Документация по рекомендательной системе

Общий итог работы: был построен сервис выдающий рекомендации пользователям по их visitor\_id

Изначально были предоставлены данные в виде 4 файлов с данными:

Events — датасет с событиями:

* timestamp — время события
* visitorid — идентификатор пользователя
* event — тип события
* itemid — идентификатор объекта
* transactionid — идентификатор транзакции, если она проходила

Category\_tree — файл с деревом категорий:

* category\_id — идентификатор категорий
* parent\_id — идентификатор родительской категории

Item\_properties\_part\_1 и Item\_properties\_part\_2 — файлы со свойствами товаров:

* timestamp — момент записи значения свойства
* item\_id — идентификатор объекта
* property — свойство, кажется, они все, кроме категории, захешированы
* value — значение свойства

## **Данные для обучения**

На основе проведённого исследования были выделены и сформированы следующие признаки для обучения модели:

### **Features Visitors**

#### **Time Features / Временные признаки**

1.1. month / месяц

1.2. day\_week / день недели

1.3. hour / час дня

#### **Events Features / Признаки действий**

2.1. add\_visitor / суммарное количество добавлений в корзину у пользователя

2.2. trans\_visitor / суммарное количество покупок у пользователя

2.3. view\_visitor / суммарное количество просмотров у пользователя

2.4. all\_events\_vis / общее количество всех действий пользователя (сумма)

2.5. buy\_% / доля покупок по отношению к общему количеству всех действий (% от всех действий) пользователя

2.6. view\_trans\_vis / среднее количество просмотров по отношению к покупке (сколько в среднем просмотров приходится на одну покупку) пользователя

2.7. add\_trans\_vis / среднее количество добавлений в корзину по отношению к покупке (сколько в среднем добавлений приходится на одну покупку) пользователя

2.8. view\_purch\_vis / количество просмотров при покупке данного айтема у пользователя (история покупок прослеживается по конкретному айтему)

2.9. add\_purch\_vis / количество добавлений в корзину при покупке данного айтема у пользователя (история покупок прослеживается по конкретному айтему)

#### **Purchases Frequency Features / Признаки частотности покупок**

3.1. items\_per\_trans / сколько айтемов было куплено в одном чеке (комплексность)

3.2. repeat\_count / количество повторных покупок

#### **Bestsellers Features / Признаки бестселлеров**

4.1. top\_20\_purch / количество купленных пользователем товаров из списка ТОП-20 (20 наиболее часто продававшихся товаров)

4.2. month\_top\_purch /количество купленных пользователем товаров из списка бестселлеров месяца (10 наиболее часто продававшихся товаров, общий список уникальных артикулов для каждого месяца из периода наблюдений в датасете)

#### **Activity Features / Признаки периодов наибольшей активности пользователя**

5.1. activity\_duration\_vis / длительность периода активности (исчисляется в днях, от первого до последнего дня активности в системе)

5.2. month\_vis / месяц совершения наибольшего числа покупок

5.3. day\_vis / день недели, в который пользователь совершает наибольшее количество покупок

5.4. hour\_vis / час дня, в который пользователь совершает наибольшее количество покупок

5.5. activity\_vis\_days / количество дней, в которые покупатель совершил покупку (например, покупатель был активен 5 дней, но покупал только 2 дня из них, 2 дня относятся к этому признаку)

5.6. usedays% / доля результативных дней, в которые совершались покупки, по отношению ко всему периоду активности (например, покупатель был активен 5 дней, но покупал только 2 дня из них, 2/5\*100% - 40% были результативными для системы)

### **ITEM features**

#### **1. Time Features / Временные признаки**

1.1. month / месяц

1.2. day\_week / день недели

1.3. hour / час дня

#### **2. Events Features / Признаки действий**

2.1. add\_item / суммарное количество добавлений артикула (товара) в корзину

2.2. trans\_item / суммарное количество покупок артикула

2.3. view\_item / суммарное количество просмотров артикула

2.4. all\_events\_itm / общее количество всех действий по отношению к артикулу (сумма)

2.5. sale\_% / доля продаж по отношению к общему количеству всех действий с товаром

#### **3. Sales Frequency Features / Признаки частотности продаж**

3.1. repeat\_count / количество повторных покупок артикула

#### **4. Bestsellers Features / Признаки бестселлеров**

4.1. top\_20\_sale / входит ли артикул в ТОП-20 (20 наиболее часто продававшихся товаров)

4.2. top\_month\_sale /входит ли артикул в общий список бестселлеров месяца (10 наиболее часто продававшихся товаров, общий список уникальных артикулов для каждого месяца из периода наблюдений в датасете)

#### **5. Activity Features / Признаки периодов наиболее активного взаимодействия пользователя с товарами**

5.1. activity\_duration\_itm / длительность периода активности (исчисляется в днях, от первого до последнего дня взаимодействия пользователя с товаром в системе)

5.2. month\_itm / месяц наиболее частых продаж артикула

5.3. day\_itm / день недели, в который артикул чаще всего продавался

5.4. hour\_itm / час дня, в который артикул чаще всего продавался

5.5. sale\_days / количество дней, в которые совершалась продажа артикула

5.6. sale\_day\_s% / доля результативных дней, в которые совершались продажи артикула, по отношению ко всему периоду активности (например, покупатели просматривали артикул в течение 5 дней, но покупали только 2 дня из них, 2/5\*100% - 40% были результативными для системы)

Произошедшие трансформации с исходным датасетом для получения датасета для обучения описаны в приложении 1.

## **Разделение на Train и Test:**

В датасет, который был затем разделен на 2 части, были отобраны только пользователи, которые совершили не менее, чем 2 покупки. Затем лог каждого пользователя был разделен на Train и Test с последующим объединением, так, чтобы первая покупка вошла в Train, а вторая, более поздняя по времени - в Test. Если покупок было больше, чем 2, они распределялись между Train и Test в пропорции 70/30.

## **Проведённые эксперименты:**

При построении рекомендательной системы было проведено 3 эксперимента:

### **Baseline (Experiment 0)**

- предложение каждому пользователю 3 самых популярных товаров

На случайной выборке из 30 пользователей с применением бейзлайна мы получили следующую метрику:

precision@3 - 0.01

Далее будут эксперименты одной из систем рекомендаций Google, Two Towers (<https://medium.com/@rubenszimbres/train-and-deploy-google-clouds-two-towers-recommender-3ae5c29891af>). Вкратце ее структура такова:

* определение похожих на исходного user пользователей и отбор товаров-кандидатов по их истории покупок с последующей фильтрацией (Башня 1)
* генерация товаров-кандидатов на основе истории покупок или рейтингов пользователя (Башня 2)
* определение пересечений в результатах двух предыдущих отборов (dot-item)

Однако практические попытки построить такую систему на основе исходного датасета показали, что при большом количестве единичных покупок редко генерируются пересекающиеся множества товаров при коллаборативной фильтрации по пользователям и по товарам. Поэтому генерация кандидатов на основе пользователей и товаров были рассмотрены как самостоятельные эксперименты.

### **Сosine Similarity (Experiment 1)**

- кандидатогенерация на основе Cosine similarity юзеров и айтемов с последующей фильтрацией при помощи рейтинга (При этом оценка ставится по количеству действий каждого типа, совершенных пользователями - 1 балл просмотр, 2 балла - добавление в корзину, 5 баллов - покупка.),

#### 

#### **VISITORS matrix**

На случайной выборке из 30 пользователей с применением коллаборативной фильтрации по пользователям (оценка близости векторов методом Cosine similarity) мы получили следующую метрику:

precision@3 - 0.01

#### **ITEMS matrix**

На случайной выборке из 30 пользователей с применением коллаборативной фильтрации по товарам (оценка близости векторов методом Cosine similarity) мы получили следующую метрику:

precision@3 - 0.06

### **Logistic Matrix Factorization (Experiment 2)**

- кандидатогенерация на основе матрицы item-user с последующей фильтрацией при помощи рейтинга.

#### **Item-User Matrix only**

На случайной выборке из 30 пользователей с применением логистической матричной факторизации (простое функциональное предсказание) мы получили следующую метрику:

precision@3 - 0.06

#### 

#### **Similar VISITORS prediction**

На случайной выборке из 30 пользователей с применением коллаборативной фильтрации по пользователям на основании логистической матричной факторизации с последующим отборов лучших товаров на основе рейтинга мы получили следующую метрику:

precision@3 - 0.33

#### **Similar ITEMS prediction**

На случайной выборке из 30 пользователей с применением коллаборативной фильтрации по товарам на основе логистической матричной факторизации мы получили следующую метрику:

precision@3: 0.32

В итоге, лучший результат показала коллаборативная фильтрация по пользователям на основании логистической матричной факторизации с последующим отборов лучших товаров на основе рейтинга.

## **Приложение**

Для создания приложения были использованы Flask и Pycharm.

Само приложение лежит в папке **prj**. Приложение базируется на 3 файлах .py (app.py(основной), form.py, ml\_model.py) и 2 файла templates(index.html, recom.html). Также ввиду сложно процесса рекомендаций был подгружены следующие файлы: user\_index.csv, user\_features.csv, item\_user\_coo.pkl.

Для создания docker image были также написаны файлы dockerfile и requirements.

Общий принцип действия приложения следующий:

При запуске программы из файла app.py активируется приложение. Всего есть 3 возможные ссылки:

@app.route('/',)

@app.route('/index',)

@app.route('/recom')

При запросе GET мы попадаем на index, где предлагается ввести свой id (visitorid) в специальную форму заданную form.py. Получая visitorid на вход (метод POST) функция get\_prediction (файл ml\_model.py ) проверяет есть ли он в наших данных. Если visitorid уже существует, то запускается вторая функция, которая с помощью логистической матричной факторизации, производит коллаборативную фильтрацию по пользователям на основании логистической матричной факторизации с последующим отборов лучших товаров на основе рейтинга и выдаёт персональные рекомендации, если это новый пользователь, то выдаются топ-3 популярных товара (бейзлайн). Полученный ответ выводится на страницу recom

Для возможности запуска данного приложения из любой среды, был создан dockerfile на основе python.

Docker image называется my\_project\_image.

Создан с помощью команды: docker build -t my\_project\_image .

Поднять образ можно с помощью команды: docker run -it --rm --name my\_project\_image my\_project\_image

app.py:

from flask import Flask

from flask\_sqlalchemy import SQLAlchemy

from flask\_migrate import Migrate

import os

from form import EditForm

from flask import render\_template, redirect, request

from sqlalchemy import func

from ml\_model import model\_prediction

basedir = os.path.abspath(os.path.dirname(\_\_file\_\_))

app = Flask(\_\_name\_\_)

app.config['SECRET\_KEY'] = 'you-will-never-guess'

app.config['SQLALCHEMY\_DATABASE\_URI'] = os.environ.get('DATABASE\_URL') or \

'sqlite:///' + os.path.join(basedir, 'app.db')

app.config['SQLALCHEMY\_TRACK\_MODIFICATIONS'] = False

db = SQLAlchemy(app)

migrate = Migrate(app, db)

@app.route('/', methods=['GET','POST'])

@app.route('/index', methods=['GET','POST'])

@app.route('/recom', methods=['GET','POST'])

def get\_prediction():

if request.method == 'GET':

form=EditForm()

return render\_template('index.html', form=form)

else:

args = request.form

visitor\_id = args['visitor\_id']

return render\_template('recom.html', prediction = model\_prediction(visitor\_id))

if \_\_name\_\_ == '\_\_main\_\_':

app.run()

**form.app**

from flask\_wtf import FlaskForm

from wtforms import IntegerField, SubmitField

from wtforms.validators import DataRequired

class EditForm(FlaskForm):

visitor\_id = IntegerField('YourId')

submit = SubmitField('Sign in')

**ml\_model.app**

import pandas as pd

import implicit

import numpy as np

import pickle

import scipy

user\_index=pd.read\_csv('user\_index.csv')

user\_features=pd.read\_csv('user\_features.csv')

with open('item\_user\_coo.pkl', 'rb') as f:

item\_user\_coo = pickle.load(f)

model\_best = implicit.lmf.LogisticMatrixFactorization(factors=20, regularization=0.1, iterations=20, neg\_prop=5)

model\_best.fit(item\_user\_coo)

def m\_vis\_items(num):

### 1. Similar users from matrix

# Index for desired user

usr = user\_index.loc[user\_index.visitorid == num]['index'].iloc[0]

# Similar users generation (5 users)

similar\_users = model\_best.similar\_users(userid=usr, N=5)

similar\_users = pd.DataFrame(similar\_users)

similar\_users = similar\_users.T

similar\_users.columns = ['us\_index', 'score']

# Indices list

users\_index\_list = similar\_users.us\_index.tolist()

similar\_visitors = user\_index.loc[user\_index.index.isin(users\_index\_list)]

similar\_visitors = similar\_visitors.visitorid.to\_list()

### 2. Items generation

cols = ['visitorid', 'itemid', 'event']

purchased\_items = user\_features[cols]

purchased\_items = purchased\_items.loc[

(purchased\_items.event == 'transaction') & (purchased\_items.visitorid.isin(similar\_visitors))]

purchased\_items = purchased\_items.itemid.unique().tolist() # unique items

### 3. Items filtering by rating

# 3.1. Similar users ratingitemid

cols = ['visitorid', 'itemid', 'event']

vis\_items\_rating = user\_features[cols]

cols = ['visitorid', 'itemid', 'event']

sim\_vis\_items\_rating = user\_features[cols]

sim\_vis\_items\_rating = sim\_vis\_items\_rating.loc[(sim\_vis\_items\_rating.visitorid.isin(similar\_visitors)) & (

sim\_vis\_items\_rating.itemid.isin(purchased\_items))]

score\_dict = {'transaction': 5,

'addtocart': 2,

'view': 1}

sim\_vis\_items\_rating['score'] = sim\_vis\_items\_rating['event'].map(score\_dict)

sim\_vis\_items\_rating = pd.pivot\_table(sim\_vis\_items\_rating,

values='score',

index=['visitorid', 'itemid'],

aggfunc=np.sum)

sim\_vis\_items\_rating = sim\_vis\_items\_rating.reset\_index() # initial users rating

# 3.2. Desired visitor's backlog rating

sing\_vis\_items\_rating = user\_features[cols]

sing\_vis\_items\_rating = sing\_vis\_items\_rating.loc[(sing\_vis\_items\_rating.visitorid == num) & (

sing\_vis\_items\_rating.itemid.isin(purchased\_items))]

sing\_vis\_items\_rating['score'] = sing\_vis\_items\_rating['event'].map(score\_dict)

sing\_vis\_items\_rating = pd.pivot\_table(sing\_vis\_items\_rating,

values='score',

index=['itemid'],

aggfunc=np.sum)

sing\_vis\_items\_rating = sing\_vis\_items\_rating.reset\_index()

# 3.3. Final rating (similar users + desired visitor)

common\_rating = sim\_vis\_items\_rating.merge(sing\_vis\_items\_rating, on='itemid', how='left')

columns\_count = common\_rating.shape[1]

if columns\_count == 3:

common\_rating['sing\_score'] = 0

else:

common\_rating['sing\_score'] = common\_rating['score\_y']

common\_rating['final'] = common\_rating[common\_rating.columns[2]] + common\_rating['sing\_score']

common\_rating = common\_rating.sort\_values(by='final', ascending=False)

prediction = common\_rating[0:3]['itemid'].to\_list()

return prediction

def model\_prediction(visitor\_id):

user\_list = user\_index['visitorid'].tolist()

if visitor\_id in user\_list:

prediction = m\_vis\_items(visitor\_id)

return prediction

else:

top\_3 = [213834, 461686, 445351]

prediction = top\_3

return prediction

return prediction

**index.html**

<!DOCTYPE html>

<html>

<head>

<title> Sign in </title>

</head>

<body>

{% block content %}

<h1>Sign in with your id</h1>

<form action="" method="post" novalidate>

<p>

{{ form.visitor\_id.label }}<br>

{{ form.visitor\_id(size=32) }}

</p>

<p>{{ form.submit() }}</p>

</form>

{% endblock %}

</body>

</html>

recom.html

<!DOCTYPE html>

<html>

<head>

<title> Personal recomendation </title>

</head>

<body>

<h1>Personal recomendation!</h1>

<div>

<hr>

<form action="" method="post" novalidate>

<p><b>{{ prediction }}</b></p>

</div>

</body>

</html>

**dockerfile:**

FROM python:3.8

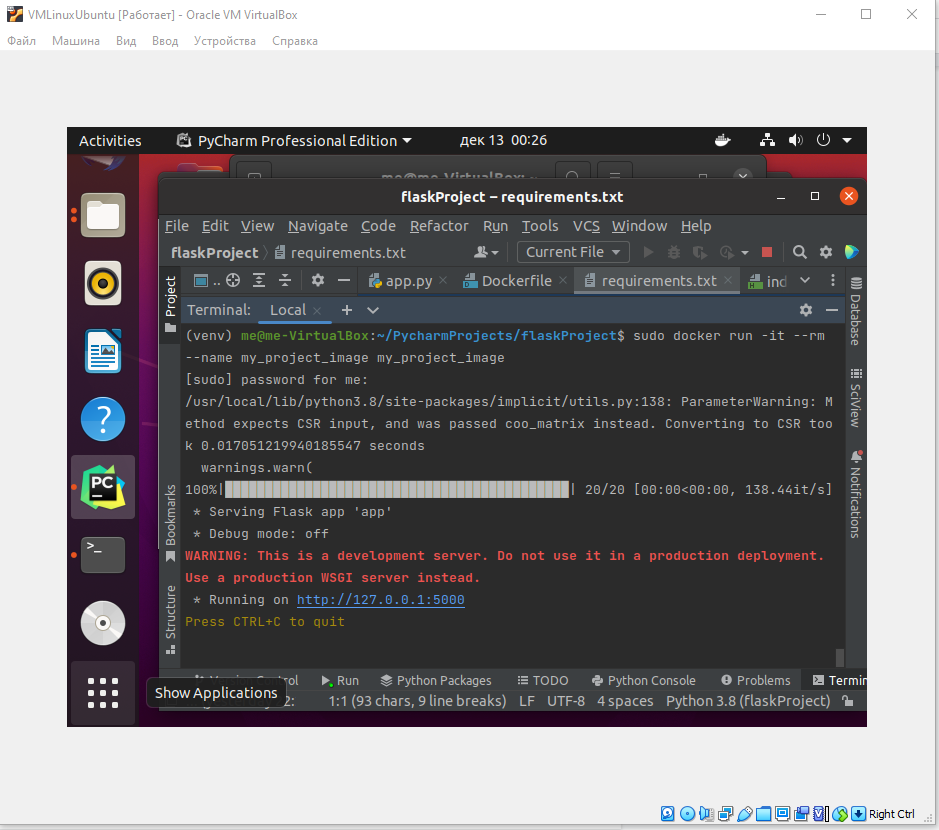
WORKDIR /usr/prj/app

COPY requirements.txt ./

RUN pip install --no-cache-dir -r requirements.txt

COPY ./prj/ .

CMD [ "python", "./app.py"]



**requirements.txt**

flask

flask\_wtf

wtforms

flask\_sqlalchemy

flask\_migrate

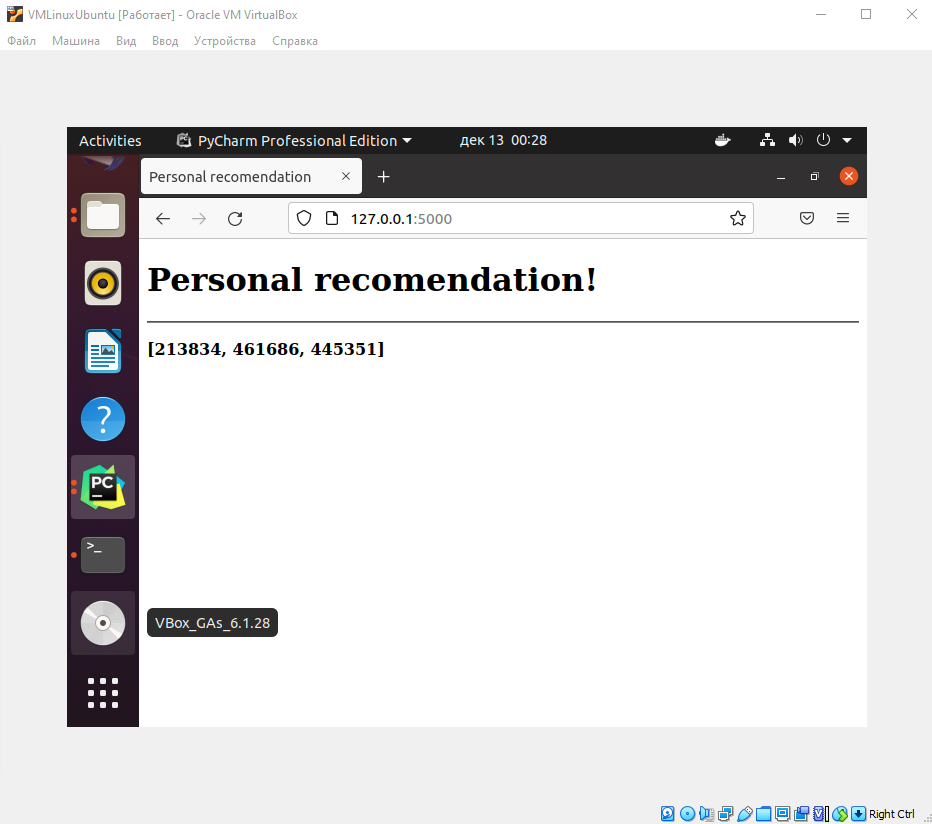
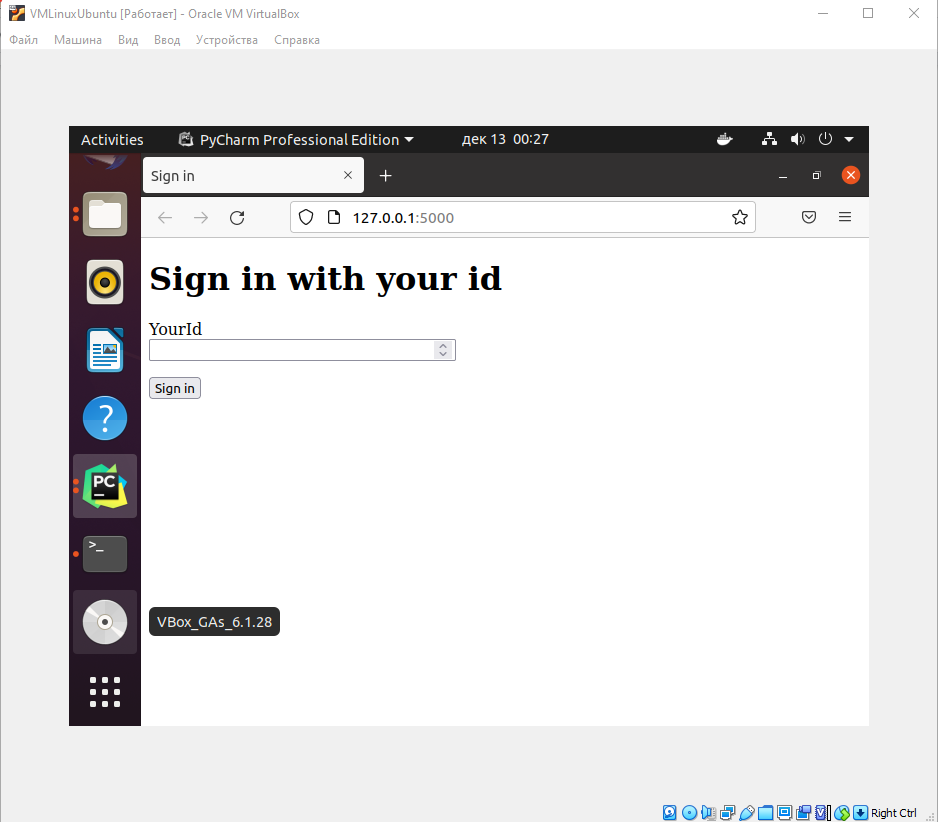
sqlalchemy

pandas

implicit

numpy

scipy



Приложение 1

[21]:

events.event.value\_counts()

[21]:

view 2664312

addtocart 69332

transaction 22457

Name: event, dtype: int64

[22]:

events.nunique()

[22]:

timestamp 2750455

visitorid 1407580

event 3

itemid 235061

transactionid 17672

dtype: int64

[23]:

duplicateRows **=** events[events.duplicated()]

[24]:

duplicateRows.event.value\_counts()

[24]:

addtocart 366

view 94

Name: event, dtype: int64

Как можно видеть, в данном датасете есть дубликаты. Они относятся не к транзакциям, а к просмотру товара или добавлению в корзину. По логике, добавить в корзину несколько штук товара сразу, возможно. Но вот просмотр одновременно одного и того же товара один и тем же клиентом в одно время, кажется сомнительным. Пока оставим данные дубликаты, но в дальнейшим удалим их при необходимости events.drop\_duplicates()

[25]:

events.nunique()

[25]:

timestamp 2750455

visitorid 1407580

event 3

itemid 235061

transactionid 17672

dtype: int64

[26]:

events.isnull().sum()

[26]:

timestamp 0

visitorid 0

event 0

itemid 0

transactionid 2733644

dtype: int64

Как можно увидеть, в transactionid много пропусков. Скорее всего, это означает, что visitor не совершил покупку, проверим.

[27]:

events.groupby(['event']).transactionid.value\_counts()

[27]:

event transactionid

transaction 7063.0 31

765.0 28

8351.0 27

2753.0 23

6993.0 21

..

17666.0 1

17667.0 1

17668.0 1

17670.0 1

17671.0 1

Name: transactionid, Length: 17672, dtype: int64

Действительно, transactionid есть только у event-transaction, и у каждого event-transaction есть transactionid

[28]:

events **=** events.fillna(0)

[29]:

*#Making date features from timestamp*

events['date'] **=** pd.to\_datetime(events['timestamp'], unit**=**'ms')

events['format\_date'] **=** pd.DatetimeIndex(events['date']).date

events['month'] **=** pd.DatetimeIndex(events['date']).month

events['day\_week'] **=** pd.DatetimeIndex(events['date']).dayofweek

events['hour'] **=** pd.DatetimeIndex(events['date']).hour

[31]:

events.event.value\_counts()

[31]:

view 2664312

addtocart 69332

transaction 22457

Name: event, dtype: int64

[32]:

len(events.loc[events.event**==**'transaction'])**/**events.visitorid.nunique()

[32]:

0.015954332968641214

[33]:

events.loc[events.event**==**'transaction']['visitorid'].value\_counts().reset\_index(drop**=True**)

[33]:

0 559

1 349

2 286

3 189

4 188

...

11714 1

11715 1

11716 1

11717 1

11718 1

Name: visitorid, Length: 11719, dtype: int64

[34]:

events.loc[events.event**==**'transaction']['visitorid'].value\_counts().loc[**lambda** x : x **>** 1].reset\_index(drop**=True**)

[34]:

0 559

1 349

2 286

3 189

4 188

...

2571 2

2572 2

2573 2

2574 2

2575 2

Name: visitorid, Length: 2576, dtype: int64

[35]:

len(events.loc[events.event**==**'view'])**/**events.visitorid.nunique()

[35]:

1.89283166853749

[36]:

len(events.loc[events.event**==**'addtocart'])**/**events.visitorid.nunique()

[36]:

0.04925617016439563

[37]:

events.loc[events.event**==**'addtocart']['visitorid'].value\_counts().reset\_index(drop**=True**)

[37]:

0 719

1 419

2 371

3 296

4 231

...

37717 1

37718 1

37719 1

37720 1

37721 1

Name: visitorid, Length: 37722, dtype: int64

[38]:

events.loc[events.event**==**'addtocart']['visitorid'].value\_counts().loc[**lambda** x : x **>** 1].reset\_index(drop**=True**)

[38]:

0 719

1 419

2 371

3 296

4 231

...

9070 2

9071 2

9072 2

9073 2

9074 2

Name: visitorid, Length: 9075, dtype: int64

[39]:

len(events.loc[events.event**==**'transaction']['itemid'].unique())**/**events.itemid.nunique()

[39]:

0.051156933732095074

[40]:

len(events.loc[events.event**==**'view'])**/**events.itemid.nunique()

[40]:

11.334555711070744

[41]:

events.loc[events.event**==**'view']['itemid'].value\_counts().reset\_index(drop**=True**)

[41]:

0 3410

1 2539

2 2325

3 1854

4 1740

...

234833 1

234834 1

234835 1

234836 1

234837 1

Name: itemid, Length: 234838, dtype: int64

[42]:

events.loc[events.event**==**'view']['itemid'].value\_counts().loc[**lambda** x : x **>** 1].reset\_index(drop**=True**)

[42]:

0 3410

1 2539

2 2325

3 1854

4 1740

...

161040 2

161041 2

161042 2

161043 2

161044 2

Name: itemid, Length: 161045, dtype: int64

Анализируя данный датасет, можно сделать следующие выводы: 1)Меньше 2 процентов пользователей делают покупку, и около 5 что-то кладут в корзину, при этом все пользователи в среднем просматривают около 2 товаров, что говорит о достаточно активном поведении. 2)Из всех представленных в датасете товаров, только 5 процентов покупается. На каждый товар приходится примерно по 11 просмотров 3)Стоит учитывать наличие клиентов, которые покупают большое количество товаров, так и наличие популярных товаров.

### **Items**

[44]:

items\_1.nunique()

timestamp 18

itemid 417053

property 1097

value 1231581

dtype: int64

[45]:

items\_1[items\_1.duplicated()].value\_counts()

Series([], dtype: int64)

[46]:

items\_2.nunique()

[47]:

timestamp 18

itemid 417053

property 1094

value 1075730

dtype: int64

[48]:

items\_2[items\_2.duplicated()].value\_counts()

[48]:

Series([], dtype: int64)

timestamp имеет в обоих датасетах очень малое количество уникальных значение. Так как на данный момент, нет оснований полагать, что момент записи значения свойства имеет большое значение, удалим данный столбец

[49]:

items\_1**=**items\_1.drop(columns **=** ['timestamp'],axis **=** 1)

[50]:

items\_2**=**items\_2.drop(columns **=** ['timestamp'],axis **=** 1)

[51]:

items\_1[items\_1.duplicated()].value\_counts()

[51]:

itemid property value

441950 888 866645 40557 1256206 1337784 992862 951748 685028 1284577 892689 1284577 n33.600 1135780 1284577 n48.000 1187104 1319429 661116 1257525 1284577 n12000.000 1187104 1284577 618514 980588 927741 544576 1284577 670753 n96.000 1284577 1169991 17

285749 888 325083 40557 10680 427700 768342 992862 951748 1011636 1284577 1299688 1284577 n34.800 1135780 1284577 n96.000 1187104 539109 661116 1257525 1284577 n12000.000 1187104 1284577 618514 980588 399087 593327 1284577 670753 n96.000 1284577 1169991 17

287242 888 5135 1095464 992862 951748 1137726 n21.600 1135780 1284577 n187.200 1284577 n16392.000 628176 n9216.000 134030 1284577 n72.000 1187104 1284577 n12.000 1175087 n6000.000 1187104 1284577 1186729 681805 612841 1284577 1055803 1164666 625815 584849 1284577 1141052 n48.000 288796 1284577 814297 670753 n96.000 17

118803 888 128748 547104 n687823920.000 992862 951748 103615 1284577 1096273 1284577 n40.800 1135780 1284577 n96.000 1187104 1319429 661116 1257525 1284577 n24000.000 1187104 1284577 1055803 1164666 625815 941273 1284577 670753 n96.000 1284577 1169991 17

118771 888 435092 191135 120057 1055803 278045 1284577 267729 1284577 n15.600 1135780 1284577 n48.000 1187104 661116 1257525 1284577 n12000.000 1187104 1284577 1055803 1164666 1121447 1284577 842796 1284577 1298303 17

..

12357 790 n43560.000 1

n42360.000 1

347523 888 762199 327156 n24.000 1187104 1

27804 888 495153 1297579 1

301892 888 302280 n696.000 584910 1154859 1

Length: 635350, dtype: int64

add Codeadd Markdown

[52]:

items\_2[items\_2.duplicated()].value\_counts()

[52]:

itemid property value

411225 888 128748 499427 n687907440.000 992862 647924 1284577 1031160 1284577 n28.800 1135780 1284577 n48.000 1187104 539109 661116 1257525 1284577 n6000.000 1187104 1284577 992862 625815 1119999 1284577 842796 1284577 1169991 17

256206 888 1206276 1055803 278045 718056 n27.600 1135780 1284577 n207.600 1284577 n19200.000 628176 n10800.000 134030 1284577 n72.000 1187104 1284577 n24.000 1175087 n9000.000 1187104 1284577 1186729 681805 612841 1284577 1055803 1164666 625815 62261 1284577 1141052 n48.000 288796 1284577 814297 670753 n84.000 685584 789585 n768.000 535229 17

44493 364 266290 17

77379 888 874902 862092 992862 951748 1090293 n26.400 1135780 1284577 n187.200 1284577 n16392.000 628176 n9216.000 134030 1284577 n48.000 1187104 1284577 n12.000 1175087 n6000.000 1187104 1284577 491900 1284577 1055803 1164666 625815 683119 1284577 1141052 n36.000 288796 1284577 250863 17

377640 888 866645 40557 1180929 1195099 992862 951748 1011636 1284577 13722 1284577 n19.200 1135780 1284577 n48.000 1187104 1319429 661116 1257525 1284577 n12000.000 1187104 1284577 618514 980588 927741 376092 1284577 670753 n96.000 1284577 1169991 17

..

26920 6 537209 1

394044 790 n110520.000 1

394040 categoryid 491 1

241718 888 150169 1249617 561964 n2808.000 1154859 86628 n1344000.000 786769 1

335229 790 n20760.000 1

Length: 545501, dtype: int64

Как можно увидеть по количеству уникальных значений, в датасетах item большое количество повторов, относящиеся к разным данным. Так как полные дубликаты появились при удаление timestap, есть вероятность, что многие товары дублировались при загрузки в разное время. Возможно данную информацию можно использовать как признак, но на данном этапе, ввиду большего количества данных, удалим дубликаты.

[53]:

items\_1**=**items\_1.drop\_duplicates().reset\_index(drop**=True**)

[54]:

items\_2**=**items\_2.drop\_duplicates().reset\_index(drop**=True**)

[55]:

items\_1.nunique()

[55]:

itemid 417053

property 1097

value 1231581

dtype: int64

[56]:

items\_2.nunique()

[56]:

itemid 417053

property 1094

value 1075730

dtype: int64

[57]:

display(items\_1.shape)

display(items\_2.shape)

(7042064, 3)

(5958844, 3)

Интересно, что несмотря на то, что количество наименований свойств практически одинаковое в обоих датасетах, количество значений свойств разнится. То есть, судя по всему каждому свойству может соответствовать несколько значений, в дальнейшем стоит обратить на это внимание

### **Tree**

[58]:

tree[tree.duplicated()].value\_counts()

[58]:

Series([], dtype: int64)

[59]:

tree.nunique()

[59]:

categoryid 1669

parentid 362

dtype: int64

[60]:

Пропущенные значения имеются только в parentid, их вего 25. Мы могли бы удалить их, но более правильно заполнить их. Можно заполнить пропуски с использованием средних значений, что является довольно топорным. Поэтому обратитмся к алгоритму k ближайших соседей

imputer **=** KNNImputer(n\_neighbors**=**5)

no\_nan **=** imputer.fit\_transform(tree[['categoryid', 'parentid']])

tree **=** pd.DataFrame(no\_nan, columns **=** ['categoryid','parentid'])

tree['parentid'] **=** round(tree['parentid'])

tree.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 1669 entries, 0 to 1668

Data columns (total 2 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 categoryid 1669 non-null float64

1 parentid 1669 non-null float64

dtypes: float64(2)

memory usage: 26.2 KB

[67]:

*#Selecting visitors with non less than 2 purchases*

df\_1 **=** events.copy()

​

cols **=** ['visitorid', 'event']

df\_vis **=** df\_1[cols]

df\_vis **=** df\_vis.loc[df\_vis.event**==**'transaction']

df\_vis['count'] **=** 1

df\_vis.drop(columns**=**'event', axis**=**1, inplace**=True**)

df\_vis **=** df\_vis.groupby('visitorid').sum().reset\_index()

df\_vis **=** df\_vis.loc[df\_vis['count']**>**1]

# **8. Feature engineering**

add Codeadd Markdown

### **Features Visitors**

#### **Time Features / Временные признаки**

1.1. month / месяц

1.2. day\_week / день недели

1.3. hour / час дня

#### **Events Features / Признаки действий**

2.1. add\_visitor / суммарное количество добавлений в корзину у пользователя

2.2. trans\_visitor / суммарное количество покупок у пользователя

2.3. view\_visitor / суммарное количество просмотров у пользователя

2.4. all\_events\_vis / общее количество всех действий пользователя (сумма)

2.5. buy\_% / доля покупок по отношению к общему количеству всех действий (% от всех действий) пользователя

2.6. view\_trans\_vis / среднее количество просмотров по отношению к покупке (сколько в среднем просмотров приходится на одну покупку) пользователя

2.7. add\_trans\_vis / среднее количество добавлений в корзину по отношению к покупке (сколько в среднем добавлений приходится на одну покупку) пользователя

2.8. view\_purch\_vis / количество просмотров при покупке данного айтема у пользователя (история покупок прослеживается по конкретному айтему)

2.9. add\_purch\_vis / количество добавлений в корзину при покупке данного айтема у пользователя (история покупок прослеживается по конкретному айтему)

#### **Purchases Frequency Features / Признаки частотности покупок**

3.1. items\_per\_trans / сколько айтемов было куплено в одном чеке (комплексность)

3.2. repeat\_count / количество повторных покупок

#### **Bestsellers Features / Признаки бестселлеров**

4.1. top\_20\_purch / количество купленных пользователем товаров из списка ТОП-20 (20 наиболее часто продававшихся товаров)

4.2. month\_top\_purch /количество купленных пользователем товаров из списка бестселлеров месяца (10 наиболее часто продававшихся товаров, общий список уникальных артикулов для каждого месяца из периода наблюдений в датасете)

#### **Activity Features / Признаки периодов наибольшей активности пользователя**

5.1. activity\_duration\_vis / длительность периода активности (исчисляется в днях, от первого до последнего дня активности в системе)

5.2. month\_vis / месяц совершения наибольшего числа покупок

5.3. day\_vis / день недели, в который пользователь совершает наибольшее количество покупок

5.4. hour\_vis / час дня, в который пользователь совершает наибольшее количество покупок

5.5. activity\_vis\_days / количество дней, в которые покупатель совершил покупку (например, покупатель был активен 5 дней, но покупал только 2 дня из них, 2 дня относятся к этому признаку)

5.6. usedays% / доля результативных дней, в которые совершались покупки, по отношению ко всему периоду активности (например, покупатель был активен 5 дней, но покупал только 2 дня из них, 2/5\*100% - 40% были результативными для системы)

[84]:

user\_features **=** train.copy()

[85]:

user\_features['month'] **=** pd.DatetimeIndex(user\_features['format\_date']).month

user\_features['day\_week'] **=** pd.DatetimeIndex(user\_features['format\_date']).dayofweek

user\_features['hour'] **=** pd.DatetimeIndex(user\_features['format\_date']).hour

[86]:

*#Pivoting*

cols **=** ['visitorid', 'event']

user\_events **=** train[cols]

user\_events['count'] **=** 1

​

user\_events **=** pd.pivot\_table(user\_events,

values**=**'count',

index**=**['visitorid'],

columns**=**['event'], aggfunc**=**np.sum, fill\_value**=**0)

*#Adding features*

user\_events **=** user\_events.reset\_index()

user\_events['all\_events\_vis'] **=** user\_events['addtocart']**+**user\_events['transaction']**+**user\_events['view']

user\_events['buy\_%'] **=** round((user\_events['transaction']**/**user\_events['all\_events\_vis']**\***100))

​

user\_events **=** user\_events.rename(columns**=**{'addtocart': 'add\_visitor',

'transaction': 'trans\_visitor',

'view': 'view\_visitor'})

​

​

user\_events['view\_trans\_vis'] **=** round((user\_events['view\_visitor']**/**user\_events['trans\_visitor']),2)

user\_events['add\_trans\_vis'] **=** round((user\_events['add\_visitor']**/**user\_events['trans\_visitor']),2)

​

*#Adding to frame*

user\_features **=** user\_features.merge(user\_events, on**=**'visitorid', how**=**'left')

​

*#Purchases statistic*

cols **=** ['visitorid', 'itemid', 'event']

user\_purchases **=** train[cols]

purch\_items **=** user\_purchases.loc[user\_purchases.event**==**'transaction']['itemid'].tolist()

user\_purchases **=** user\_purchases.loc[user\_purchases.itemid.isin(purch\_items)]

user\_purchases['count'] **=** 1

​

*#Pivoting*

user\_purchases **=** pd.pivot\_table(user\_purchases,

values**=**'count',

index**=**['visitorid'],

columns**=**['event'], aggfunc**=**np.sum, fill\_value**=**0)

​

​

user\_purchases['view\_purch\_vis'] **=** round((user\_purchases['view']**/**user\_purchases['transaction']),2)

user\_purchases['add\_purch\_vis'] **=** round((user\_purchases['addtocart']**/**user\_purchases['transaction']),2)

​

cols **=** ['view\_purch\_vis', 'add\_purch\_vis']

user\_purchases **=** user\_purchases[cols].reset\_index()

​

*#Adding to frame*

user\_features **=** user\_features.merge(user\_purchases, on**=**'visitorid', how**=**'left')

[87]:

*#Items per Transaction*

cols **=** ['visitorid', 'transactionid']

​

prod\_features\_df **=** train.loc[train.event**==**'transaction'][cols]

prod\_features\_df['items\_per\_trans'] **=** 1

​

*#Pivoting*

prod\_features\_df **=** prod\_features\_df.groupby(by**=**['visitorid', 'transactionid']).sum()

​

prod\_features\_df **=** prod\_features\_df.groupby(by**=**['visitorid']).mean()

prod\_features\_df['items\_per\_trans'] **=** round(prod\_features\_df['items\_per\_trans'],2)

prod\_features\_df **=** prod\_features\_df.reset\_index()

​

*#Adding to frame*

user\_features **=** user\_features.merge(prod\_features\_df, on**=**'visitorid', how**=**'left')

​

*#Repeat purchasing*

cols **=** ['visitorid', 'itemid']

repeat\_purch **=** user\_features.loc[user\_features.event**==**'transaction'][cols]

repeat\_purch['items\_count'] **=** 1

repeat\_purch **=** repeat\_purch.groupby(['visitorid', 'itemid']).sum()

repeat\_purch **=** repeat\_purch.loc[repeat\_purch['items\_count']**>**1]

repeat\_purch['repeat\_count'] **=** 1

repeat\_purch **=** repeat\_purch.groupby(by**=**'visitorid').sum().reset\_index()

repeat\_purch **=** repeat\_purch[['visitorid', 'repeat\_count']]

​

*#Adding to frame*

user\_features **=** user\_features.merge(repeat\_purch, on**=**'visitorid', how**=**'left')

[88]:

*#Top\_20 (frequency)*

*### Top\_20 Products*

top\_20\_items **=** train.loc[train.event**==**'transaction']['itemid'].value\_counts()[:20].index.tolist()

​

cols **=** ['visitorid', 'itemid']

bestsellers\_purchase **=** user\_features.loc[user\_features.event**==**'transaction'][cols]

bestsellers\_purchase **=** bestsellers\_purchase.loc[bestsellers\_purchase.itemid.isin(top\_20\_items)]

bestsellers\_purchase['top\_20\_purch'] **=** 1

bestsellers\_purchase **=** bestsellers\_purchase.groupby(by**=**'visitorid').sum().reset\_index()

bestsellers\_purchase **=** bestsellers\_purchase[['visitorid', 'top\_20\_purch']]

​

*#Adding to frame*

user\_features **=** user\_features.merge(bestsellers\_purchase, on**=**'visitorid', how**=**'left')

​

*#Monthly\_top (10 items frequency)*

*#Selecting the most popular items for every month*

month\_list **=** user\_features.month.unique().tolist()

​

top\_10\_5 **=** user\_features.loc[(user\_features.event**==**'transaction')**&**(user\_features.month**==**5)]['itemid'].value\_counts()[:10].index.tolist()

top\_10\_6 **=** user\_features.loc[(user\_features.event**==**'transaction')**&**(user\_features.month**==**6)]['itemid'].value\_counts()[:10].index.tolist()

top\_10\_7 **=** user\_features.loc[(user\_features.event**==**'transaction')**&**(user\_features.month**==**7)]['itemid'].value\_counts()[:10].index.tolist()

top\_10\_8 **=** user\_features.loc[(user\_features.event**==**'transaction')**&**(user\_features.month**==**8)]['itemid'].value\_counts()[:10].index.tolist()

top\_10\_9 **=** user\_features.loc[(user\_features.event**==**'transaction')**&**(user\_features.month**==**8)]['itemid'].value\_counts()[:10].index.tolist()

​

full\_month\_top **=** list(set(top\_10\_5**+**top\_10\_6**+**top\_10\_7**+**top\_10\_8**+**top\_10\_9))

​

cols **=** ['visitorid', 'itemid']

month\_top\_purchase **=** user\_features.loc[user\_features.event**==**'transaction'][cols]

month\_top\_purchase **=** month\_top\_purchase.loc[month\_top\_purchase.itemid.isin(full\_month\_top)]

month\_top\_purchase['month\_top\_purch'] **=** 1

month\_top\_purchase **=** month\_top\_purchase.groupby(by**=**'visitorid').sum().reset\_index()

month\_top\_purchase **=** month\_top\_purchase[['visitorid', 'month\_top\_purch']]

​

*#Adding to frame*

user\_features **=** user\_features.merge(month\_top\_purchase, on**=**'visitorid', how**=**'left')

[89]:

*#Activity duration*

cols **=** ['visitorid', 'format\_date', 'month', 'day\_week', 'hour']

user\_time **=** user\_features[cols]

​

user\_time **=** user\_time.groupby('visitorid')

​

*#Avtivity period*

first\_date **=** user\_time.min().format\_date.to\_frame().reset\_index()

first\_date **=** first\_date.rename(columns**=**{'format\_date':'first\_date\_vis'})

last\_date **=** user\_time.max().format\_date.to\_frame().reset\_index()

last\_date **=** last\_date.rename(columns**=**{'format\_date':'last\_date\_vis'})

​

*#Adding to data*

user\_features **=** user\_features.merge(first\_date, how**=**'left', on**=**'visitorid')

user\_features **=** user\_features.merge(last\_date, how**=**'left', on**=**'visitorid')

​

*#Counting time delta*

user\_features['activity\_duration\_vis'] **=** user\_features['last\_date\_vis'] **-** user\_features['first\_date\_vis']

​

*#Preferred months, days and hours*

*#Purchases filtration (event == 'transaction')*

cols **=** ['visitorid', 'event', 'format\_date', 'month', 'day\_week', 'hour']

user\_time **=** user\_features[cols]

user\_time **=** user\_time.loc[user\_time.event**==**'transaction']

user\_time **=** user\_time.groupby('visitorid')

​

month\_pref **=** user\_time.month.agg(**lambda** x: scipy.stats.mode(x)[0]).reset\_index()

month\_pref **=** month\_pref.rename(columns**=**{'month':'month\_vis'})

​

day\_pref **=** user\_time.day\_week.agg(**lambda** x: scipy.stats.mode(x)[0]).reset\_index()

day\_pref **=** day\_pref.rename(columns**=**{'day\_week':'day\_vis'})

​

hour\_pref **=** user\_time.hour.agg(**lambda** x: scipy.stats.mode(x)[0]).reset\_index()

hour\_pref **=** hour\_pref.rename(columns**=**{'hour':'hour\_vis'})

​

*#Adding to data*

user\_features **=** user\_features.merge(month\_pref, how**=**'left', on**=**'visitorid')

user\_features **=** user\_features.merge(day\_pref, how**=**'left', on**=**'visitorid')

user\_features **=** user\_features.merge(hour\_pref, how**=**'left', on**=**'visitorid')

​

user\_features['activity\_vis\_days'] **=** user\_features['activity\_duration\_vis'].dt.days

​

*#We count as activity even only one day with the same date (where timedelta is 0)*

user\_features['activity\_vis\_days'] **=** user\_features['activity\_vis\_days']**+**1

​

*#Number of active days*

cols **=** ['visitorid', 'format\_date']

active\_days **=** user\_features[cols]

active\_days['date'] **=** active\_days['format\_date'].dt.date

active\_days **=** active\_days.groupby(by**=**['visitorid', 'date']).sum()

active\_days['use\_days'] **=** 1

active\_days **=** active\_days.groupby(by**=**'visitorid').sum().reset\_index()

​

*#Adding to data*

user\_features **=** user\_features.merge(active\_days, how**=**'left', on**=**'visitorid')

​

*#Activity days proportion*

user\_features['use\_days\_%'] **=** round((user\_features['use\_days']**/**user\_features['activity\_vis\_days']),2)

​

user\_features['correction'] **=** user\_features['use\_days\_%'].apply(**lambda** x: x **if** x**!=**2 **else** 1)

user\_features.drop(columns**=**['use\_days\_%'], axis**=**1, inplace**=True**)

user\_features **=** user\_features.rename(columns**=**{'correction': 'use\_days\_%'})

[90]:

user\_features

user\_features.to\_csv('user\_features.csv', header**=True**)

[91]:

*#Features*

feat\_list **=** ['buy\_%',

'view\_trans\_vis',

'add\_trans\_vis',

'view\_purch\_vis',

'add\_purch\_vis',

'items\_per\_trans',

'repeat\_count',

'top\_20\_purch',

'month\_top\_purch',

'month\_vis',

'day\_vis',

'hour\_vis',

'activity\_vis\_days',

'use\_days',

'use\_days\_%']

[92]:

*#Visualization*

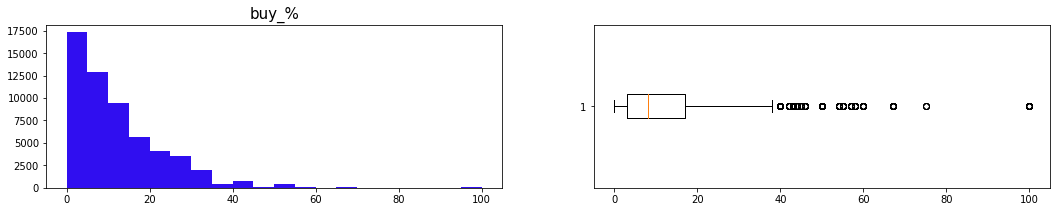
*#Colors*

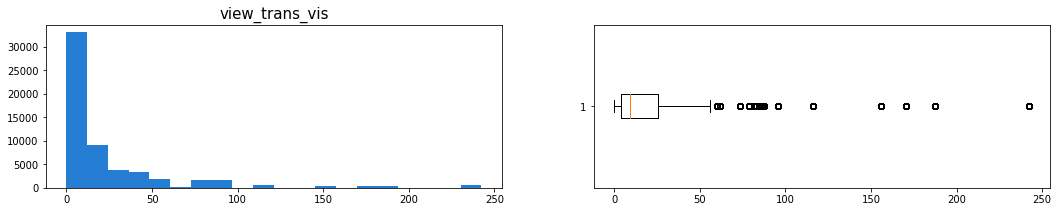
clr\_list **=** ['#300ef0', '#267dd4', '#5ae3f2']

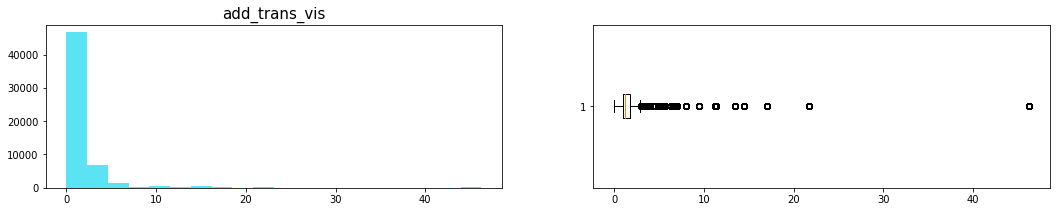
df\_list **=** [user\_features]

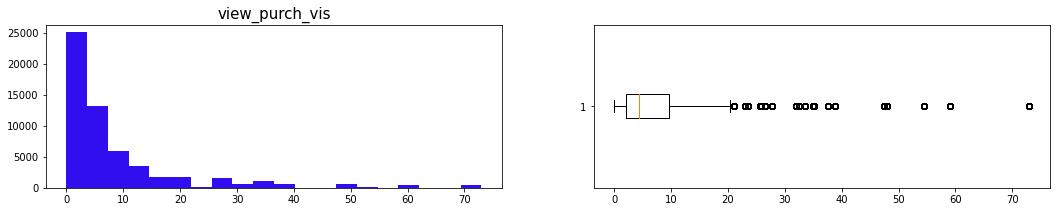
**for** df,feat,clr **in** zip(cycle(df\_list), feat\_list, cycle(clr\_list)):

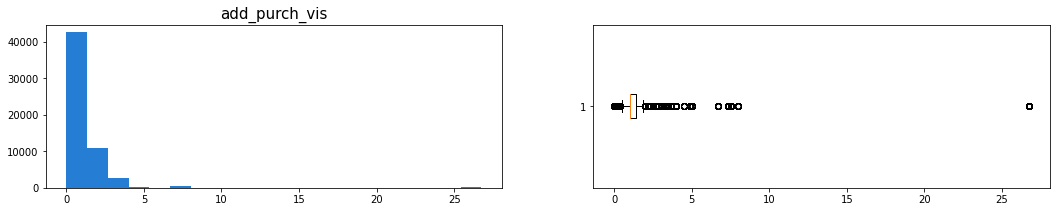
feature\_distribution(df, feat, clr)

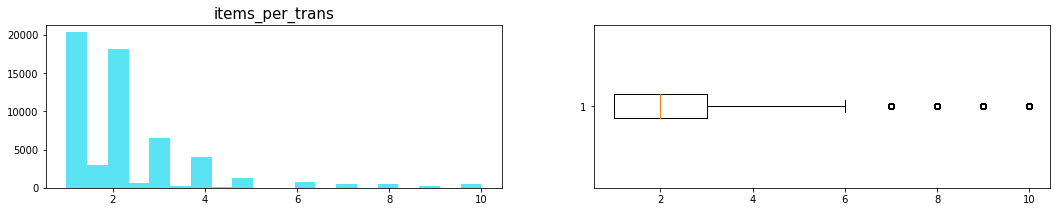


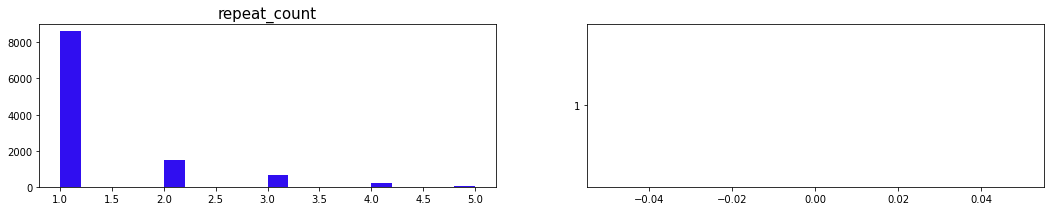


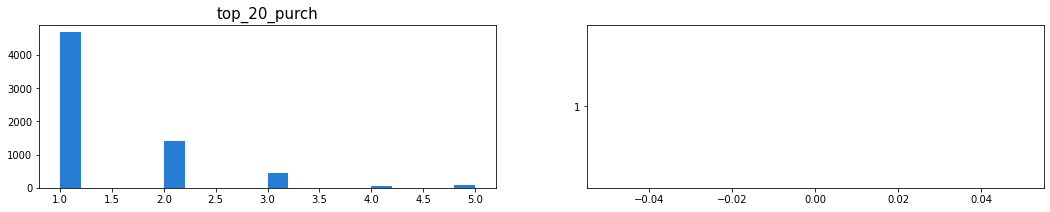


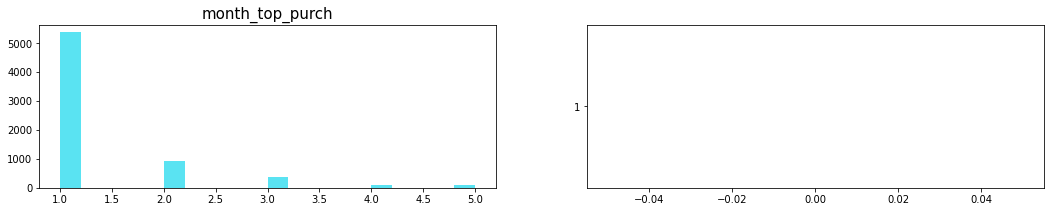


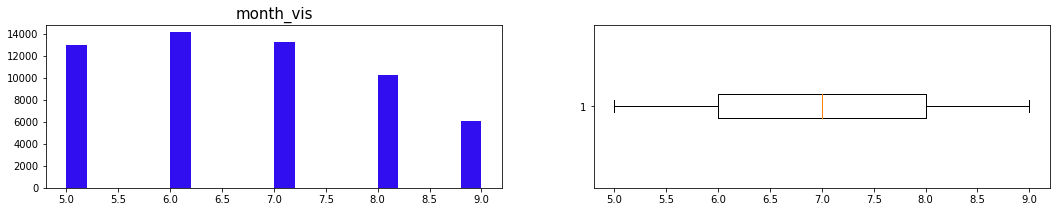


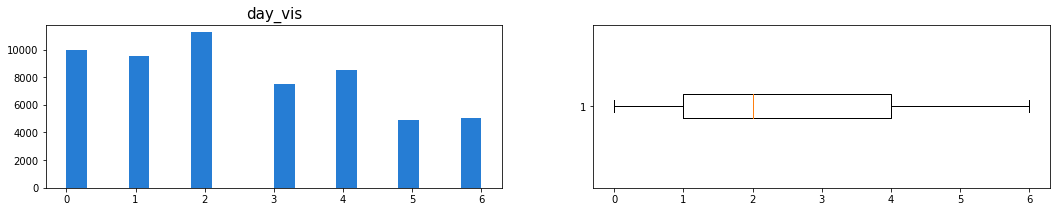


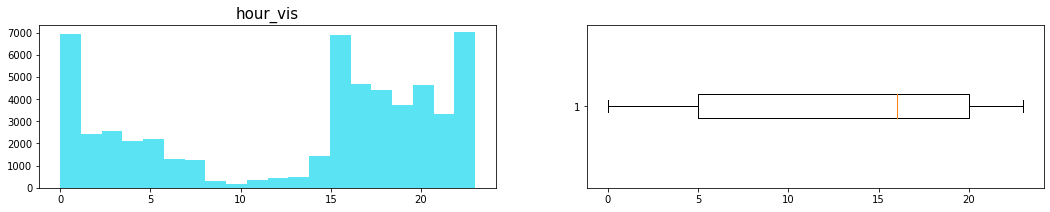


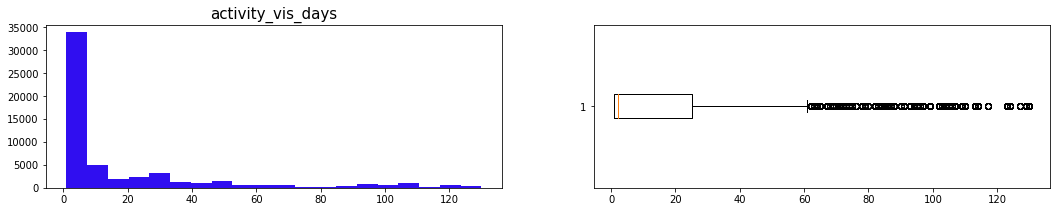


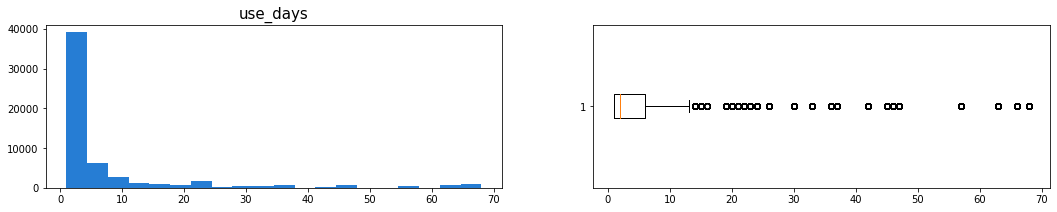


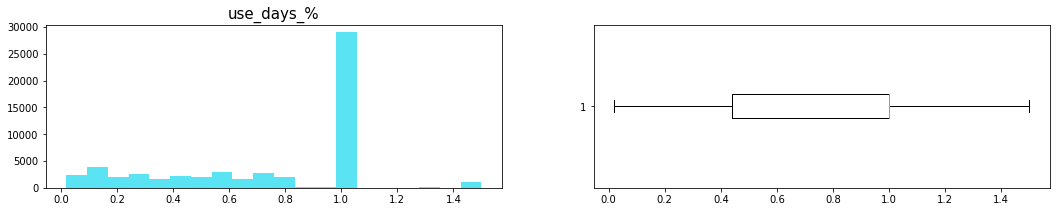












Судя по большому количеству близких к единичным покупок (от 2), малому количеству повторяющихся покупок, сравнительно малому времени, которое пользователи уделяют просмотру товаров перед покупкой, а также большой доле дней, во время которых совершалась покупка и эта же покупка была единственной во всем периоде активности (use\_days\_% = 1), существенная доля всех пользователей приходила в систему для того, чтобы купить заранее намеченные хиты. Похожую картину мы видели выше, когда проводили полный обзор датасета до разделения на Train & Test. Иными словами, общий тренд - единичные покупки хитов.

### **ITEM features**

#### **1. Time Features / Временные признаки**

1.1. month / месяц

1.2. day\_week / день недели

1.3. hour / час дня

#### **2. Events Features / Признаки действий**

2.1. add\_item / суммарное количество добавлений артикула (товара) в корзину

2.2. trans\_item / суммарное количество покупок артикула

2.3. view\_item / суммарное количество просмотров артикула

2.4. all\_events\_itm / общее количество всех действий по отношению к артикулу (сумма)

2.5. sale\_% / доля продаж по отношению к общему количеству всех действий с товаром

#### **3. Sales Frequency Features / Признаки частотности продаж**

3.1. repeat\_count / количество повторных покупок артикула

#### **4. Bestsellers Features / Признаки бестселлеров**

4.1. top\_20\_sale / входит ли артикул в ТОП-20 (20 наиболее часто продававшихся товаров)

4.2. top\_month\_sale /входит ли артикул в общий список бестселлеров месяца (10 наиболее часто продававшихся товаров, общий список уникальных артикулов для каждого месяца из периода наблюдений в датасете)

#### **5. Activity Features / Признаки периодов наиболее активного взаимодействия пользователя с товарами**

5.1. activity\_duration\_itm / длительность периода активности (исчисляется в днях, от первого до последнего дня взаимодействия пользователя с товаром в системе)

5.2. month\_itm / месяц наиболее частых продаж артикула

5.3. day\_itm / день недели, в который артикул чаще всего продавался

5.4. hour\_itm / час дня, в который артикул чаще всего продавался

5.5. sale\_days / количество дней, в которые совершалась продажа артикула

5.6. sale\_day\_s% / доля результативных дней, в которые совершались продажи артикула, по отношению ко всему периоду активности (например, покупатели просматривали артикул в течение 5 дней, но покупали только 2 дня из них, 2/5\*100% - 40% были результативными для системы)

[93]:

item\_features **=** train.copy()

item\_features['month'] **=** pd.DatetimeIndex(item\_features['format\_date']).month

item\_features['day\_week'] **=** pd.DatetimeIndex(item\_features['format\_date']).dayofweek

item\_features['hour'] **=** pd.DatetimeIndex(item\_features['format\_date']).hour

[94]:

*#Pivoting*

cols **=** ['itemid', 'event']

item\_events **=** train[cols]

item\_events['count'] **=** 1

​

item\_events **=** pd.pivot\_table(item\_events,

values**=**'count',

index**=**['itemid'],

columns**=**['event'], aggfunc**=**np.sum, fill\_value**=**0)

*#Adding features*

item\_events **=** item\_events.reset\_index()

item\_events['all\_events\_itm'] **=** item\_events['addtocart']**+**item\_events['transaction']**+**item\_events['view']

item\_events['sale\_%'] **=** round((item\_events['transaction']**/**item\_events['all\_events\_itm']**\***100))

​

item\_events **=** item\_events.rename(columns**=**{'addtocart': 'add\_item',

'transaction': 'trans\_item',

'view': 'view\_item'})

​

*#Adding to frame*

item\_features **=** item\_features.merge(item\_events, on**=**'itemid', how**=**'left')

[95]:

*#Repeat sale*

cols **=** ['visitorid', 'itemid']

repeat\_sale **=** item\_features.loc[user\_features.event**==**'transaction'][cols]

repeat\_sale['items\_count'] **=** 1

repeat\_sale **=** repeat\_sale.groupby(['itemid', 'visitorid']).sum()

repeat\_sale **=** repeat\_sale.loc[repeat\_sale['items\_count']**>**1]

repeat\_sale['repeat\_count'] **=** 1

repeat\_sale **=** repeat\_sale.groupby(by**=**'itemid').sum().reset\_index()

repeat\_sale **=** repeat\_sale[['itemid', 'repeat\_count']]

​

*#Adding to frame*

item\_features **=** item\_features.merge(repeat\_sale, on**=**'itemid', how**=**'left')

[96]:

*#Top\_20 (frequency)*

*### Top\_20 Products*

top\_20\_items **=** train.loc[train.event**==**'transaction']['itemid'].value\_counts()[:20].index.tolist()

*### Make a flag for top-20 products*

cols **=** ['visitorid', 'itemid']

bestsellers\_20 **=** item\_features[cols].drop\_duplicates().reset\_index(drop**=True**)

bestsellers\_20 **=** bestsellers\_20.loc[bestsellers\_20.itemid.isin(top\_20\_items)]

bestsellers\_20['top\_20\_sale'] **=** 1

​

bestsellers\_20.drop('visitorid', axis**=**1,inplace**=True**)

bestsellers\_20 **=** bestsellers\_20.drop\_duplicates()

​

*#Adding to the frame*

item\_features **=** item\_features.merge(bestsellers\_20, on**=**'itemid', how**=**'left')

​

*#Monthly\_top (10 items frequency)*

cols **=** ['visitorid', 'itemid']

month\_bestsellers **=** item\_features[cols].drop\_duplicates().reset\_index(drop**=True**)

month\_bestsellers **=** month\_bestsellers.loc[month\_bestsellers.itemid.isin(full\_month\_top)]

month\_bestsellers['top\_month\_sale'] **=** 1

​

month\_bestsellers.drop('visitorid', axis**=**1,inplace**=True**)

month\_bestsellers **=** month\_bestsellers.drop\_duplicates()

​

*#Adding to the frame*

item\_features **=** item\_features.merge(month\_bestsellers, on**=**'itemid', how**=**'left')

[97]:

*#Activity duration*

cols **=** ['itemid', 'format\_date', 'month', 'day\_week', 'hour']

item\_time **=** item\_features[cols]

​

item\_time **=** item\_time.groupby('itemid')

​

*#Avtivity period*

first\_date **=** item\_time.min().format\_date.to\_frame().reset\_index()

first\_date **=** first\_date.rename(columns**=**{'format\_date':'first\_date\_itm'})

last\_date **=** item\_time.max().format\_date.to\_frame().reset\_index()

last\_date **=** last\_date.rename(columns**=**{'format\_date':'last\_date\_itm'})

​

*#Adding to data*

item\_features **=** item\_features.merge(first\_date, how**=**'left', on**=**'itemid')

item\_features **=** item\_features.merge(last\_date, how**=**'left', on**=**'itemid')

​

*#Counting time delta*

item\_features['activity\_duration\_itm'] **=** item\_features['last\_date\_itm'] **-** item\_features['first\_date\_itm']

​

*###Preferred months, days and hours*

*#Sales filtration (event == 'transaction')*

cols **=** ['itemid', 'event', 'format\_date', 'month', 'day\_week', 'hour']

item\_time **=** item\_features[cols]

item\_time **=** item\_time.loc[item\_time.event**==**'transaction']

item\_time **=** item\_time.groupby('itemid')

​

month\_pref **=** item\_time.month.agg(**lambda** x: scipy.stats.mode(x)[0]).reset\_index()

month\_pref **=** month\_pref.rename(columns**=**{'month':'month\_itm'})

​

day\_pref **=** item\_time.day\_week.agg(**lambda** x: scipy.stats.mode(x)[0]).reset\_index()

day\_pref **=** day\_pref.rename(columns**=**{'day\_week':'day\_itm'})

​

hour\_pref **=** item\_time.hour.agg(**lambda** x: scipy.stats.mode(x)[0]).reset\_index()

hour\_pref **=** hour\_pref.rename(columns**=**{'hour':'hour\_itm'})

​

*#Adding to data*

item\_features **=** item\_features.merge(month\_pref, how**=**'left', on**=**'itemid')

item\_features **=** item\_features.merge(day\_pref, how**=**'left', on**=**'itemid')

item\_features **=** item\_features.merge(hour\_pref, how**=**'left', on**=**'itemid')

​

item\_features['activity\_itm\_days'] **=** item\_features['activity\_duration\_itm'].dt.days

​

*#We count as activity even only one day with the same date (where timedelta is 0)*

item\_features['activity\_itm\_days'] **=** item\_features['activity\_itm\_days']**+**1

​

*#Number of active sale days*

cols **=** ['itemid', 'format\_date']

active\_days **=** item\_features[cols]

active\_days['date'] **=** active\_days['format\_date'].dt.date

active\_days **=** active\_days.groupby(by**=**['itemid', 'date']).sum()

active\_days['sale\_days'] **=** 1

active\_days **=** active\_days.groupby(by**=**'itemid').sum().reset\_index()

​

*#Adding to data*

item\_features **=** item\_features.merge(active\_days, how**=**'left', on**=**'itemid')

​

*#Activity days proportion*

item\_features['sale\_days\_%'] **=** round((item\_features['sale\_days']**/**item\_features['activity\_itm\_days']),2)

​

item\_features['correction'] **=** item\_features['sale\_days\_%'].apply(**lambda** x: x **if** x**!=**2 **else** 1)

item\_features.drop(columns**=**['sale\_days\_%'], axis**=**1, inplace**=True**)

item\_features **=** item\_features.rename(columns**=**{'correction': 'sale\_days\_%'})

[99]:

item\_feat\_list **=** ['add\_item',

'trans\_item',

'view\_item',

'all\_events\_itm',

'sale\_%',

'repeat\_count',

'month\_itm',

'day\_itm',

'hour\_itm',

'activity\_itm\_days',

'sale\_days',

'sale\_days\_%']

[100]:

item\_features[item\_feat\_list].fillna(0)

[100]:

|  | **add\_item** | **trans\_item** | **view\_item** | **all\_events\_itm** | **sale\_%** | **repeat\_count** | **month\_itm** | **day\_itm** | **hour\_itm** | **activity\_itm\_days** | **sale\_days** | **sale\_days\_%** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 1 | 1 | 5 | 7 | 14.0 | 0.0 | 6.0 | 1.0 | 5.0 | 15 | 2 | 0.13 |
| **1** | 0 | 2 | 11 | 13 | 15.0 | 1.0 | 6.0 | 1.0 | 6.0 | 22 | 3 | 0.14 |
| **2** | 1 | 0 | 4 | 5 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1 | 1 | 1.00 |
| **3** | 5 | 5 | 34 | 44 | 11.0 | 0.0 | 5.0 | 0.0 | 20.0 | 61 | 14 | 0.23 |
| **4** | 0 | 0 | 1 | 1 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1 | 1 | 1.00 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **56728** | 2 | 0 | 0 | 2 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1 | 1 | 1.00 |
| **56729** | 2 | 0 | 2 | 4 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2 | 2 | 1.00 |
| **56730** | 3 | 1 | 7 | 11 | 9.0 | 0.0 | 7.0 | 3.0 | 16.0 | 23 | 4 | 0.17 |
| **56731** | 0 | 0 | 1 | 1 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1 | 1 | 1.00 |
| **56732** | 0 | 1 | 8 | 9 | 11.0 | 0.0 | 9.0 | 3.0 | 1.0 | 125 | 6 | 0.05 |

56733 rows × 12 columns

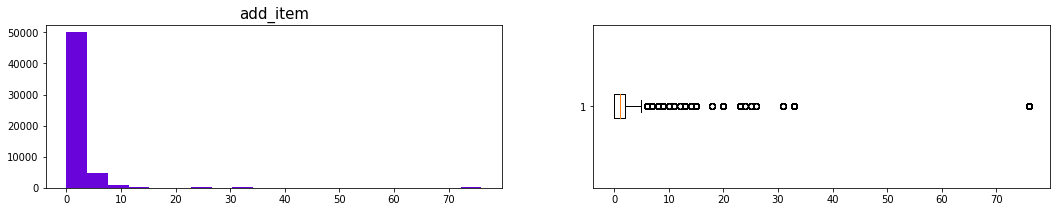
[101]:

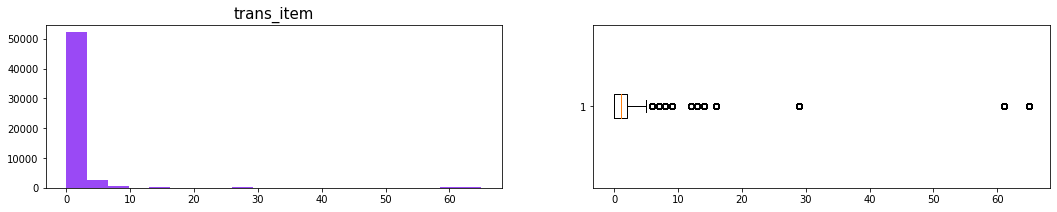
clr\_list **=** ['#6904db', '#9a49f5', '#ba5df0']

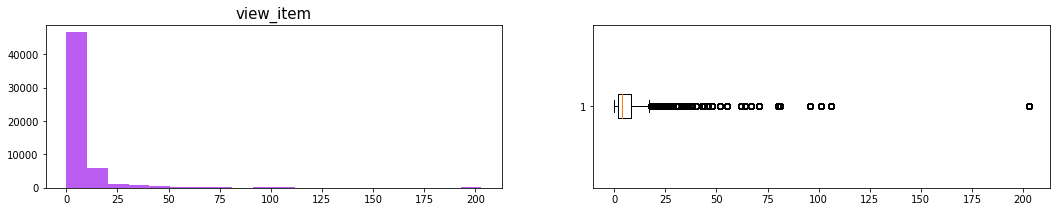
df\_list **=** [item\_features]

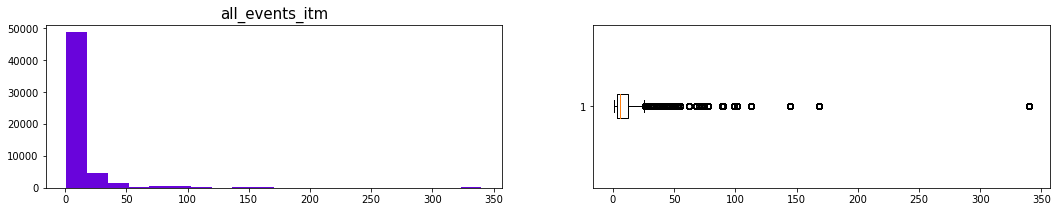
**for** df,feat,clr **in** zip(cycle(df\_list), item\_feat\_list, cycle(clr\_list)):

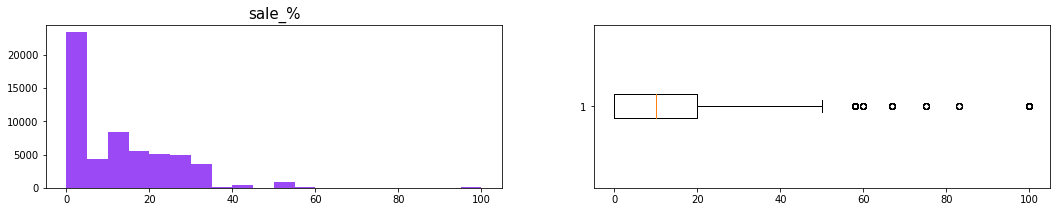
feature\_distribution(df, feat, clr)



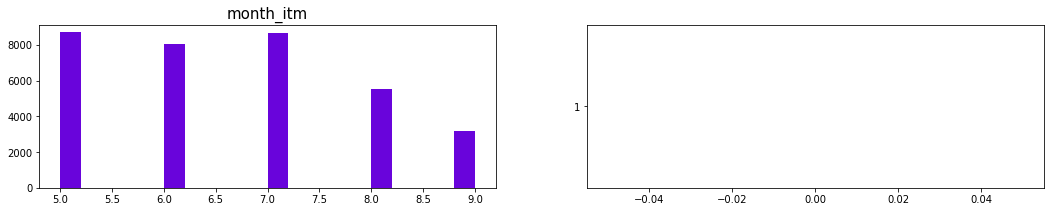


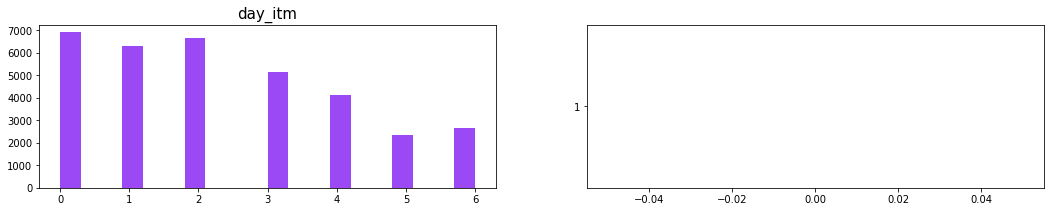




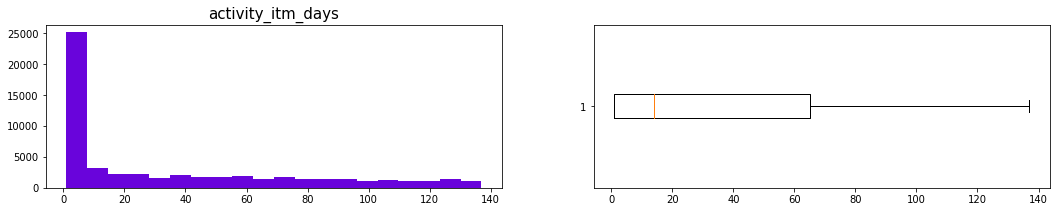


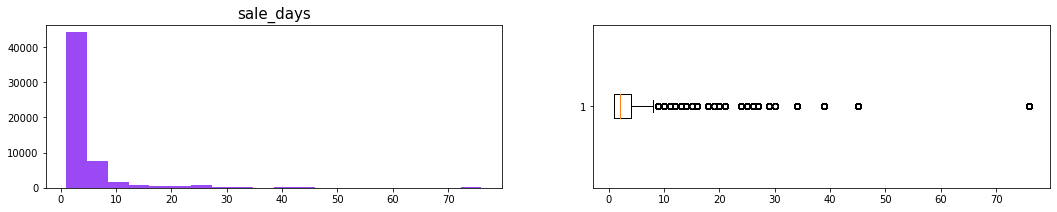


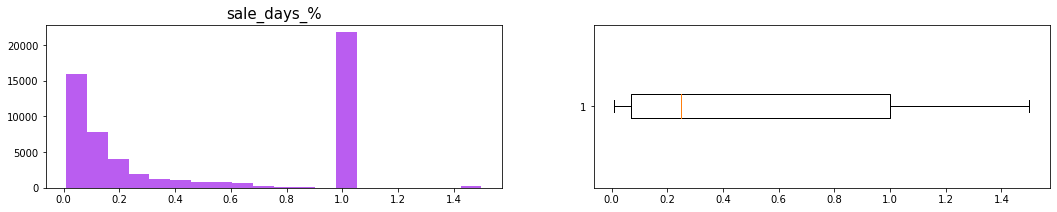












add Codeadd Markdown

[102]:

*### Preprocessing*

*#Items list*

items\_list **=** item\_features.itemid.unique().tolist()

​

*#Selecting properties*

items\_1 **=** items\_1.loc[items\_1.itemid.isin(items\_list)]

items\_2 **=** items\_2.loc[items\_2.itemid.isin(items\_list)]

​

*#Concatenating all selected*

items **=** pd.concat([items\_1, items\_2]).reset\_index(drop**=True**).sort\_values(by**=**'itemid')

​

*#Let's check the null values*

items.loc[items.value**==**'0']

[102]:

|  | **itemid** | **property** | **value** |
| --- | --- | --- | --- |
| **319118** | 15 | available | 0 |
| **265537** | 15 | available | 0 |
| **211148** | 17 | available | 0 |
| **449898** | 19 | available | 0 |
| **314456** | 25 | available | 0 |
| **...** | ... | ... | ... |
| **312346** | 466736 | available | 0 |
| **210090** | 466747 | available | 0 |
| **281257** | 466759 | available | 0 |
| **540166** | 466796 | available | 0 |
| **2474** | 466796 | available | 0 |

12584 rows × 3 columns

[103]:

*#Skip 'available' value*

items **=** items.loc[items.value**!=**'0']

[104]:

display(tree[tree.categoryid**==**112])

|  | **categoryid** | **parentid** |
| --- | --- | --- |
| **1660** | 112.0 | 381.0 |

[105]:

*### Properties concatenation*

tree **=** tree.rename(columns**=**{'categoryid': 'property'})

items **=** items.loc[(items.property**!=**'categoryid')**&**(items.property**!=**'available')]

items['property'] **=** pd.to\_numeric(items['property'])

items\_full **=** items.merge(tree, how**=**'left', on**=**'property').fillna(0)

items\_full.tail(10)

​

[105]:

|  | **itemid** | **property** | **value** | **parentid** |
| --- | --- | --- | --- | --- |
| **524123** | 466796 | 713 | 1090966 | 1615.0 |
| **524124** | 466796 | 790 | n26760.000 | 1492.0 |
| **524125** | 466796 | 689 | 471783 827388 | 207.0 |
| **524126** | 466796 | 227 | 1214748 1186610 | 602.0 |
| **524127** | 466796 | 364 | 852741 | 711.0 |
| **524128** | 466796 | 28 | 150169 610517 | 1604.0 |
| **524129** | 466796 | 558 | 1214748 1186610 | 92.0 |
| **524130** | 466796 | 283 | 1214748 1186610 368078 1229584 692527 1214748 ... | 1579.0 |
| **524131** | 466796 | 159 | 519769 | 1028.0 |
| **524132** | 466796 | 202 | 1229584 692527 | 1008.0 |

[107]:

*#Skip duplicates*

items\_full **=** items\_full.drop\_duplicates()

​

*###Adding properties to all features*

items\_full.sort\_values(by**=**'itemid').head(10)

[107]:

|  | **itemid** | **property** | **value** | **parentid** |
| --- | --- | --- | --- | --- |
| **0** | 15 | 678 | 245772 | 250.0 |
| **23** | 15 | 776 | 604754 | 727.0 |
| **22** | 15 | 283 | 433564 245772 789221 809278 245772 1213953 429... | 1579.0 |
| **21** | 15 | 812 | 769062 | 92.0 |
| **20** | 15 | 764 | 1285872 | 1370.0 |
| **19** | 15 | 790 | n9000.000 | 1492.0 |
| **18** | 15 | 698 | 433564 | 1251.0 |
| **17** | 15 | 917 | 789221 | 1374.0 |
| **16** | 15 | 159 | 519769 | 1028.0 |
| **14** | 15 | 693 | 769062 | 1630.0 |

[108]:

*#Selecting unique item IDs*

items\_df **=** items\_full.loc[items\_full.itemid.isin(items\_list)]

​

*#Making string features*

item\_features['property'] **=** item\_features['itemid'].map(get\_property)

item\_features['value'] **=** item\_features['itemid'].map(get\_value)

item\_features['parentid'] **=** item\_features['itemid'].map(get\_parentid)