным по ставке
        none_rating_idx = vendor_ratings[vendor_ratings.isnull()].index
        # Рейтинг исключения (нулевой) кто из клиентов, которые не
        # оценивают введенного поставщика
        vendor_ratings = vendor_ratings.drop(none_rating_idx)
        # Исключение: кто из клиентов, которые не оценивают введенного
        # поставщика
        sim_scores = sim_scores.drop(none_rating_idx)
```

```
        if neighbor_size == 0:
            # Средневзвешенное значение оценок клиентов, которые
            # оценивают введенного поставщика
            mean_rating = np.dot(sim_scores, vendor_ratings) / sim_scores.sum()

        # Случай, когда указан размер соседа
        else:
            # Случай, когда 2 или более человек оценивают введенного
            # поставщика
            if len(sim_scores) > 1:
                # Минимальное значение среди введенных размеров соседей
                # и количества клиентов, которые оценивают
                # введенного поставщика
                neighbor_size = min(neighbor_size, len(sim_scores))
                # транспонировать в массив Numpy для использования argsort
                sim_scores = np.array(sim_scores)
                vendor_ratings = np.array(vendor_ratings)
                # Сортировочное сходство
                customer_idx = np.argsort(sim_scores)
                # Сходство такое же, как и размер соседа
                sim_scores = sim_scores[customer_idx][-neighbor_size:]
                # Рейтинги такие же, как и размер соседа
                vendor_ratings = vendor_ratings[customer_idx][-neighbor_size:]
                # Рассчитать окончательные прогнозируемые разглагольствования
                mean_rating = np.dot(sim_scores, vendor_ratings) / sim_scores.s
            else:
                # Заменить действительным значением в другом случае
                mean_rating = valid_rating_mean
    else:
        # Заменить действительным значением в другом случае
        mean_rating = valid_rating_mean
    return mean_rating
```

```
knn_score(cf_knn, neigbor_size=20)
```

```
0.3865892604982157
```

4.1.3 Рекомендация для Клиента на основе CF

```
# Функция, которая представляет
# список рекомендаций для определенного клиента по CF
def cf_recom_vendor(customer_id, n_items, neighbor_size=0):
    # Поставщики, которые были оценены введенным клиентом
    customer_vendor = rating_full_matrix.loc[customer_id].copy()

    for vendor in rating_full_matrix:
        # Исключение поставщики, которые уже оценены введенным клиентом
        if pd.notnull(customer_vendor.loc[vendor]):
            customer_vendor.loc[vendor] = 0
        # Рассчитать прогнозируемый рейтинг поставщиков,
        # который не оценен введенным клиентом
        else:
            customer_vendor.loc[vendor] = cf_knn(customer_id, vendor, neighbor_

    # Сортировка поставщиков по прогнозируемому рейтингу
    vendor_sort = customer_vendor.sort_values(ascending=False)[:n_items]
    recom_vendors_temp = df1_train_for_anal.loc[vendor_sort.index]
    recom_vendors_temp2 = recom_vendors_temp[['vendor_id', 'mean_rating', 'vend
    recom_vendors = recom_vendors_temp2.reset_index(drop=True)
    return recom_vendors
```

```
# Пример списка рекомендаций
cf_recom_vendor(customer_id='ZZV76GY', n_items=5, neighbor_size=30)
```

	vendor_id	mean_rating	vendor_tag_name
0	288	4.6	Asian,Desserts,Rice,Salads,Soups,Thai
1	577	4.5	Burgers,Desserts,Family Meal,Salads
2	92	4.6	Asian,Fresh Juices,Kids meal
3	92	4.6	Asian,Fresh Juices,Kids meal
4	676	4.4	Biryani,Desserts,Indian,Kebabs,Rice

4.2 Матричная факторизация с глубоким обучением (MF с DL)

4.2.1 Создание полной матрицы с индексом

Для MF с DL необходима полная матрица, состоящая из непрерывного числового имени столбца и имени строки.

```
# Извлечение требуемой переменной для MF
ratings = cus_ven_ratings
```

```
# Интеграция в индивидуальные рейтинги по группам означает
ratings = ratings.groupby(['customer_id', 'vendor_id']).mean().reset_index()
```

```
# Создаем полную матрицу для временной предварительной обработки
R_temp = ratings.pivot(index='customer_id', columns='vendor_id', values='rating
R_temp
```

vendor_id	4	13	20	23	28	33	43	44	55	66	...	681	841	843	845	846	849	855	8
customer_id																			
000THBA	0.0	0.0	0.0	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	(
005ECL6	0.0	0.0	0.0	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	(
009UFS1	0.0	0.0	0.0	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	(
00F8I3F	0.0	0.0	0.0	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	(
00FQ1U9	0.0	0.0	0.0	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	(
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
ZZP7JCZ	0.0	0.0	0.0	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	(
ZZV76GY	0.0	0.0	0.0	0.000000	4.435256	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	(
ZZVLIB5	0.0	0.0	0.0	4.435256	0.000000	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	(
ZZWKMGG	0.0	0.0	0.0	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	(
ZZY3N0D	0.0	0.0	0.0	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	(

18550 rows × 100 columns

```
# Сопоставление идентификаторов клиентов
# с индексом (непрерывное числовое имя строки)
customer_id_index = []

for i, one_id in enumerate(R_temp.T) :
  customer_id_index.append([one_id, i])
```

```
df_customer_id_index = pd.DataFrame(customer_id_index)
df_customer_id_index.rename(columns={0:'customer_id', 1:'customer_idx'}, inplac
df_customer_id_index
```

	customer_id	customer_idx
0	000THBA	0
1	005ECL6	1
2	009UFS1	2
3	00F8I3F	3
4	00FQ1U9	4
...	...	...
18545	ZZP7JCZ	18545
18546	ZZV76GY	18546
18547	ZZVLIB5	18547
18548	ZZWKMGG	18548
18549	ZZY3N0D	18549

18550 rows × 2 columns

```
# Сопоставление идентификаторов
# поставщиков с индексом (непрерывное числовое имя столбца)
vendor_id_index = []

for i, one_id in enumerate(R_temp) :
  vendor_id_index.append([one_id, i])
```

```
df_vendor_id_index = pd.DataFrame(vendor_id_index)
df_vendor_id_index.rename(columns={0:'vendor_id', 1:'vendor_idx'}, inplace=True
df_vendor_id_index
```

	vendor_id	vendor_idx
0	4	0
1	13	1
2	20	2
3	23	3
4	28	4
...	...	...
95	849	95
96	855	96
97	856	97
98	858	98
99	907	99

100 rows × 2 columns

```
# Объединим рейтинг и каждый индекс
ratings_with_index = pd.merge(ratings, df_customer_id_index, on='customer_id')
ratings_with_index = pd.merge(ratings_with_index, df_vendor_id_index, on='vendo
ratings = ratings_with_index[['customer_idx', 'vendor_idx', 'rating']].astype(i
ratings.rename(columns={'customer_id_num':'customer_idx', 'vendor_id_num':'vend
ratings
```

	customer_idx	vendor_idx	rating
0	0	31	4
1	16	31	4
2	36	31	4
3	45	31	4
4	46	31	4
...	...	...	...
54139	18142	96	4
54140	18203	96	4
54141	18270	96	4
54142	18395	96	4
54143	18449	96	5

54144 rows × 3 columns

```
# Создание Полной матрицы (Разреженная матрица)
rating_full_matrix_by_index_with_nan = ratings.pivot(index='customer_idx', colu
rating_full_matrix_by_index_with_nan
```

vendor_idx	0	1	2	3	4	5	6	7	8	9	...	90	91	92	93	94	95
customer_idx																	
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
18545	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN
18546	NaN	NaN	NaN	NaN	4.0	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN
18547	NaN	NaN	NaN	4.0	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN
18548	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN
18549	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN

18550 rows × 100 columns

```
# Транспонировать None в ноль
rating_full_matrix_by_index = ratings.pivot(index='customer_idx', columns='vend
rating_full_matrix_by_index
```

vendor_idx	0	1	2	3	4	5	6	7	8	9	...	90	91	92	93	94	95	96	97	98	99
customer_idx																					
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
18545	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
18546	0.0	0.0	0.0	0.0	4.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
18547	0.0	0.0	0.0	4.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
18548	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
18549	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

18550 rows × 100 columns

4.2.2 Создание модели MF с DL

```
# Количество скрытых факторов
K = 100
# Общее среднее
mu = ratings.rating.mean()
# Количество клиентов
M = ratings.customer_idx.unique().shape[0]
# Количество поставщиков
N = ratings.vendor_idx.unique().shape[0]

# Функция расчета RMSE
def RMSE(y_true, y_pred):
    return tf.sqrt(tf.reduce_mean(tf.square(y_true - y_pred)))

# Embedding для модели Keras
customer = Input(shape=(1, ))
vendor = Input(shape=(1, ))
P_embedding = Embedding(M, K, embeddings_regularizer=l2())(customer)
Q_embedding = Embedding(N, K, embeddings_regularizer=l2())(vendor)
customer_bias = Embedding(M, 1, embeddings_regularizer=l2())(customer)
vendor_bias = Embedding(N, 1, embeddings_regularizer=l2())(vendor)

# Слои
from tensorflow.keras.layers import Dense, Concatenate, Activation
P_embedding = Flatten()(P_embedding)
Q_embedding = Flatten()(Q_embedding)
customer_bias = Flatten()(customer_bias)
vendor_bias = Flatten()(vendor_bias)
R = Concatenate()([P_embedding, Q_embedding, customer_bias, vendor_bias])
```

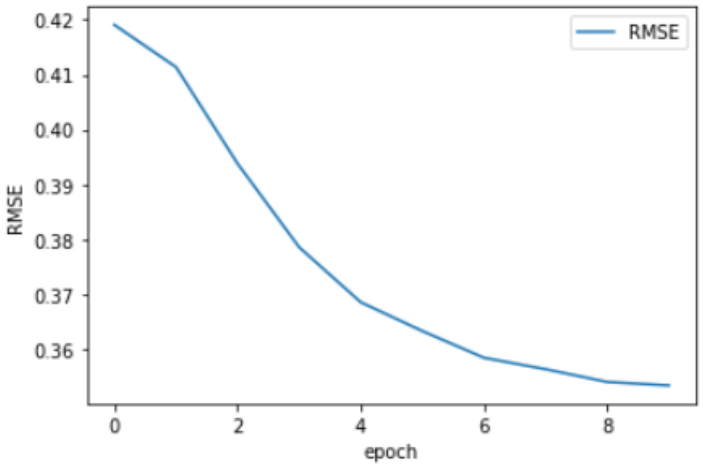
```
# Нейронная сеть
R = Dense(2048)(R)
R = Activation('linear')(R)
R = Dense(256)(R)
R = Activation('linear')(R)
R = Dense(1)(R)

# Компиляция модели
model = Model(inputs=[customer, vendor], outputs=R)
model.compile(
  loss=RMSE,
  optimizer=Adamax(),
  metrics=[RMSE]
)
model.summary()
```

```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 1)]          0
__________________________________________________________________________________________________
input_2 (InputLayer)            [(None, 1)]          0
__________________________________________________________________________________________________
embedding (Embedding)           (None, 1, 100)       1855000     input_1[0][0]
__________________________________________________________________________________________________
embedding_1 (Embedding)         (None, 1, 100)       10000       input_2[0][0]
__________________________________________________________________________________________________
embedding_2 (Embedding)         (None, 1, 1)         18550       input_1[0][0]
__________________________________________________________________________________________________
embedding_3 (Embedding)         (None, 1, 1)         100         input_2[0][0]
__________________________________________________________________________________________________
flatten (Flatten)               (None, 100)          0           embedding[0][0]
__________________________________________________________________________________________________
flatten_1 (Flatten)             (None, 100)          0           embedding_1[0][0]
__________________________________________________________________________________________________
flatten_2 (Flatten)             (None, 1)            0           embedding_2[0][0]
__________________________________________________________________________________________________
flatten_3 (Flatten)             (None, 1)            0           embedding_3[0][0]
__________________________________________________________________________________________________
concatenate (Concatenate)       (None, 202)          0           flatten[0][0]
                                                                 flatten_1[0][0]
                                                                 flatten_2[0][0]
                                                                 flatten_3[0][0]
__________________________________________________________________________________________________
dense (Dense)                   (None, 2048)         415744      concatenate[0][0]
__________________________________________________________________________________________________
activation (Activation)         (None, 2048)         0           dense[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 256)          524544      activation[0][0]
__________________________________________________________________________________________________
activation_1 (Activation)       (None, 256)          0           dense_1[0][0]
__________________________________________________________________________________________________
dense_2 (Dense)                 (None, 1)            257         activation_1[0][0]
==================================================================================================
Total params: 2,824,195
Trainable params: 2,824,195
Non-trainable params: 0
__________________________________________________________________________________________________
```

```
# Обучение модели
result = model.fit(
  x=[ratings.customer_idx.values, ratings.vendor_idx.values],
  y=ratings.rating.values - mu,
  epochs=10,
  batch_size=512,
  validation_data=(
    [ratings.customer_idx.values, ratings.vendor_idx.values],
    ratings.rating.values - mu
  )
)
```

```
Epoch 1/10
106/106 [==============================] - 8s 66ms/step - loss: 6.4020 - RMSE: 0.4284 - val_los
s: 0.4431 - val_RMSE: 0.4020
Epoch 2/10
106/106 [==============================] - 6s 55ms/step - loss: 0.4373 - RMSE: 0.4118 - val_los
s: 0.3867 - val_RMSE: 0.3755
Epoch 3/10
106/106 [==============================] - 6s 56ms/step - loss: 0.4023 - RMSE: 0.3909 - val_los
s: 0.3684 - val_RMSE: 0.3577
Epoch 4/10
106/106 [==============================] - 6s 53ms/step - loss: 0.3913 - RMSE: 0.3800 - val_los
s: 0.3599 - val_RMSE: 0.3484
Epoch 5/10
106/106 [==============================] - 5s 51ms/step - loss: 0.3664 - RMSE: 0.3546 - val_los
s: 0.3515 - val_RMSE: 0.3394
Epoch 6/10
106/106 [==============================] - 6s 52ms/step - loss: 0.3658 - RMSE: 0.3534 - val_los
s: 0.3474 - val_RMSE: 0.3349
Epoch 7/10
106/106 [==============================] - 6s 53ms/step - loss: 0.3582 - RMSE: 0.3453 - val_los
s: 0.3460 - val_RMSE: 0.3330
Epoch 8/10
106/106 [==============================] - 5s 51ms/step - loss: 0.3631 - RMSE: 0.3500 - val_los
s: 0.3429 - val_RMSE: 0.3296
Epoch 9/10
106/106 [==============================] - 6s 54ms/step - loss: 0.3644 - RMSE: 0.3510 - val_los
s: 0.3402 - val_RMSE: 0.3268
Epoch 10/10
106/106 [==============================] - 5s 51ms/step - loss: 0.3557 - RMSE: 0.3422 - val_los
s: 0.3420 - val_RMSE: 0.3285
```

```
# Дипграмма RMSE
import matplotlib.pyplot as plt
plt.plot(result.history['RMSE'], label="RMSE")
plt.xlabel('epoch')
plt.ylabel('RMSE')
plt.legend()
plt.show()
```




```
# Сравнение фактического и прогнозируемого рейтинга
customer_ids = ratings.customer_idx.values[0:6]
vendor_ids = ratings.vendor_idx.values[0:6]
predictions = model.predict([customer_ids, vendor_ids]) + mu
print("Actuals: \n", ratings[0:6])
print()
print("Predictions: \n", predictions)
```

```
Actuals:
    customer_idx  vendor_idx  rating
0              0          31       4
1             16          31       4
2             36          31       4
3             45          31       4
4             46          31       4
5            120          31       4

Predictions:
 [[3.9837234]
 [4.198856 ]
 [4.0481825]
 [3.8934042]
 [4.0173125]
 [3.9866211]]
```

4.2.3 Рекомендация для Клиента на основе MF

```
def recom_vendor(customer_idx, n_items):
    # Поставщики, которые были оценены введенным клиентом
    customer_vendor = rating_full_matrix_by_index_with_nan.loc[customer_idx].co

    for vendor in rating_full_matrix_by_index_with_nan:
        # Исключение поставщики, которые уже оценены введенным клиентом
        if pd.notnull(customer_vendor.loc[vendor]):
            customer_vendor.loc[vendor] = 0
        # Рассчитать прогнозируемый рейтинг поставщиков,
        # который не оценен введенным клиентом
        else:
            customer_vendor.loc[vendor] = round(min(model.predict([np.array([cu

    # Сортировка поставщиков по прогнозируемому рейтингу
    vendor_sort = customer_vendor.sort_values(ascending=False)[:n_items]
    df_vendor_sort = pd.DataFrame(vendor_sort)
    df_vendor_sort.rename(columns={'vendor_idx':'vendor_idx', customer_idx:'pre

    return df_vendor_sort
```

```
# Переназначение идентификаторов клиентов и индекса клиентов
customer_id_idx = ratings_with_index[['customer_id', 'customer_idx']]
customer_id_idx = customer_id_idx.drop_duplicates()
```

```
# Переназначение идентификаторов поставщиков и индекса поставщиков
vendor_id_idx = ratings_with_index[['vendor_id', 'vendor_idx']]
vendor_id_idx = vendor_id_idx.drop_duplicates()
```

```
# Подготовка фрейма данных для извлечения тега поставщика
mf_df1 = df1_train_for_anal[['vendor_id', 'mean_rating', 'vendor_tag_name']]
mf_df1 = mf_df1.drop_duplicates()
```

```
# Функция, которая представляет список рекомендаций
# для определенного клиента по MF с DL
def mf_dl_recom_vendor_list(customer_id, n_items) :
  df_specified_customer = customer_id_idx[customer_id_idx['customer_id']== cust
  specified_customer_idx = df_specified_customer.iloc[0][1]
  mf_recom_list_temp = recom_vendor(customer_idx = specified_customer_idx, n_it
  mf_recom_list_temp2 = pd.merge(mf_recom_list_temp, vendor_id_idx, how='inner'
  mf_recom_list_temp3 = mf_recom_list_temp2[['vendor_id', 'predicted_rating']]
  mf_recom_list = pd.merge(mf_recom_list_temp3, mf_df1, how='inner', on='vendor
  return mf_recom_list
```

```
# Пример списка рекомендаций
mf_dl_recom_vendor_list(customer_id = 'ZZV76GY', n_items = 5)
```

	vendor_id	predicted_rating	mean_rating	vendor_tag_name
0	386	4.045	4.5	Churros
1	86	4.036	4.5	Cakes,Crepes,Desserts,Donuts,Fresh Juices,Ice ...
2	216	4.030	4.7	Coffee,Organic
3	310	4.029	4.8	Bagels,Desserts,Salads
4	679	4.024	4.5	Biryani,Desserts,Indian,Kebabs,Rice

5. Модель, основанная на содержании

```
df1_contents_for_anal=train_contents[:]
```

```
df1_contents_for_anal.head()
```

	customer_id	vendor_id	OpeningTime	vendor_tag_name	Open	Close	afternoc
0	TCHWPBT	113	10:59AM-10:59PM	arabic,desserts,free delivery,indian	10:59AM	10:59PM	1
1	TCHWPBT	237	08:30PM-11:59PM	american,burgers,desserts,donuts,fries,pasta,s...	08:30PM	11:59PM	0
2	ZGFSYCZ	4	11:00AM-11:30PM	arabic,breakfast,burgers,desserts,free deliver...	11:00AM	11:30PM	1
3	ZGFSYCZ	28	11:00AM-11:45PM	burgers	11:00AM	11:45PM	1
4	ZGFSYCZ	28	11:00AM-11:45PM	burgers	11:00AM	11:45PM	1

```
df1_contents_for_anal['vendor_tag']= df1_contents_for_anal['vendor_tag_name'].s
df1_contents_for_anal['vendor_tag']=df1_contents_for_anal['vendor_tag'].apply(l
```

```
# Проверим похожее слово
def similar(a, b):
    ratio=SequenceMatcher(None, a, b).ratio()
    return print("Similarity of {} and {}  : {}".format(a,b,ratio) )

similar('pasta','pastas')
similar('pasta','pastry')
similar('pizza','pizzas')
similar('soups','shuwa')
similar('shawarma','shuwa')
similar('thali','thai')
similar('milkshakes','mishkak')

# изменить слова, когда сходство превышает 0,8
df1_contents_for_anal['vendor_tag']=df1_contents_for_anal['vendor_tag'].apply(l
df1_contents_for_anal['vendor_tag']=df1_contents_for_anal['vendor_tag'].apply(l
df1_contents_for_anal['vendor_tag']=df1_contents_for_anal['vendor_tag'].apply(l

df1_contents_for_anal['vendor_tag1']=df1_contents_for_anal['vendor_tag'].apply(
```

```
Similarity of pasta and pastas  : 0.9090909090909091
Similarity of pasta and pastry  : 0.7272727272727273
Similarity of pizza and pizzas  : 0.9090909090909091
Similarity of soups and shuwa  : 0.4
Similarity of shawarma and shuwa  : 0.6153846153846154
Similarity of thali and thai  : 0.8888888888888888
Similarity of milkshakes and mishkak  : 0.7058823529411765
```

```
df1_contents_for_anal.head()
```

	customer_id	vendor_id	OpeningTime	vendor_tag_name	Open	Close	afternoc
0	TCHWPBT	113	10:59AM-10:59PM	arabic,desserts,free delivery,indian	10:59AM	10:59PM	1
1	TCHWPBT	237	08:30PM-11:59PM	american,burgers,desserts,donuts,fries,pasta,s...	08:30PM	11:59PM	0
2	ZGFSYCZ	4	11:00AM-11:30PM	arabic,breakfast,burgers,desserts,free deliver...	11:00AM	11:30PM	1
3	ZGFSYCZ	28	11:00AM-11:45PM	burgers	11:00AM	11:45PM	1
4	ZGFSYCZ	28	11:00AM-11:45PM	burgers	11:00AM	11:45PM	1

```
df1_contents_for_anal['vendor_id'].value_counts()
```

```
113    5792
105    4822
84     4163
79     4046
78     3688
       ...
295     390
582     387
841     379
845     365
843     339
Name: vendor_id, Length: 95, dtype: int64
```

```
prac= df1_contents_for_anal.drop_duplicates("vendor_id", keep="first", inplace=
print(prac.shape)

prac['vendor_id']=prac['vendor_id'].astype(str)
prac1=prac[:]
```

```
(95, 11)
```

5.1 TF-IDF

5.1.1 Модель на основе TF-IDF

```
prac.set_index('vendor_id',inplace=True)
prac.head(2)
```

vendor_id	customer_id	OpeningTime	vendor_tag_name	Open	Close	afternoon
113	TCHWPBT	10:59AM-10:59PM	arabic,desserts,free delivery,indian	10:59AM	10:59PM	1.0
237	TCHWPBT	08:30PM-11:59PM	american,burgers,desserts,donuts,fries,pasta,s...	08:30PM	11:59PM	0.0

```
vectorizer = TfidfVectorizer()
count_matrix = vectorizer.fit_transform(prac['vendor_tag1'])
print(vectorizer.get_feature_names())
print(vectorizer.vocabulary_)
```

```
['american', 'arabic', 'asian', 'bagels', 'biryani', 'breakfast', 'burgers', 'cafe', 'cakes',
'chinese', 'churros', 'coffee', 'combos', 'crepes', 'desserts', 'dimsum', 'donuts', 'familymea
l', 'freedelivery', 'freshjuices', 'fries', 'frozenyoghurt', 'grills', 'healthyfood', 'hotchoco
late', 'hotdogs', 'icecreams', 'indian', 'italian', 'japanese', 'karak', 'kebabs', 'kidsmeal',
'kushari', 'lebanese', 'manakeesh', 'mandazi', 'mexican', 'milkshakes', 'mishkak', 'mojitos',
'omani', 'organic', 'pancakes', 'pasta', 'pastry', 'pizza', 'rice', 'rolls', 'salads', 'sandwic
hes', 'seafood', 'shawarma', 'shuwa', 'smoothies', 'soups', 'spanishlatte', 'steaks', 'sushi',
'sweets', 'thai', 'vegetarian', 'waffles']
{'arabic': 1, 'desserts': 14, 'freedelivery': 18, 'indian': 27, 'american': 0, 'burgers': 6, 'd
onuts': 16, 'fries': 20, 'pasta': 44, 'salads': 49, 'sandwiches': 50, 'breakfast': 5, 'grills':
22, 'lebanese': 34, 'shawarma': 52, 'mexican': 37, 'asian': 2, 'healthyfood': 23, 'japanese': 2
9, 'sushi': 58, 'hotdogs': 25, 'biryani': 4, 'soups': 55, 'freshjuices': 19, 'smoothies': 54,
'cakes': 8, 'coffee': 11, 'hotchocolate': 24, 'bagels': 3, 'kidsmeal': 32, 'pizza': 46, 'family
meal': 17, 'kebabs': 31, 'rice': 47, 'cafe': 7, 'icecreams': 26, 'italian': 28, 'sweets': 59,
'thai': 60, 'vegetarian': 61, 'milkshakes': 38, 'mandazi': 36, 'omani': 41, 'steaks': 57, 'froz
enyoghurt': 21, 'mojitos': 40, 'mishkak': 39, 'organic': 42, 'manakeesh': 35, 'crepes': 13, 'pa
ncakes': 43, 'waffles': 62, 'chinese': 9, 'dimsum': 15, 'churros': 10, 'spanishlatte': 56, 'pas
try': 45, 'seafood': 51, 'rolls': 48, 'karak': 30, 'combos': 12, 'kushari': 33, 'shuwa': 53}
```

```
indices = pd.Series(prac.index)
indices[:5]
```

```
0    113
1    237
2      4
3     28
4     33
Name: vendor_id, dtype: object
```

```
count_matrix
```

```
<95x63 sparse matrix of type '<class 'numpy.float64'>'
        with 512 stored elements in Compressed Sparse Row format>
```

```
cosine_sim = cosine_similarity(count_matrix,count_matrix)
print(cosine_sim)
```

```
[[1.         0.08513408 0.41289618 ... 0.18465308 0.05375778 0.        ]
 [0.08513408 1.         0.24529403 ... 0.10612441 0.24322414 0.44340833]
 [0.41289618 0.24529403 1.         ... 0.31239162 0.03432689 0.08561252]
 ...
 [0.18465308 0.10612441 0.31239162 ... 1.         0.         0.        ]
 [0.05375778 0.24322414 0.03432689 ... 0.         1.         0.16505101]
 [0.         0.44340833 0.08561252 ... 0.         0.16505101 1.        ]]
```

```
indices[indices == '113'].index[0]
```

```
0
```

```
list(enumerate(cosine_sim[0]))
```

```
[(0, 1.0),
 (1, 0.08513408044253058),
 (2, 0.4128961795437884),
 (3, 0.0),
 (4, 0.1309992010126455),
 (5, 0.06610750732821946),
 (6, 0.1788203692335047),
 (7, 0.0),
 (8, 0.42208145319629464),
 (9, 0.0),
 (10, 0.0),
 (11, 0.20346251333003257),
 (12, 0.0),
 (13, 0.0),
 (14, 0.10655496189641896),
 (15, 0.0),
 (16, 0.101008705188368),
 (17, 0.0),
 (18, 0.33378443298095645),
 (19, 0.2076225427154327),
 (20, 0.2076225427154327),
 (21, 0.2076225427154327),
 (22, 0.08056164337775902),
 (23, 0.34796626367696515),
 (24, 0.26130167815168226),
 (25, 0.2790479119241957),
 (26, 0.0),
 (27, 0.09029950255274545),
 (28, 0.09029950255274545),
 (29, 0.29028767119522747),
 (30, 0.5015331801967973),
 (31, 0.1309992010126455),
 (32, 0.0),
 (33, 0.0),
 (34, 0.0),
 (35, 0.2076225427154327),
 (36, 0.7180609886155626),
 (37, 0.1788203692335047),
 (38, 0.0),
 (39, 0.0),
 (40, 0.0),
 (41, 0.0),
 (42, 0.4813164228848414),
 (43, 0.0),
 (44, 0.0),
 (45, 0.21708891478155332),
 (46, 0.0),
 (47, 0.0),
 (48, 0.0),
 (49, 0.076780746313055),
 (50, 0.0),
 (51, 0.0741422585077138),
 (52, 0.07306952826593163),
 (53, 0.05584803467074938),
 (54, 0.0),
 (55, 0.0),
 (56, 0.354010015423017),
 (57, 0.2076225427154327),
 (58, 0.1788203692335047),
 (59, 0.0),
 (60, 0.0),
 (61, 0.5781243366476634),
 (62, 0.0),
 (63, 0.33378443298095645),
 (64, 0.09029950255274545),
 (65, 0.0),
 (66, 0.24397699907664688),
 (67, 0.34796626367696515),
 (68, 0.0),
 (69, 0.0),
 (70, 0.33893478138175154),
```

```
(71, 0.09285144822230335),
(72, 0.14197215252871911),
(73, 0.3865872125139497),
(74, 0.0),
(75, 0.0),
(76, 0.0),
(77, 0.112473523050599),
(78, 0.3675701166839126),
(79, 0.0),
(80, 0.12134856524357193),
(81, 0.0),
(82, 0.33378443298095645),
(83, 0.2653640284495341),
(84, 0.20315472182128905),
(85, 0.0),
(86, 0.2076225427154327),
(87, 0.20413248729170183),
(88, 0.29448253399641916),
(89, 0.0),
(90, 0.0),
(91, 0.2076225427154327),
(92, 0.1846530825642572),
(93, 0.05375777654751215),
(94, 0.0)]
```

```
prac=prac.reset_index()

def get_recommendations(id, cosine_sim=cosine_sim):
    indices = pd.Series(prac.index, index = prac['vendor_id']).drop_duplicates(
    # получим индекс от vendor_id
    idx = indices[id]

    # cosin_similarity
    sim_scores = list(enumerate(cosine_sim[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    # наиболее похожий vendor_id (10)
    sim_scores = sim_scores[1:11]

    return sim_scores
```

```
get_recommendations('113')
```

```
[(36, 0.7180609886155626),
 (61, 0.5781243366476634),
 (30, 0.5015331801967973),
 (42, 0.4813164228848414),
 (8, 0.42208145319629464),
 (2, 0.4128961795437884),
 (73, 0.3865872125139497),
 (78, 0.3675701166839126),
 (56, 0.354010015423017),
 (23, 0.34796626367696515)]
```

The vendor id '113' is most similar with '36' (cosine similarity is about 0.71)

5.1.2 Добавление метки времени (утро/день/вечер)

```
cols=['afternoon','evening','morning']

prac['time']='' # создаем пустой столбец
for col_name in cols:
    prac.loc[prac[col_name]==1,'time']= prac['time']+' '+col_name

prac=prac[['vendor_id','customer_id','vendor_tag','vendor_tag1','time']]
prac.head()
```

	vendor_id	customer_id	vendor_tag	vendor_tag1	time
0	113	TCHWPBT	[arabic, desserts, freedelivery, indian]	arabic desserts freedelivery indian	afternoon evening morning
1	237	TCHWPBT	[american, burgers, desserts, donuts, fries, p...	american burgers desserts donuts fries pasta s...	evening
2	4	ZGFSYCZ	[arabic, breakfast, burgers, desserts, freedel...	arabic breakfast burgers desserts freedelivery...	afternoon evening
3	28	ZGFSYCZ	[burgers]	burgers	afternoon evening
4	33	ZGFSYCZ	[desserts, mexican]	desserts mexican	afternoon evening

```
#'time' column : str -> list
prac['time']=prac['time'].str.split(' ')

def remove_blank(lists):
  return [key for key in lists if key !='']

prac['time']=prac['time'].apply(lambda x:remove_blank(x))
prac['time'][0]
```

```
['afternoon', 'evening', 'morning']
```

```
# объединим "vendor_tag" с "временем" (str для w2v)
prac['time1']=prac['time'].apply(lambda x:' '.join(x))
prac['time_tag']=prac[['vendor_tag1','time1']].apply(lambda x: ' '.join(x),axis

prac=prac[['vendor_id','customer_id','time_tag','vendor_tag','vendor_tag1']]
prac.head(2)
prac2=prac[:]
```

```
vectorizer = TfidfVectorizer()
count_matrix = vectorizer.fit_transform(prac['time_tag'])
print(vectorizer.get_feature_names())
print(vectorizer.vocabulary_)
```

```
['afternoon', 'american', 'arabic', 'asian', 'bagels', 'biryani', 'breakfast', 'burgers', 'caf
e', 'cakes', 'chinese', 'churros', 'coffee', 'combos', 'crepes', 'desserts', 'dimsum', 'donut
s', 'evening', 'familymeal', 'freedelivery', 'freshjuices', 'fries', 'frozenyoghurt', 'grills',
'healthyfood', 'hotchocolate', 'hotdogs', 'icecreams', 'indian', 'italian', 'japanese', 'kara
k', 'kebabs', 'kidsmeal', 'kushari', 'lebanese', 'manakeesh', 'mandazi', 'mexican', 'milkshake
s', 'mishkak', 'mojitos', 'morning', 'omani', 'organic', 'pancakes', 'pasta', 'pastry', 'pizz
a', 'rice', 'rolls', 'salads', 'sandwiches', 'seafood', 'shawarma', 'shuwa', 'smoothies', 'soup
s', 'spanishlatte', 'steaks', 'sushi', 'sweets', 'thai', 'vegetarian', 'waffles']
{'arabic': 2, 'desserts': 15, 'freedelivery': 20, 'indian': 29, 'afternoon': 0, 'evening': 18,
'morning': 43, 'american': 1, 'burgers': 7, 'donuts': 17, 'fries': 22, 'pasta': 47, 'salads': 5
2, 'sandwiches': 53, 'breakfast': 6, 'grills': 24, 'lebanese': 36, 'shawarma': 55, 'mexican': 3
9, 'asian': 3, 'healthyfood': 25, 'japanese': 31, 'sushi': 61, 'hotdogs': 27, 'biryani': 5, 'so
ups': 58, 'freshjuices': 21, 'smoothies': 57, 'cakes': 9, 'coffee': 12, 'hotchocolate': 26, 'ba
gels': 4, 'kidsmeal': 34, 'pizza': 49, 'familymeal': 19, 'kebabs': 33, 'rice': 50, 'cafe': 8,
'icecreams': 28, 'italian': 30, 'sweets': 62, 'thai': 63, 'vegetarian': 64, 'milkshakes': 40,
'mandazi': 38, 'omani': 44, 'steaks': 60, 'frozenyoghurt': 23, 'mojitos': 42, 'mishkak': 41, 'o
rganic': 45, 'manakeesh': 37, 'crepes': 14, 'pancakes': 46, 'waffles': 65, 'chinese': 10, 'dims
um': 16, 'churros': 11, 'spanishlatte': 59, 'pastry': 48, 'seafood': 54, 'rolls': 51, 'karak':
32, 'combos': 13, 'kushari': 35, 'shuwa': 56}
```

```python
prac.set_index('vendor_id',inplace=True)

indices = pd.Series(prac.index)
cosine_sim = cosine_similarity(count_matrix,count_matrix)
cosine_sim.shape
```

```
(95, 95)
```

```python
prac=prac.reset_index()

def item_recommendations(id, cosine_sim=cosine_sim):
    indices = pd.Series(prac.index, index = prac['vendor_id']).drop_duplicates(
    idx = indices[id]

    sim_scores = list(enumerate(cosine_sim[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:11]

    return sim_scores
```

```python
item_recommendations('113')
```

```
[(36, 0.7646218746937348),
 (61, 0.60208312512282),
 (30, 0.5282753498785006),
 (42, 0.47959440453508845),
 (2, 0.42700392534024345),
 (8, 0.4154209006850713),
 (18, 0.40946943842460437),
 (63, 0.40946943842460437),
 (82, 0.40946943842460437),
 (73, 0.3984131739369953)]
```

Идентификатор поставщика "113" наиболее похож на "36" (сходство по косинусу составляет около 0,76), поэтому мы видим, что сходство по косинусу немного улучшилось.

5.2 Doc2Vec

5.2.1 Word2Vec для Doc2Vec

```
prac1.head(2)
```

	customer_id	vendor_id	OpeningTime	vendor_tag_name	Open	Close	afternoc
0	TCHWPBT	113	10:59AM-10:59PM	arabic,desserts,free delivery,indian	10:59AM	10:59PM	1
1	TCHWPBT	237	08:30PM-11:59PM	american,burgers,desserts,donuts,fries,pasta,s...	08:30PM	11:59PM	0

```
prac1.set_index('vendor_id',inplace=True)
# Встраивание слова с помощью word2vec
corpus=prac1['vendor_tag']
model = Word2Vec(size=4, window=1, min_count=1, workers=4)
model.build_vocab(corpus)

word_vectors = model.wv
vocabs = word_vectors.vocab.keys()
vocabs
```

```
dict_keys(['arabic', 'desserts', 'freedelivery', 'indian', 'american', 'burgers', 'donuts', 'fr
ies', 'pasta', 'salads', 'sandwiches', 'breakfast', 'grills', 'lebanese', 'shawarma', 'mexica
n', 'asian', 'healthyfood', 'japanese', 'sushi', 'hotdogs', 'biryani', 'soups', 'freshjuices',
'smoothies', 'cakes', 'coffee', 'hotchocolate', 'bagels', 'kidsmeal', 'pizza', 'familymeal', 'k
ebabs', 'rice', 'cafe', 'icecreams', 'italian', 'sweets', 'thai', 'vegetarian', 'milkshakes',
'mandazi', 'omani', 'steaks', 'frozenyoghurt', 'mojitos', 'mishkak', 'organic', 'manakeesh', 'c
repes', 'pancakes', 'waffles', 'chinese', 'dimsum', 'churros', 'spanishlatte', 'pastry', 'seafo
od', 'rolls', 'karak', 'combos', 'kushari', 'shuwa'])
```

```
word_vectors.most_similar('breakfast')
```

```
[('kidsmeal', 0.9876297116279602),
 ('spanishlatte', 0.8556029200553894),
 ('seafood', 0.7407674789428711),
 ('dimsum', 0.6946371793746948),
 ('combos', 0.6774498224258423),
 ('mishkak', 0.6172083616256714),
 ('indian', 0.539692759513855),
 ('crepes', 0.42991745471954346),
 ('milkshakes', 0.4238659739494324),
 ('arabic', 0.4150455594062805)]
```

Эти слова похожи на слово "breakfast".

5.2.2 Doc2Vec

```
# Значение вектора слов
def vectors(document_list):
    document_embedding_list = []

    for line in document_list:
        doc2vec = None
        count = 0
        for word in line.split():
            if word in model.wv.vocab:
                count += 1
                if doc2vec is None:
                    doc2vec = model[word]
                else:
                    doc2vec = doc2vec + model[word]

        if doc2vec is not None:
            doc2vec = doc2vec / count
            document_embedding_list.append(doc2vec)
    return document_embedding_list

document_embedding_list = vectors(prac1['vendor_tag1'])
print('Number of document vector:',len(document_embedding_list))
```

```
Number of document vector: 95
```

```
cosine_sim = cosine_similarity(document_embedding_list, document_embedding_list

print('the shape of cosine similarity matrix :',cosine_sim.shape)
```

```
the shape of cosine similarity matrix : (95, 95)
```

```
prac1=prac1.reset_index()
def get_recommendations_w2v(id, cosine_sim=cosine_sim):
    indices = pd.Series(prac1.index, index = prac1['vendor_id']).drop_duplicate

    idx = indices[id]
    sim_scores = list(enumerate(cosine_sim[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    sim_scores = sim_scores[1:11]

    return sim_scores
```

```
get_recommendations_w2v('113')
```

```
[(22, 0.976377),
 (73, 0.951464),
 (14, 0.94462705),
 (61, 0.9121437),
 (80, 0.8967541),
 (29, 0.8609116),
 (15, 0.8567154),
 (42, 0.83256024),
 (36, 0.82227546),
 (30, 0.8153461)]
```

5.2.3 Doc2Vec с меткой времени

```
prac1.head()
```

	vendor_id	customer_id	OpeningTime	vendor_tag_name	Open	Close	afternoc
0	113	TCHWPBT	10:59AM-10:59PM	arabic,desserts,free delivery,indian	10:59AM	10:59PM	1
1	237	TCHWPBT	08:30PM-11:59PM	american,burgers,desserts,donuts,fries,pasta,s...	08:30PM	11:59PM	0
2	4	ZGFSYCZ	11:00AM-11:30PM	arabic,breakfast,burgers,desserts,free deliver...	11:00AM	11:30PM	1
3	28	ZGFSYCZ	11:00AM-11:45PM	burgers	11:00AM	11:45PM	1
4	33	ZGFSYCZ	11:00AM-10:30PM	desserts,mexican	11:00AM	10:30PM	1

```
prac2['time_tag_list']= prac2['time_tag'].str.split(' ')

corpus=prac2['time_tag_list']

model = Word2Vec(size=4, window=1, min_count=1, workers=4)
model.build_vocab(corpus)

word_vectors = model.wv
vocabs = word_vectors.vocab.keys()
vocabs
```

```
dict_keys(['arabic', 'desserts', 'freedelivery', 'indian', 'afternoon', 'evening', 'morning',
'american', 'burgers', 'donuts', 'fries', 'pasta', 'salads', 'sandwiches', 'breakfast', 'grill
s', 'lebanese', 'shawarma', 'mexican', 'asian', 'healthyfood', 'japanese', 'sushi', 'hotdogs',
'biryani', 'soups', 'freshjuices', 'smoothies', 'cakes', 'coffee', 'hotchocolate', 'bagels', 'k
idsmeal', 'pizza', 'familymeal', 'kebabs', 'rice', 'cafe', 'icecreams', 'italian', 'sweets', 't
hai', 'vegetarian', 'milkshakes', 'mandazi', 'omani', 'steaks', 'frozenyoghurt', 'mojitos', 'mi
shkak', 'organic', 'manakeesh', 'crepes', 'pancakes', 'waffles', 'chinese', 'dimsum', '', 'chur
ros', 'spanishlatte', 'pastry', 'seafood', 'rolls', 'karak', 'combos', 'kushari', 'shuwa'])
```

```
word_vectors.most_similar('breakfast')
```

```
[('kidsmeal', 0.9876297116279602),
 ('spanishlatte', 0.8556029200553894),
 ('seafood', 0.7407674789428711),
 ('dimsum', 0.6946371793746948),
 ('combos', 0.6774498224258423),
 ('mishkak', 0.6172083616256714),
 ('indian', 0.539692759513855),
 ('crepes', 0.42991745471954346),
 ('milkshakes', 0.4238659739494324),
 ('arabic', 0.4150455594062805)]
```

```
def vectors(document_list):
    document_embedding_list = []

    for line in document_list:
        doc2vec = None
        count = 0
        for word in line.split():
            if word in model.wv.vocab:
                count += 1
                if doc2vec is None:
                    doc2vec = model[word]
                else:
                    doc2vec = doc2vec + model[word]

        if doc2vec is not None:
            doc2vec = doc2vec / count
            document_embedding_list.append(doc2vec)
    return document_embedding_list

document_embedding_list = vectors(prac2['time_tag'])
print('the number of document vector:',len(document_embedding_list))

cosine_sim = cosine_similarity(document_embedding_list, document_embedding_list
print('the shape of cosine similarity matrix :',cosine_sim.shape)

indices = pd.Series(prac.index, index=prac2['vendor_id']).drop_duplicates()

def get_recommendations_w2v_time(id, cosine_sim=cosine_sim):
    indices = pd.Series(prac.index, index = prac2['vendor_id']).drop_duplicates

    idx = indices[id]
    sim_scores = list(enumerate(cosine_sim[idx]))
```

```
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:11]

    return sim_scores
```

```
the number of document vector: 95
the shape of cosine similarity matrix : (95, 95)
```

```
get_recommendations_w2v_time('113')
```

```
[(85, 0.97273004),
 (79, 0.96565914),
 (14, 0.95653903),
 (38, 0.9519874),
 (13, 0.9442766),
 (36, 0.9342253),
 (34, 0.92946523),
 (11, 0.91802347),
 (78, 0.9174308),
 (29, 0.91351855)]
```

Увидим, что производительность модели лучше всего в Doc2Vec с меткой времени в соответствии с косинусным сходством.

Doc2Vec с меткой времени > Doc2Vec > Tf-Idf с меткой времени > Tf-Idf

В этом порядке производительность модели хороша в системе рекомендаций, основанной на содержании.