**Отчёт по домашнему заданию**

**“Homework\_1.ipynb, Homework\_2.ipynb”**

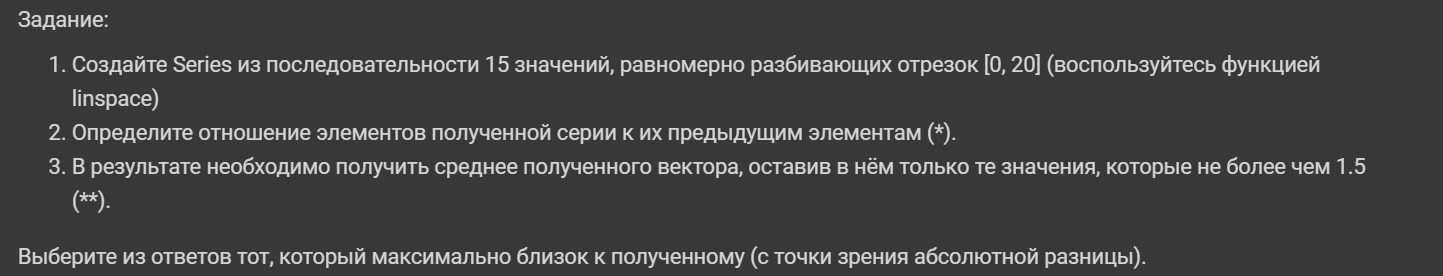
Выполнил:

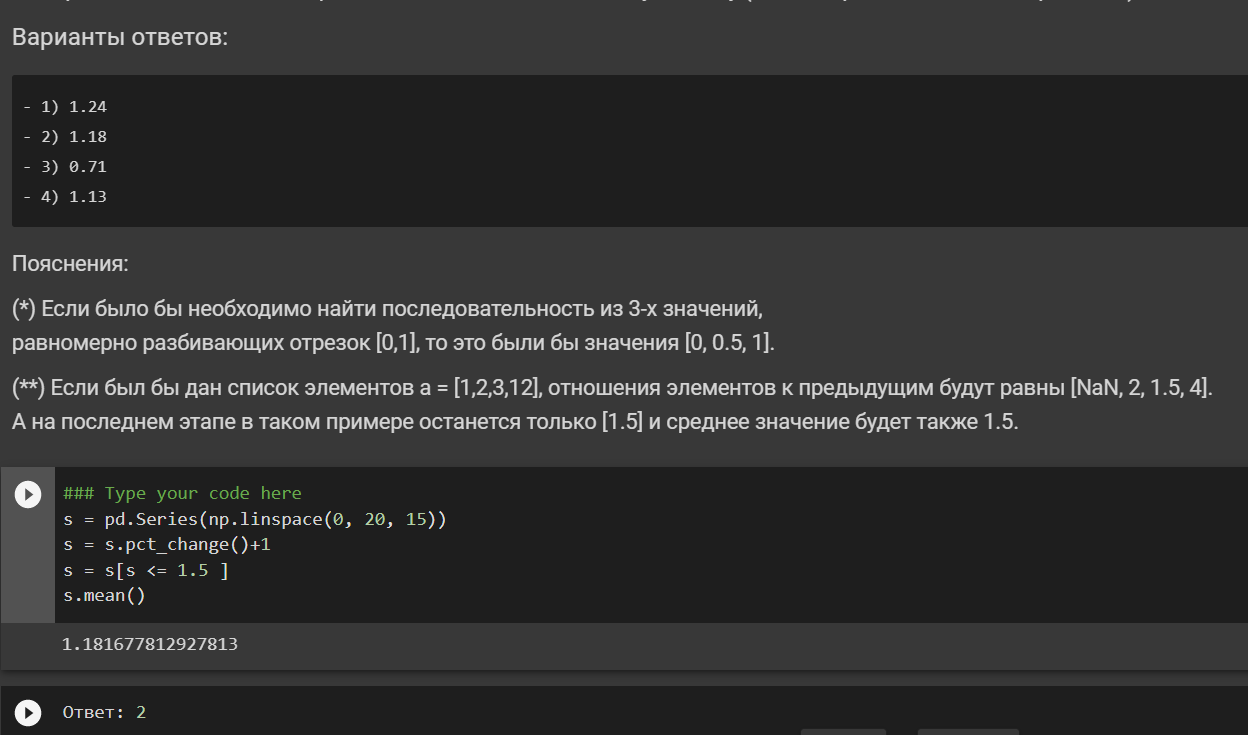
студент БПМ-17-2

Скибин К.С.

**Homework\_1.ipynb**

**TASK 1**





s = pd.Series(np.linspace(0, 20, 15))

Создает Series из последовательности 15 значений, равномерно разбивающих отрезок [0, 20]

s = s.pct\_change()+1

pct\_change() – находит отношение **дельты** к предыдущему, поэтому необходимо добавить ещё 1 (100%)

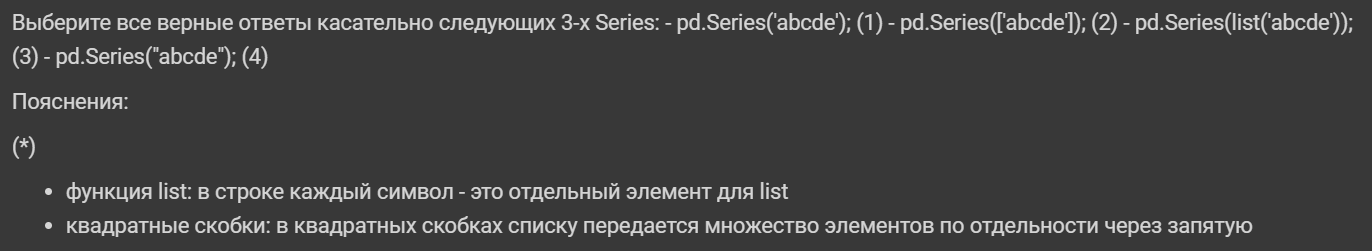
s = s[s <= 1.5 ]

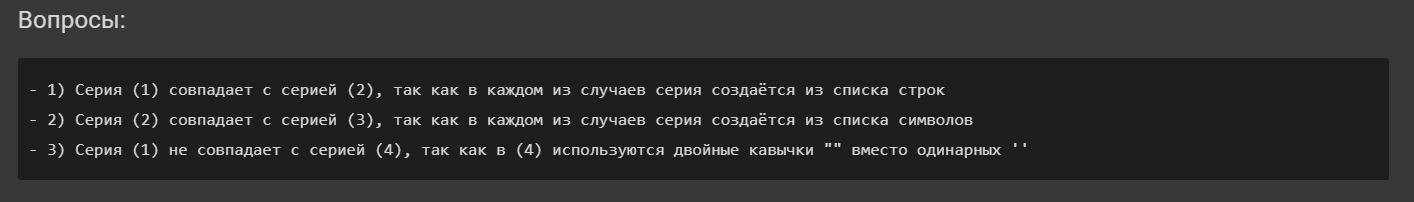
фильтр с помощи индексации

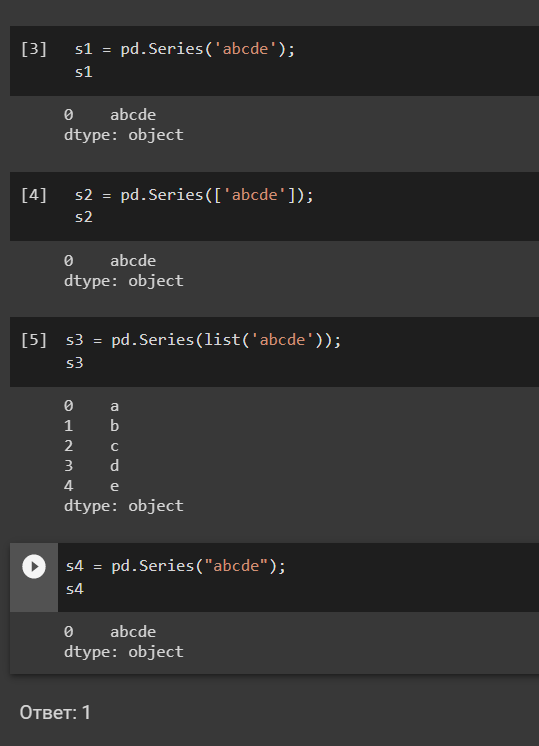
s.mean()

среднее по всем значениям

**TASK 2**





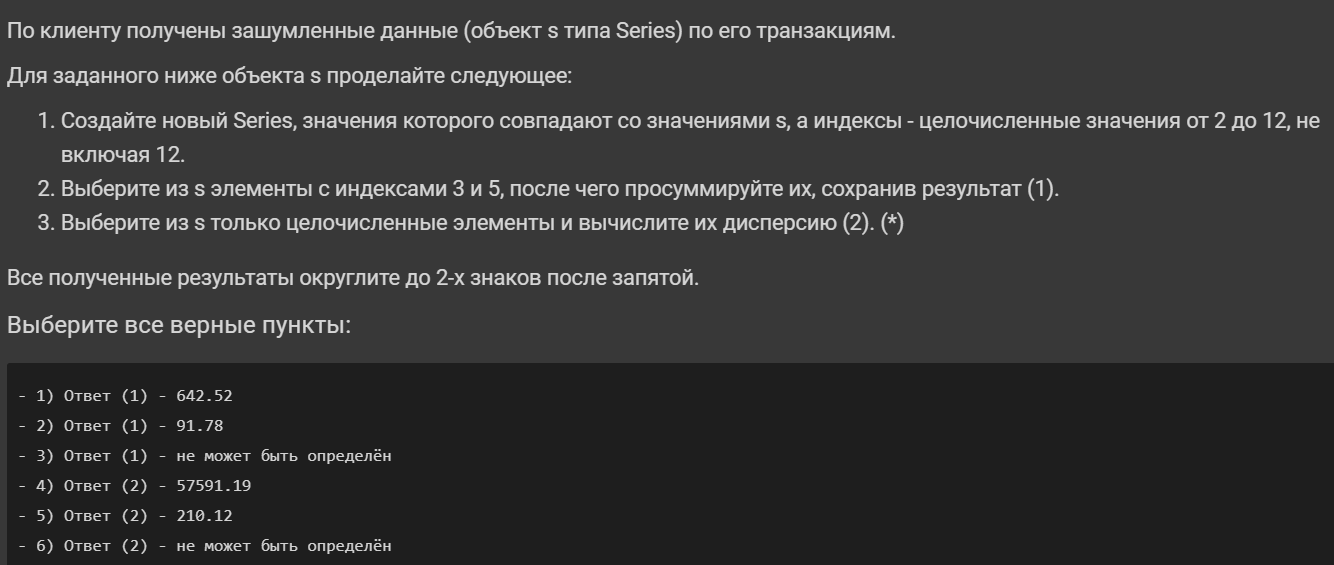


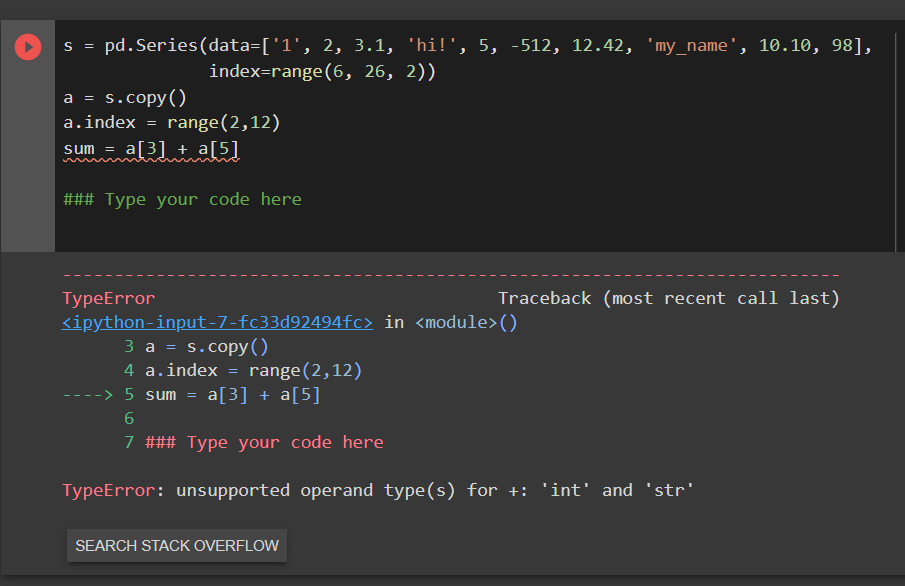
s3 = pd.Series(list('abcde'));

функция list обрабатывает итерируемый объект. В данном случае – строку ‘abcde’ разбивает на отдельные символы

Разницы между одинарными и двойными кавычками при создании Series – нет

**TASK 3**





a = s.copy()

a.index = range(2,12)

создаем копию series при помощи copy(), меняем индексацию, присваивая range(2,12), теперь индексы от 2 до 11 включительно

sum = a[3] + a[5]

a[3] = 2

a[5] = ‘hi!’

ошибка из-за суммирования int и str (оператор на такое не перегружен)

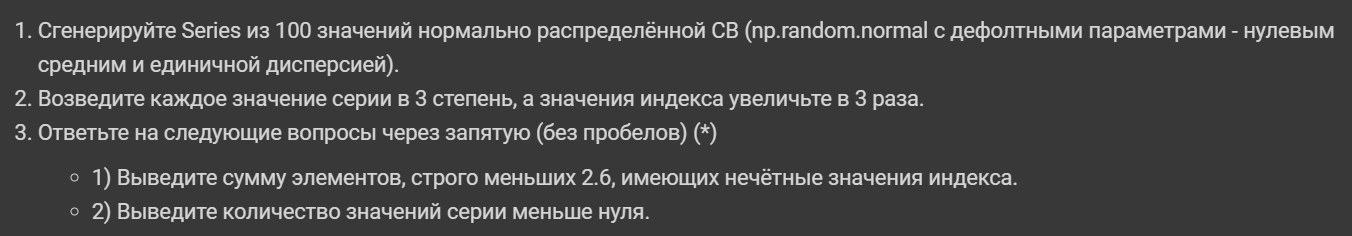
a = a[a.apply(lambda x : isinstance(x, (int, np.uint)))]

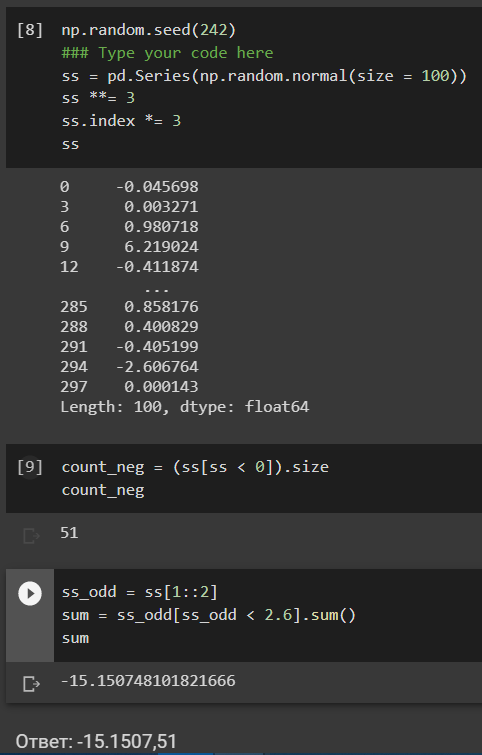
a.var(ddof =0)

через apply применяем итеративно лямбду ко всем элементам, получаем булеан маску, где true – если элемент целочисенный. Данный буллеан массив применяем как фильтр для того же массива

Находим дисперсию при помощи var

**TASK 4**

****



np.random.seed(242)

ss = pd.Series(np.random.normal(size = 100))

ss \*\*= 3

ss.index \*= 3

Выставляем зерно генератора на 242. Создаем Series из 100 элементов, распределенный по стандартному нормальному распределению. Возводим все элементы в третью степень. Умножаем индекс каждого элемента на 3.

count\_neg = (ss[ss < 0]).size

Фильтруем при помощи булеан массива ss < 0 массив ss и считаем сколько в нем элементов при помощи size

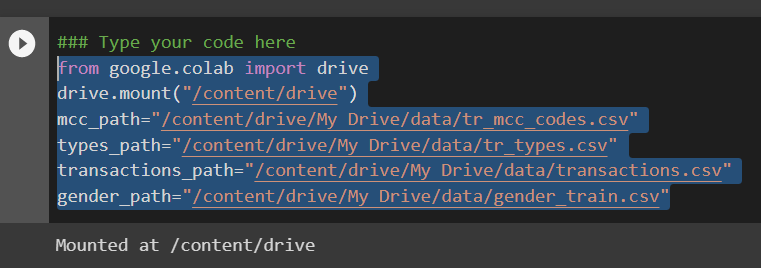
ss\_odd = ss[1::2]

sum = ss\_odd[ss\_odd < 2.6].sum()

Используем индексацию для получения элементов с нечетными индексами. Начинаем с индекса 1 и идём до конца с шагом 2 [1::2]

Фильтруем получившийся массив при помощи ss\_odd < 2.6, суммируем элементы в отфильтрованном массива .sum()

**===================== Загрузка датасетов==================**



from google.colab import drive

drive.mount("/content/drive")

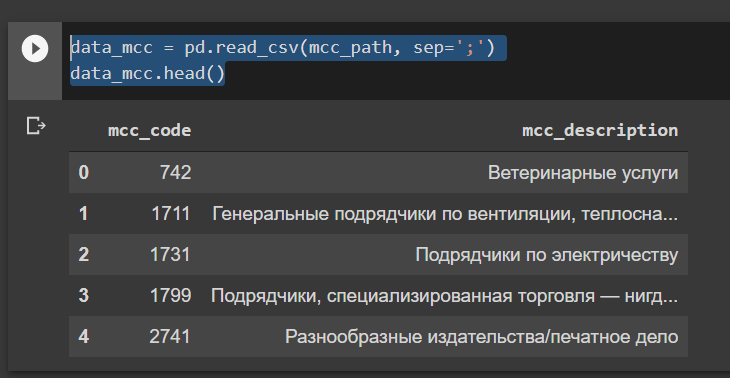
mcc\_path="/content/drive/My Drive/data/tr\_mcc\_codes.csv"

types\_path="/content/drive/My Drive/data/tr\_types.csv"

transactions\_path="/content/drive/My Drive/data/transactions.csv"

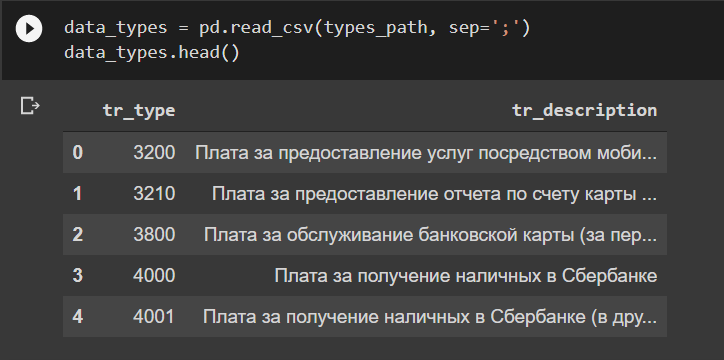
gender\_path="/content/drive/My Drive/data/gender\_train.csv"

Подключаем файловую систему с гугл драйв (путь /content/drive/My Drive/), чтобы загружать оттуда датасеты.

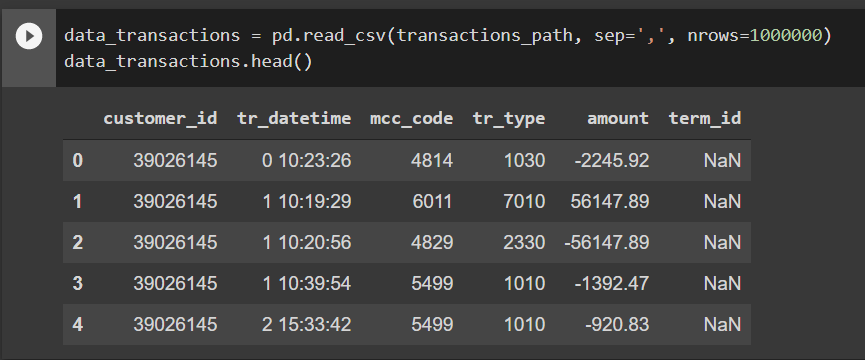


data\_mcc = pd.read\_csv(mcc\_path, sep=';')

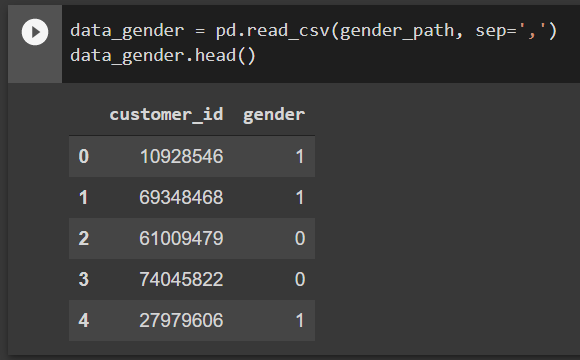
Подгружаем датасет. В качестве разделителя в том файле используется ;, поэтому sep=’;’



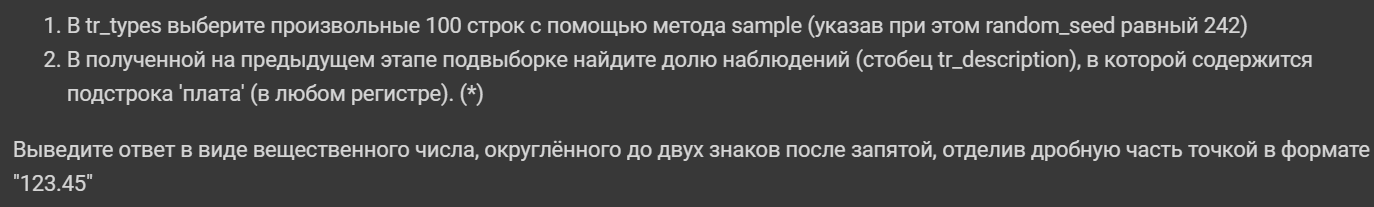
Аналогично

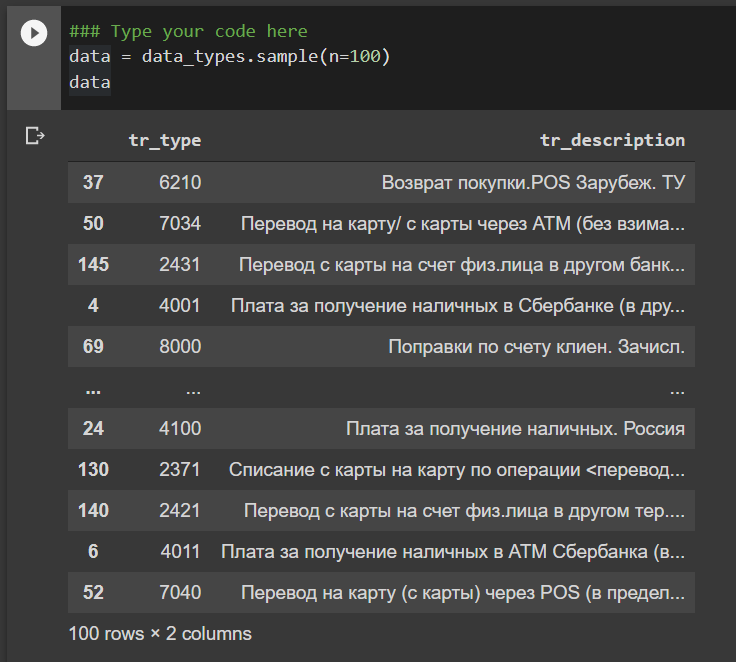


Тут разделитель – запятая. Также нам нужны только первые 1000000 записей, поэтому nrows=1000000

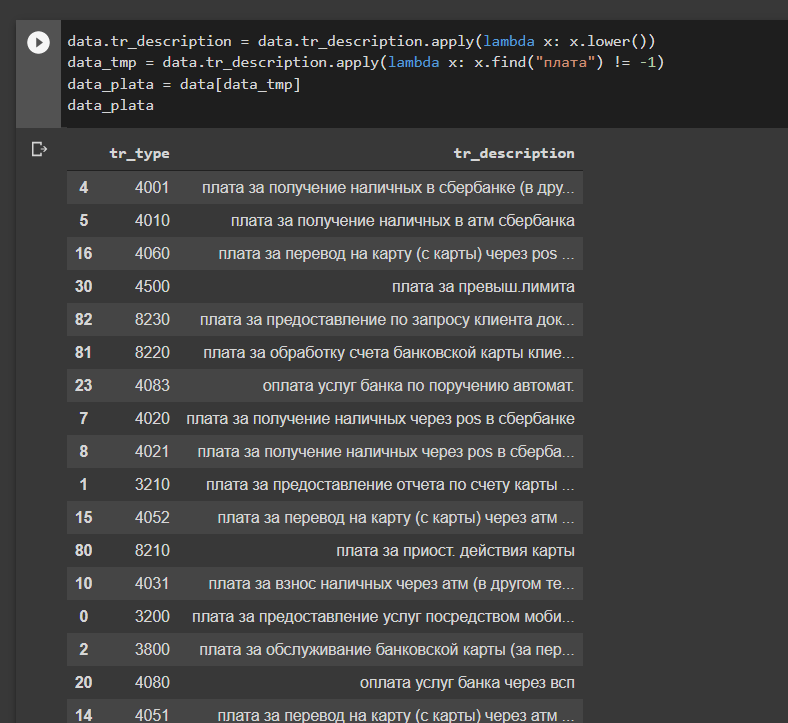


**TASK 5**





Выбираем произвольные 100 строк с помощью метода sample, аргумент n = 100



data.tr\_description = data.tr\_description.apply(lambda x: x.lower())

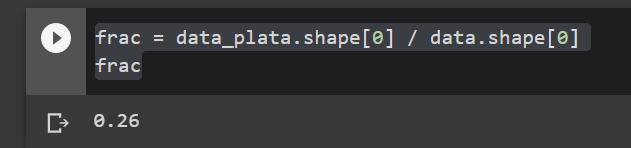
Приводим к нижнему регистру записи

data\_tmp = data.tr\_description.apply(lambda x: x.find("плата") != -1)

Делаем фильтр из строк, где есть подстрока “плата”

data\_plata = data[data\_tmp]

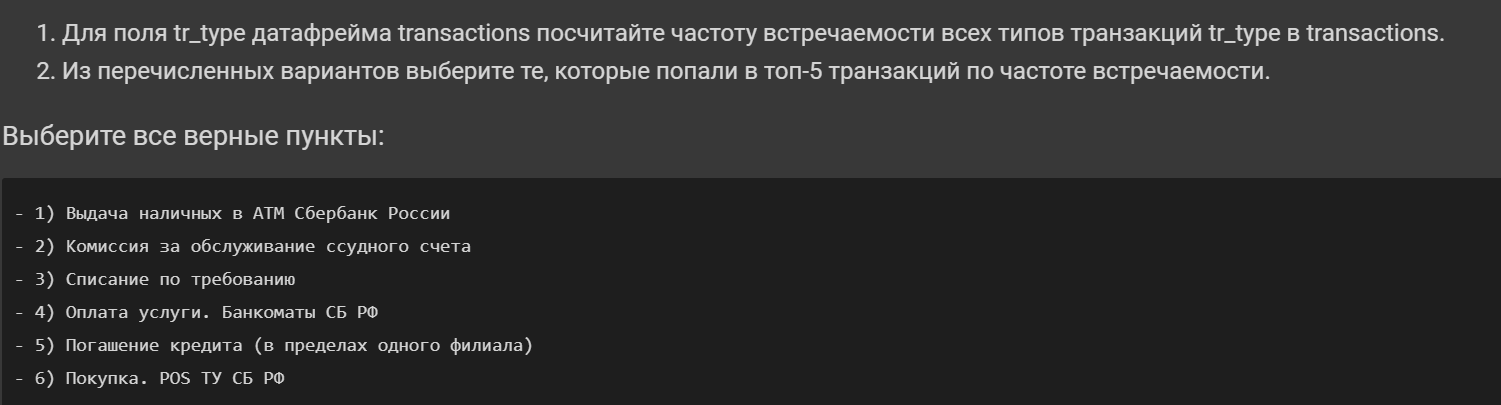
Фильтруем записи

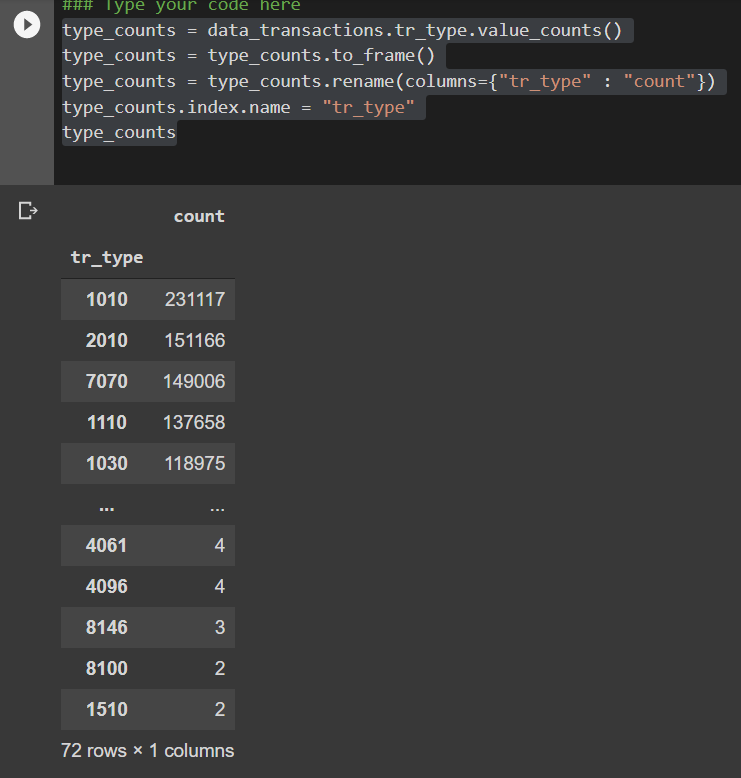


frac = data\_plata.shape[0] / data.shape[0]

Находим долю записей

**TASK 6**





type\_counts = data\_transactions.tr\_type.value\_counts()

Считает количество записей для всех видов tr\_type

type\_counts = type\_counts.to\_frame()

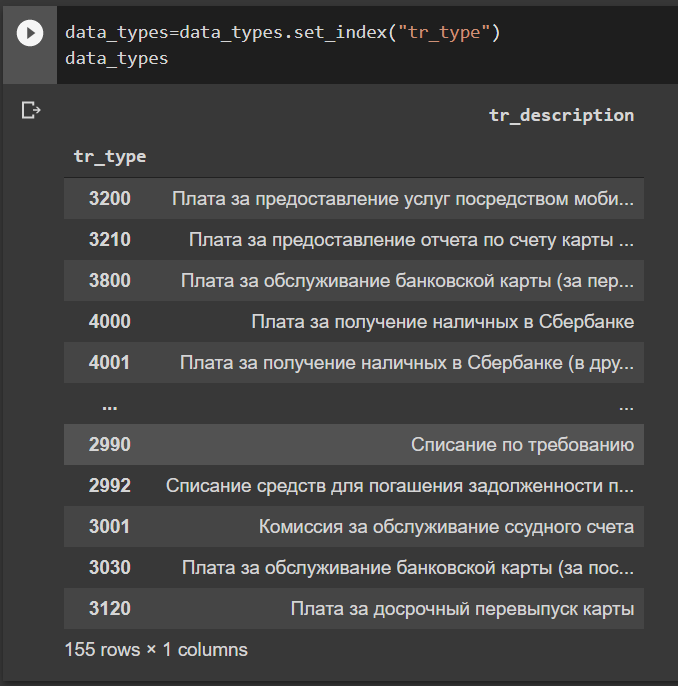
Переводим в DataFrame

type\_counts = type\_counts.rename(columns={"tr\_type" : "count"})

Меняем название у колонок (косметика)

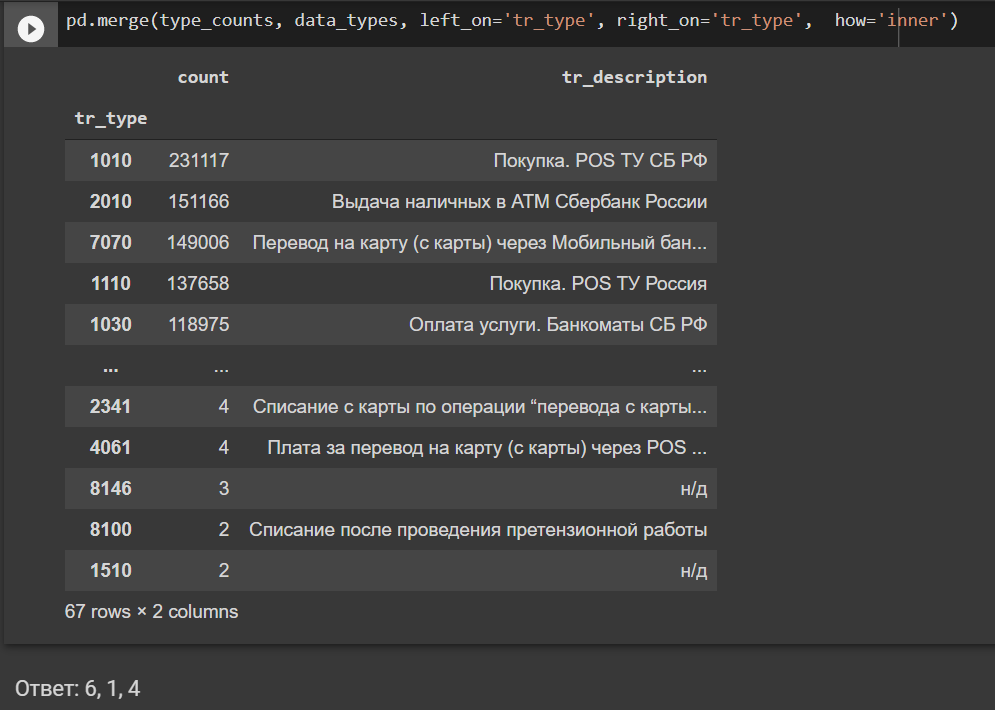
type\_counts.index.name = "tr\_type"

Индексируем по колонке tr\_type



data\_types=data\_types.set\_index("tr\_type")

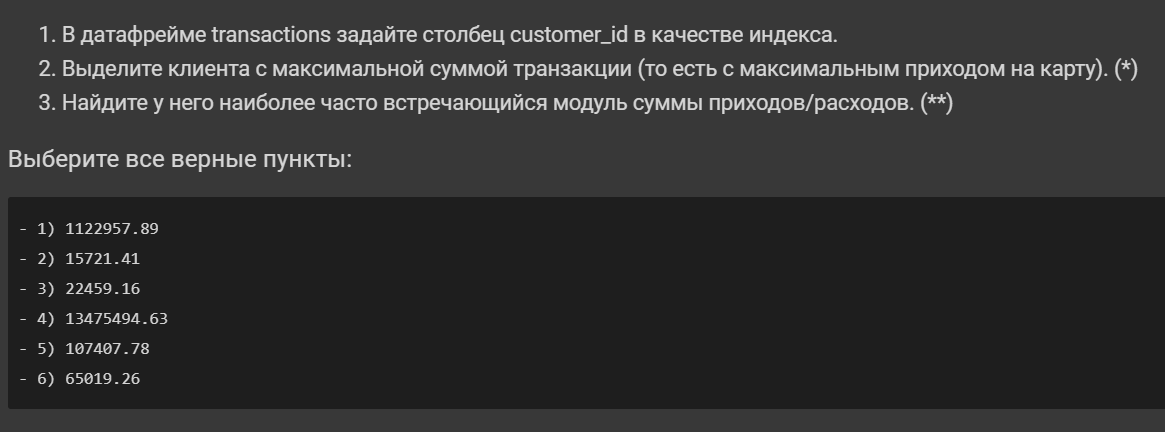
Устанавливаем в этой табице индексирующую колонку на tr\_type также

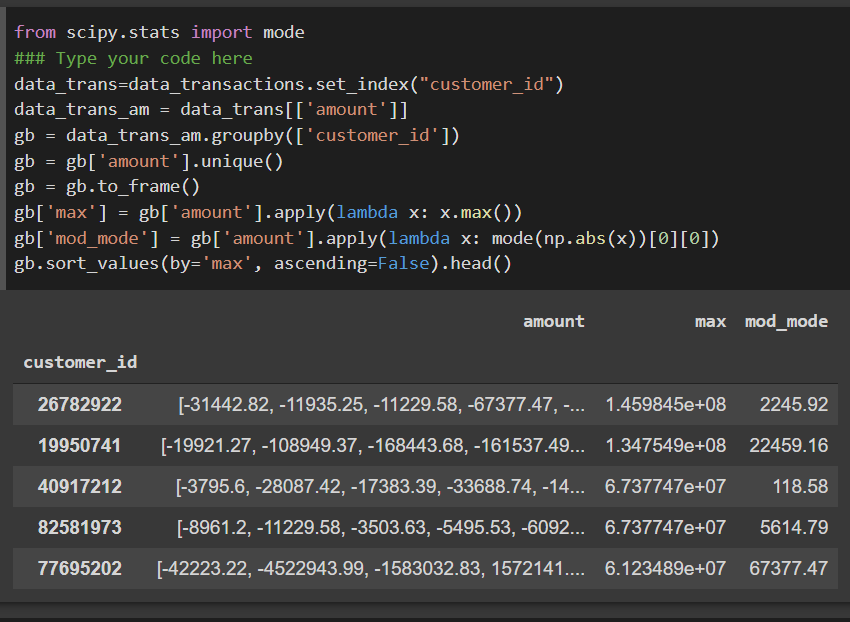


pd.merge(type\_counts, data\_types, left\_on='tr\_type', right\_on='tr\_type',  how='inner')

Мерджим таблицы по колонке tr\_type, чтобы можно было легко видеть описание транзакции и подобрать ответы.

**TASK 7**





data\_trans=data\_transactions.set\_index("customer\_id")

data\_trans\_am = data\_trans[['amount']]

gb = data\_trans\_am.groupby(['customer\_id'])

gb = gb['amount'].unique()

gb = gb.to\_frame()

Группируем таблицу по customer\_id, собирая для каждого покупателя в лист суммы всех его транзакций

gb['max'] = gb['amount'].apply(lambda x: x.max())

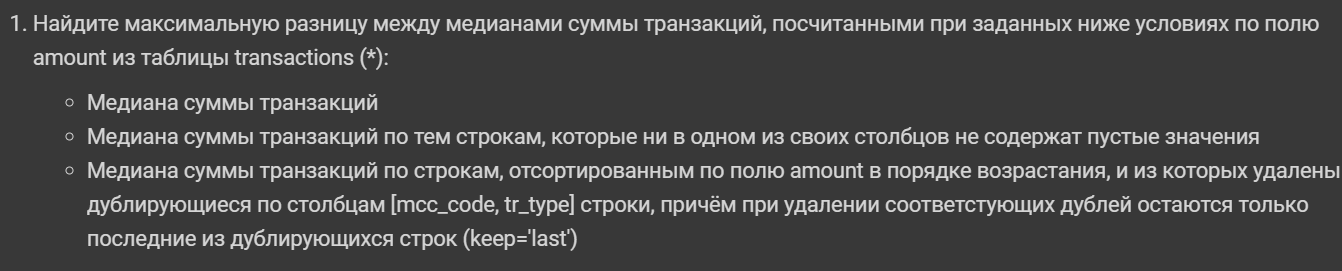
gb['mod\_mode'] = gb['amount'].apply(lambda x: mode(np.abs(x))[0][0])

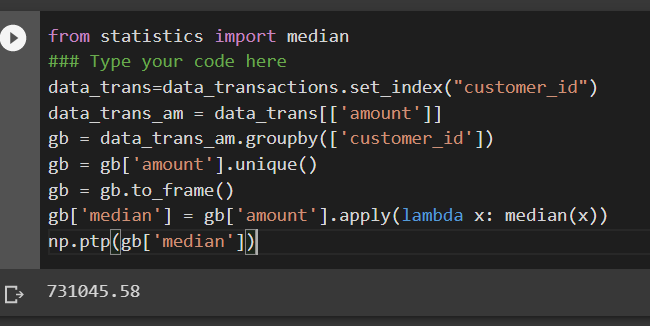
Добавляем колонки max и mod\_mode (мода модуля) и считаем данные значения для каждого пользователя, пользуясь колонкой ‘amount’, где каждый элемент лист.

gb.sort\_values(by='max', ascending=False).head()

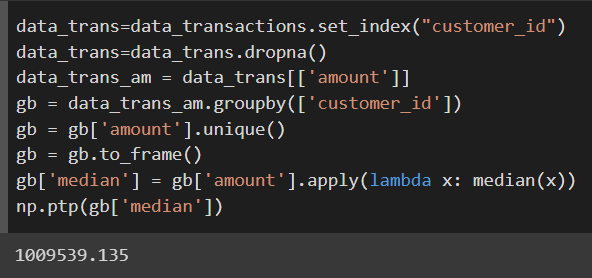
сортируем по колонке max от большего к меньшему, выводим первые пять ответов – получаем ответы задачи

**TASK 8**



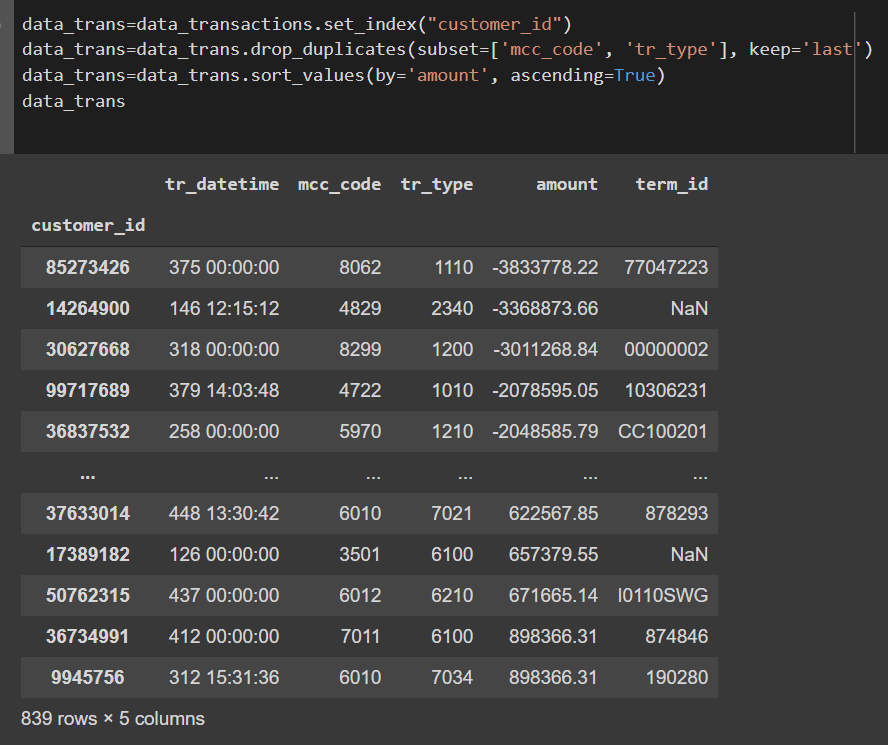


Делаем то же самое, что и в TASK 7, только в конце новый столбец заменяем медианами. И при помощи np.ptp находим максимальную разницу между элементами столбца



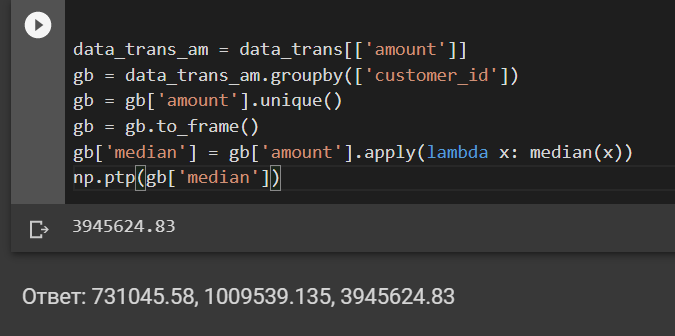
data\_trans=data\_trans.dropna()

Убираем те строки, где хотя бы один столбец nan. Далее делаем то же, что и выше



data\_trans=data\_trans.drop\_duplicates(subset=['mcc\_code', 'tr\_type'], keep='last')

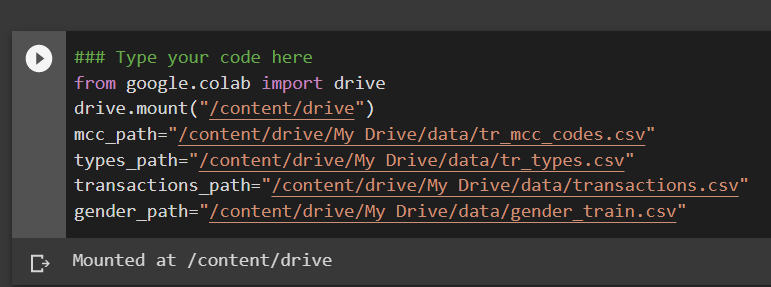
Убираем дубле по столбцам mcc\_code, tr\_type (subset=[‘mcc\_code’,’tr\_type’]), оставляем последнюю запись при этом keep=’last’



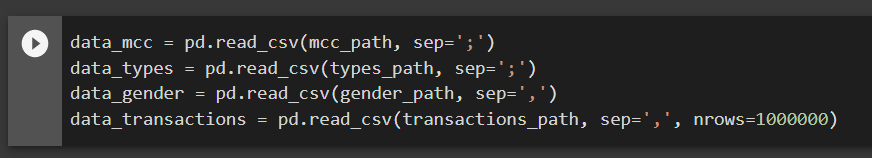
В модифицированной таблице снова повторяем те же действия, что и выше

**Homework\_2.ipynb**

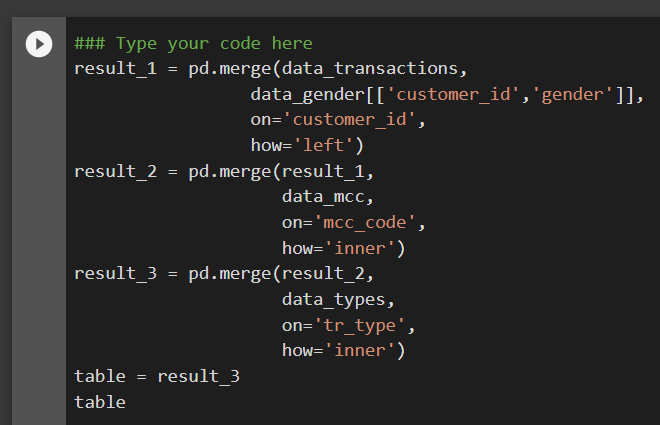
**===================== Загрузка датасетов==================**

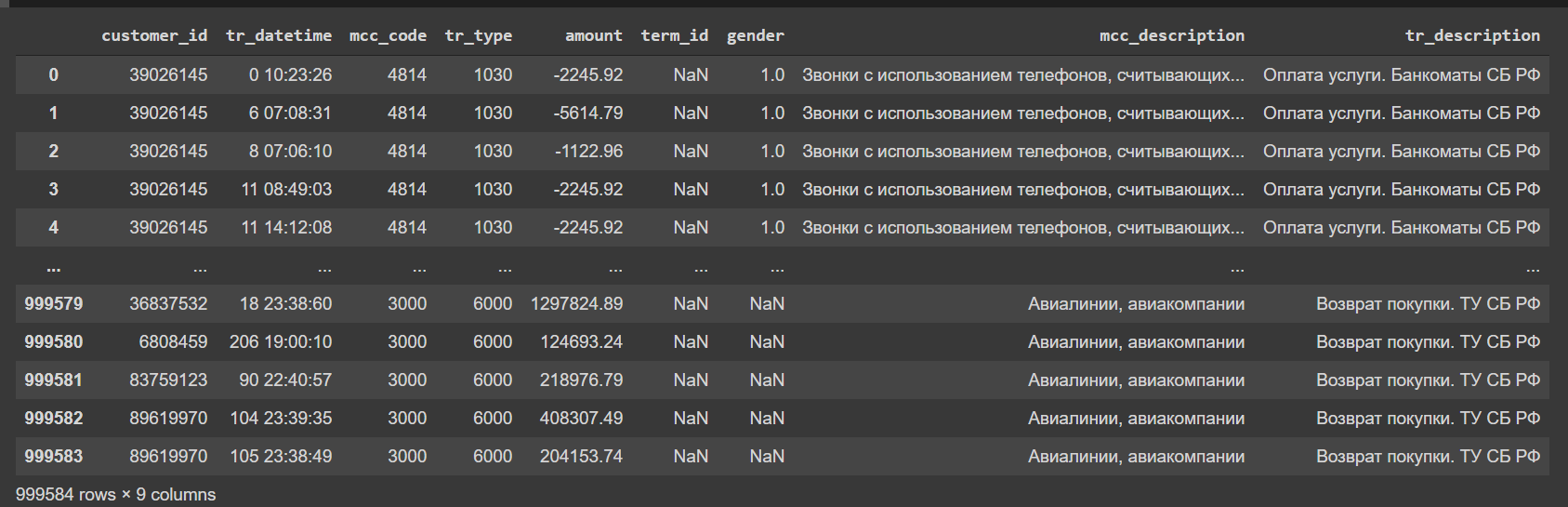


Подключаем файловую систему с гугл драйв (путь /content/drive/My Drive/), чтобы загружать оттуда датасеты.



Загружаем датасеты в оперативку. Подробнее код описан выше.

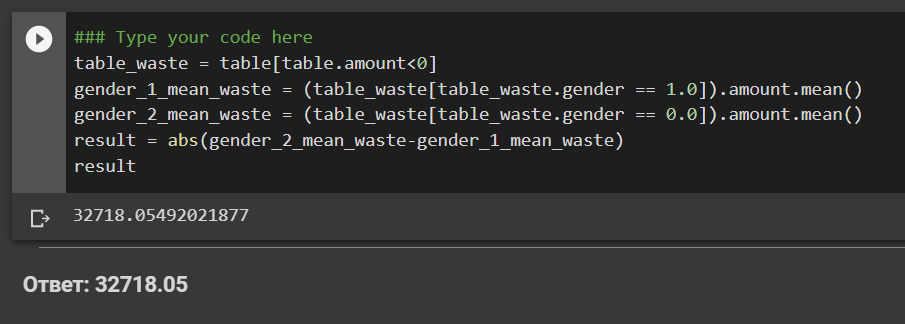




Три раза мерджим таблицы через left join, inner join и inner join.

**TASK 1**





table\_waste = table[table.amount<0]

Фильтруем таблицу при помощи булеан массива table.amount < 0. (требуются только расходы)

gender\_1\_mean\_waste = (table\_waste[table\_waste.gender == 1.0]).amount.mean()

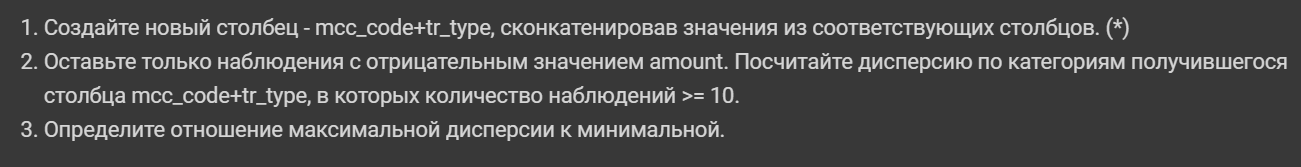
gender\_2\_mean\_waste = (table\_waste[table\_waste.gender == 0.0]).amount.mean()

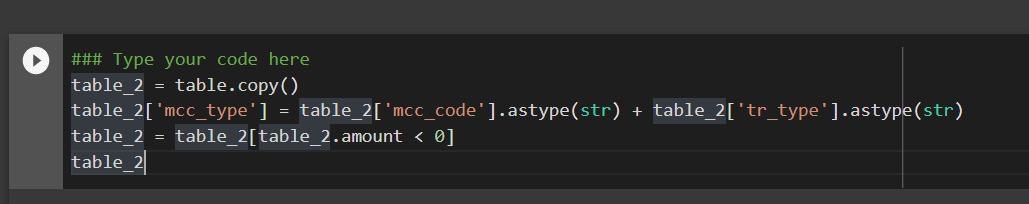
Находим среднее трат отдельно для мужчин и для женщин (применяем снова фильтр для выделения гендера)

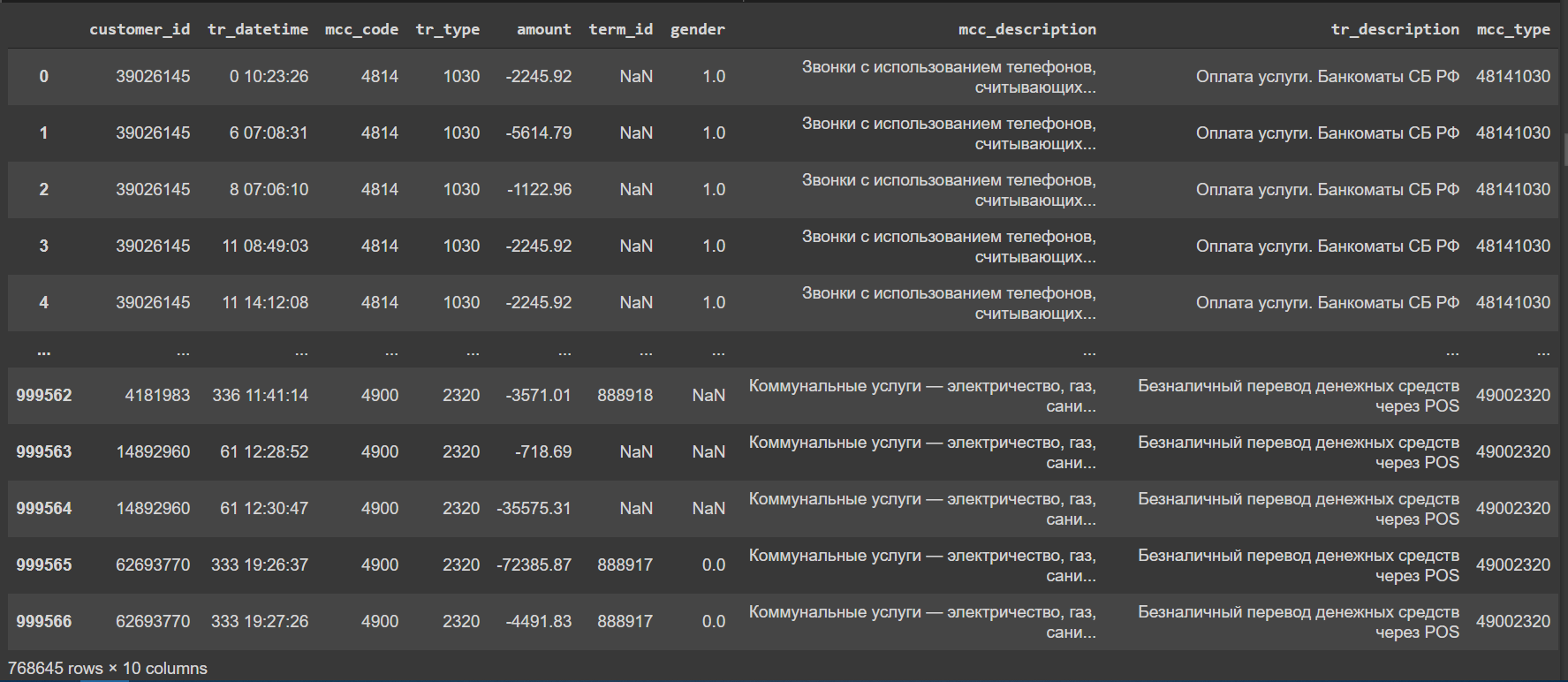
result = abs(gender\_2\_mean\_waste-gender\_1\_mean\_waste)

Модуль разности – ответ

**TASK 2**



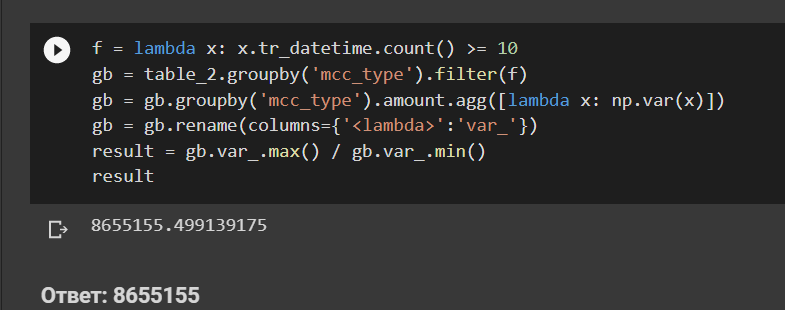




table\_2 = table.copy()

table\_2['mcc\_type'] = table\_2['mcc\_code'].astype(str) + table\_2['tr\_type'].astype(str)

Создаем новый столбец mcc\_type, куда записываем конкатенацию столбцов с mcc\_code и tr\_type. Для этого значения этих столбцов переводим в строки при помощи astype(str)



f = lambda x: x.tr\_datetime.count() >= 10

gb = table\_2.groupby('mcc\_type').filter(f)

Создаем фильтр, чтобы выделить группы по mcc\_type, где записей >= 10

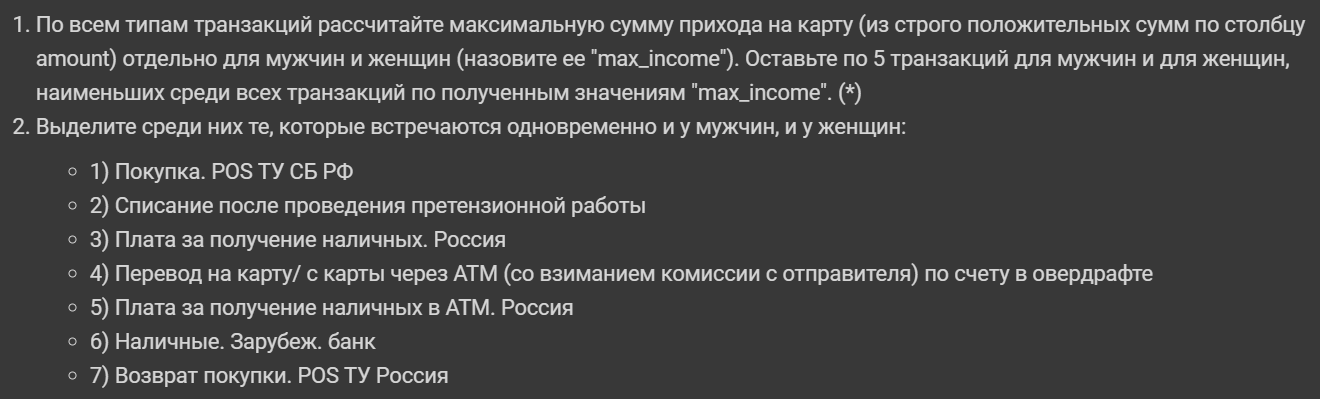
gb = gb.groupby('mcc\_type').amount.agg([lambda x: np.var(x)])

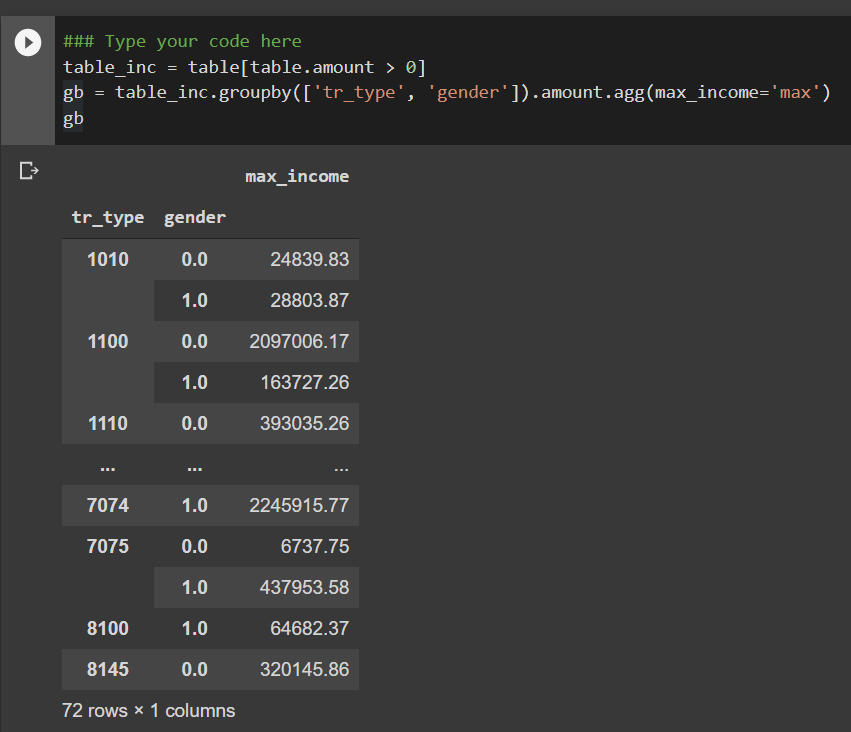
Для каждой группы по mcc\_type после фильтрации находим диcперсию при помощи np.var

result = gb.var\_.max() / gb.var\_.min()

Отношений максимальной среди дисперсий к минимальной – ответ

**TASK 3**



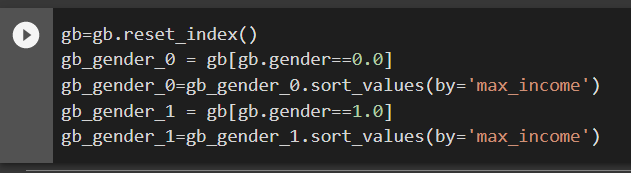


table\_inc = table[table.amount > 0]

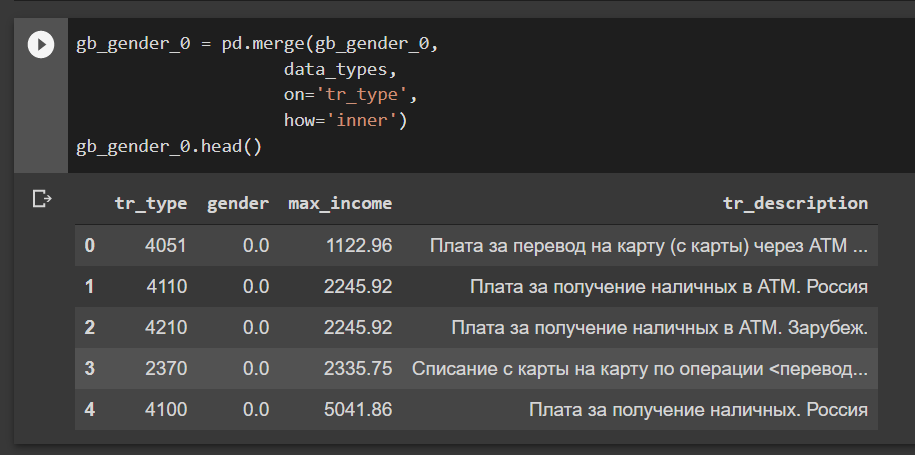
Фильтруем, выбираем только поступления на карту.

gb = table\_inc.groupby(['tr\_type', 'gender']).amount.agg(max\_income='max')

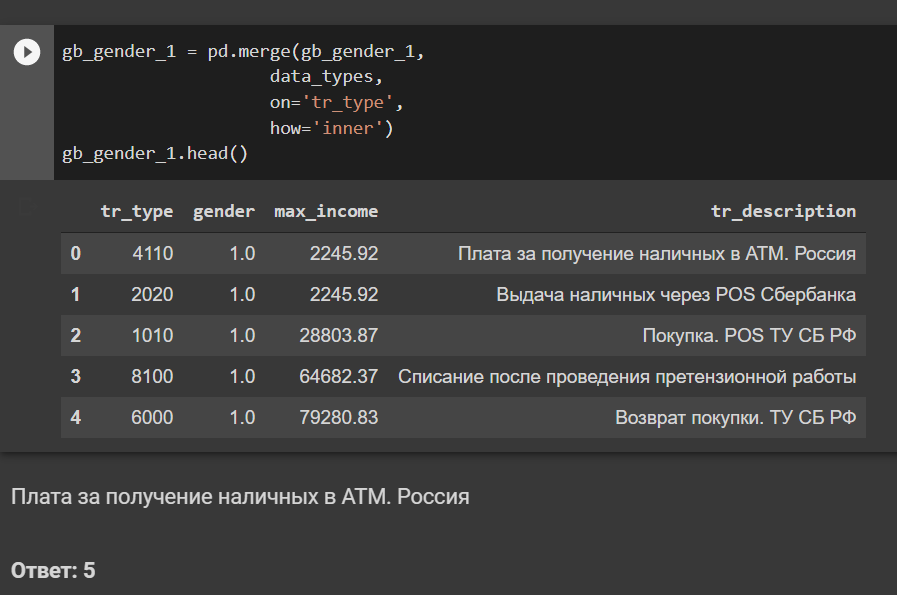
Группируем по парам ['tr\_type', 'gender']. Для каждой группы находим максимум по amount



Разбиваем таблицу отдельно на два гендера. И сортируем их по максимальному приходу на карту.

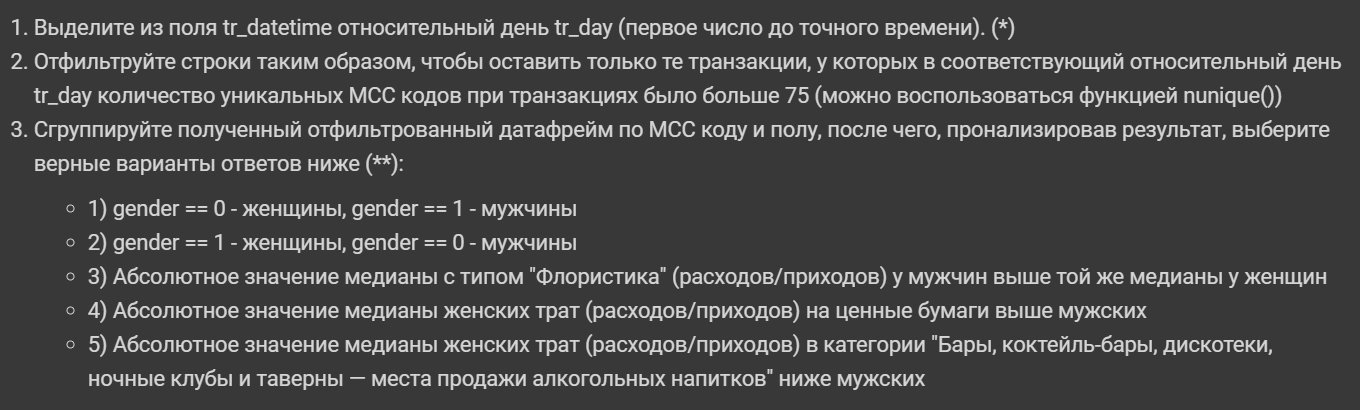


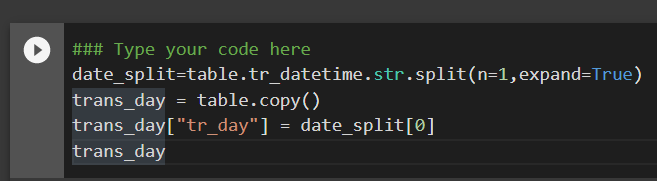
Мерджим с data\_types, чтобы в таблице были описания транзакций. Нужно для получения ответа на задание

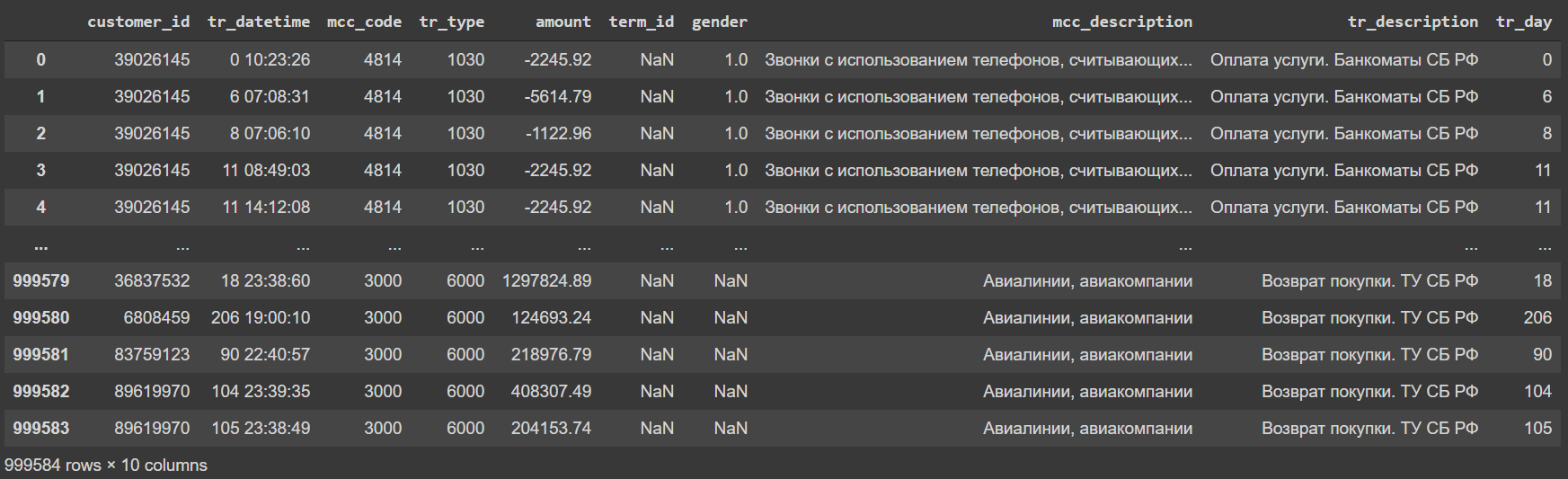


Делаем то же самое для другого гендера. Дальше получаем ответ.

**TASK 4**





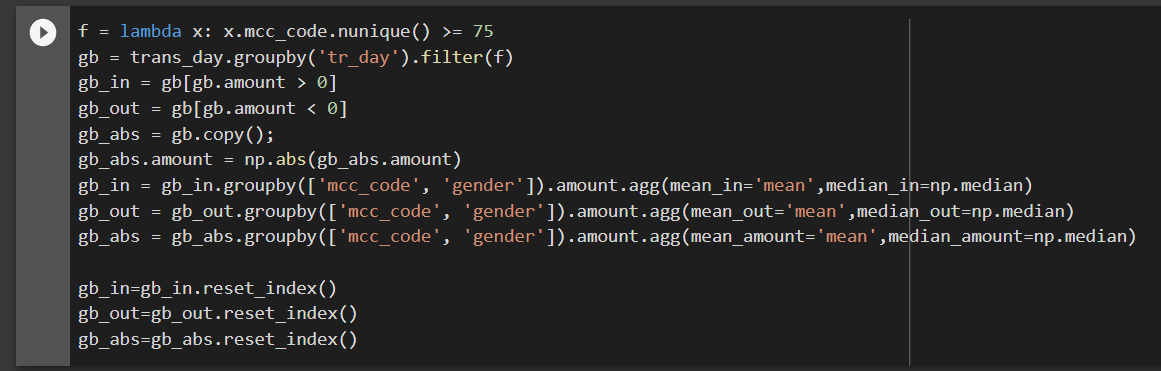


date\_split=table.tr\_datetime.str.split(n=1,expand=True)

trans\_day = table.copy()

trans\_day["tr\_day"] = date\_split[0]

Выделяем из даты день при помощи .str.split. В качестве сепаратора по умолчанию – пробел. date\_split[0] содержит series с днями – заносим его в новый столбец tr\_day



f = lambda x: x.mcc\_code.nunique() >= 75

gb = trans\_day.groupby('tr\_day').filter(f)

Фильтруем, чтобы остались дни с >= 75 уникальными записями по mcc\_code

gb\_in = gb[gb.amount > 0]

gb\_out = gb[gb.amount < 0]

gb\_abs = gb.copy();

gb\_abs.amount = np.abs(gb\_abs.amount)

Отдельно будем рассматривает приход на карты, траты и модуль приходов и трат

gb\_in = gb\_in.groupby(['mcc\_code', 'gender']).amount.agg(mean\_in='mean',median\_in=np.median)

gb\_out = gb\_out.groupby(['mcc\_code', 'gender']).amount.agg(mean\_out='mean',median\_out=np.median)

gb\_abs = gb\_abs.groupby(['mcc\_code', 'gender']).amount.agg(mean\_amount='mean',median\_amount=np.median)

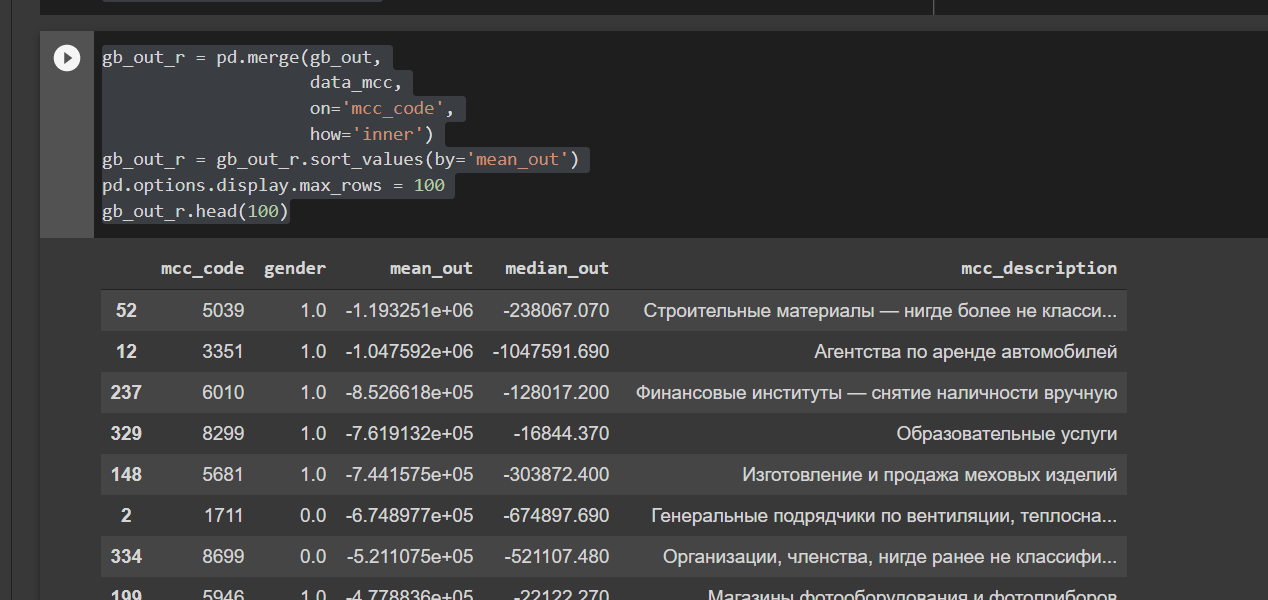
Группируем все три таблицы по парам (['mcc\_code', 'gender']), находя для каждой группы среднее и медиану по amount.

gb\_in=gb\_in.reset\_index()

gb\_out=gb\_out.reset\_index()

gb\_abs=gb\_abs.reset\_index()

Ресетим индексы, чтобы дальше было проще мерджить



gb\_out\_r = pd.merge(gb\_out,

                    data\_mcc,

                    on='mcc\_code',

                    how='inner')

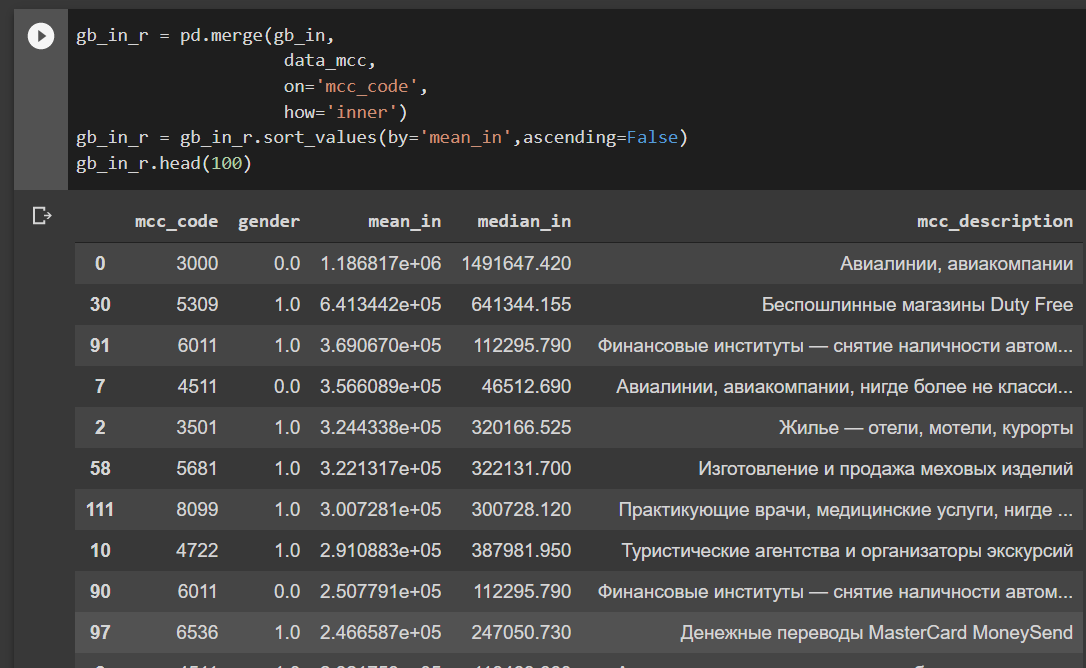
Мерджим таблицу с расходами с data\_mcc, чтобы узнать, на что была транзакиция.

gb\_out\_r = gb\_out\_r.sort\_values(by='mean\_out')

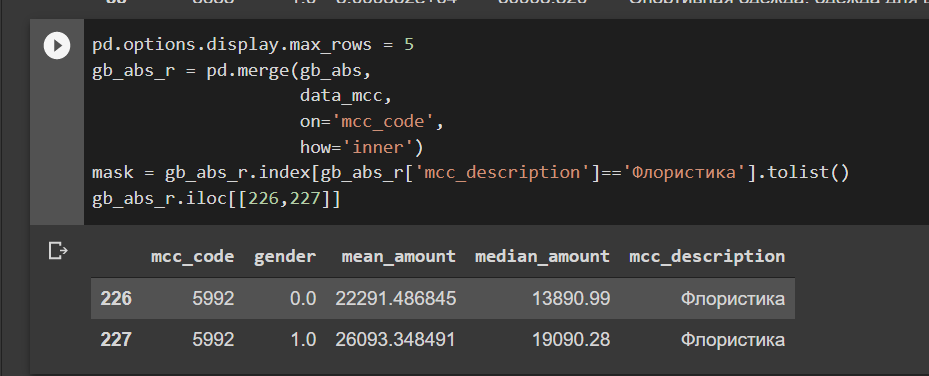
pd.options.display.max\_rows = 100

gb\_out\_r.head(100)

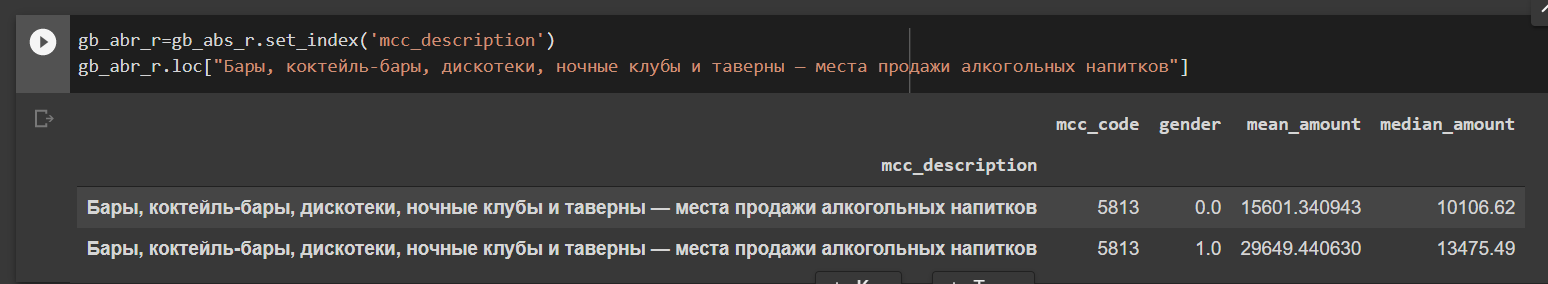
Сортируем по среднему, выводим первые 100 элементов



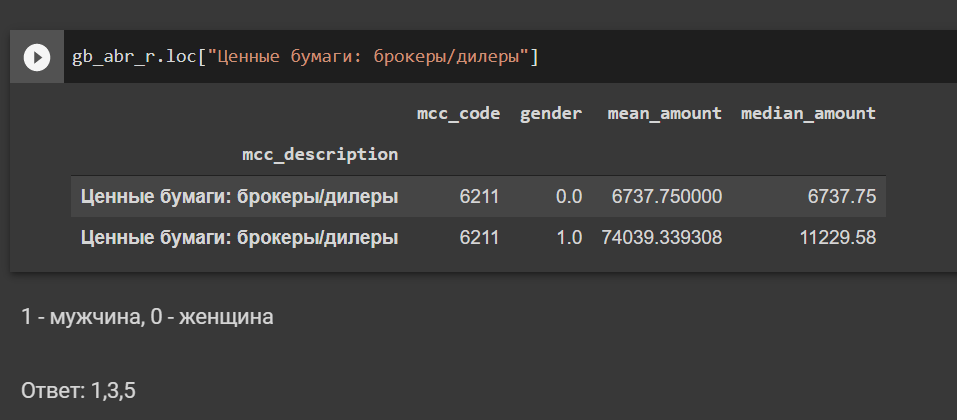
Делаем то же самое для прибыли



Снова мерджим, только абсолютное значение по тратам/расходам. И смотри средние и медиану для обоих гендеров для Флористики

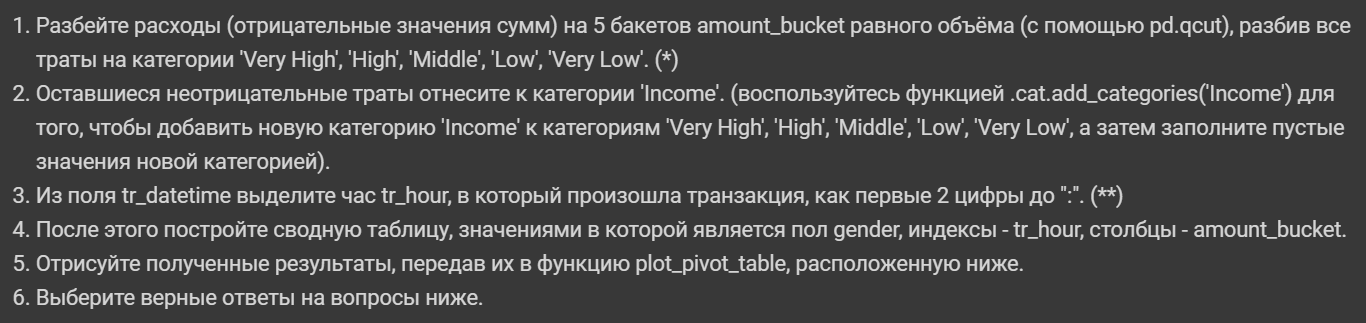


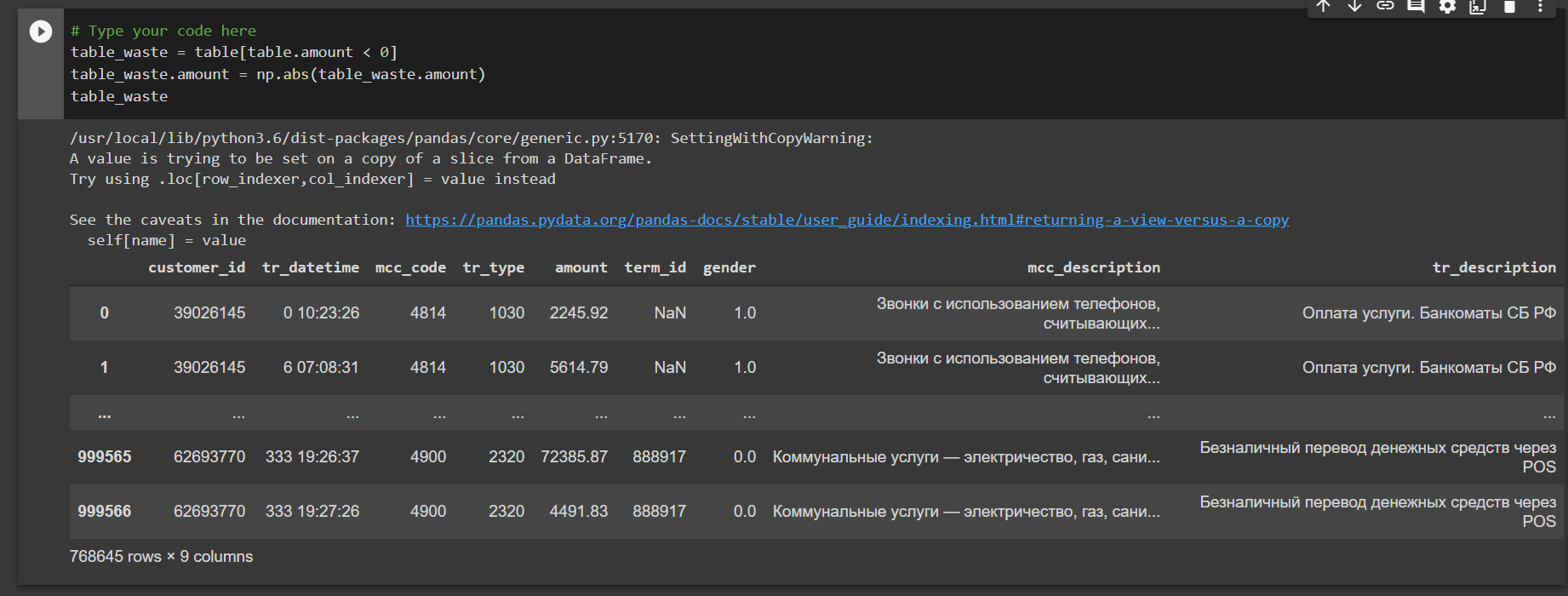
Смотри средние и медиану (модуль трат/расходов) для обоих гендеров по Барам…



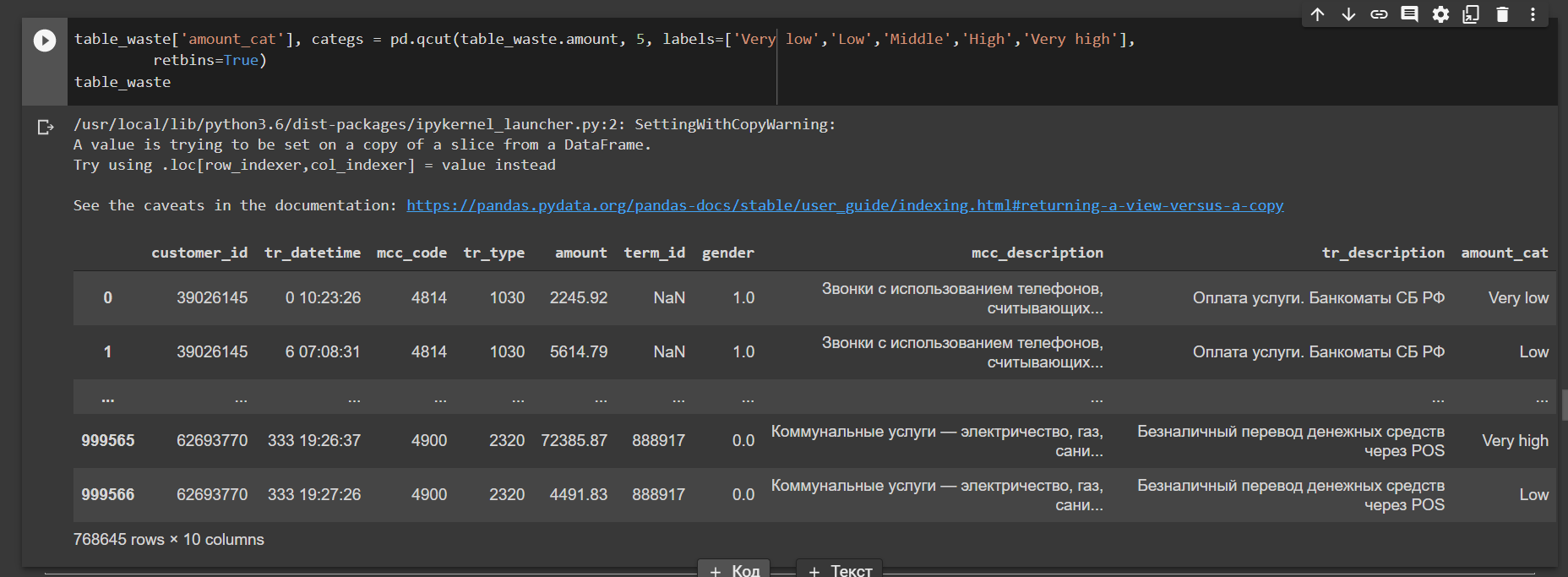
Смотри средние и медиану (модуль трат/расходов) для обоих гендеров Ценным бумагам

**TASK 5**





Вычленим только записи с тратами.

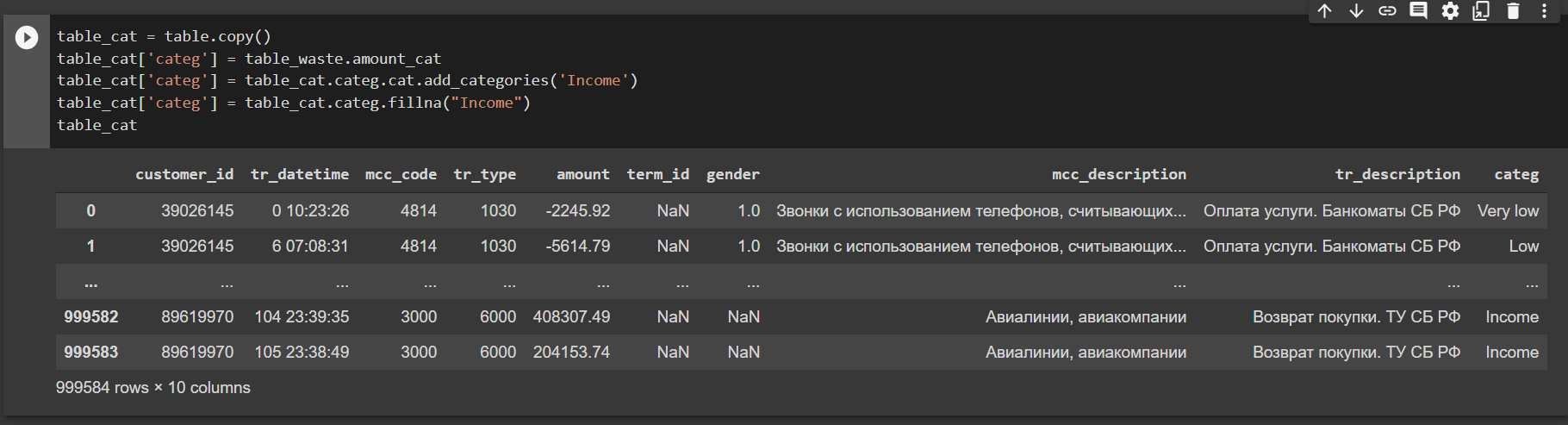


table\_waste['amount\_cat'], categs = pd.qcut(table\_waste.amount, 5, labels=['Very low','Low','Middle','High','Very high'],

         retbins=True)

table\_waste

Разбиваем расходы на 5 бакетов при помощи pd.qcut, название категории (бакета) записываем в новую колонку amount\_cat



table\_cat['categ'] = table\_waste.amount\_cat

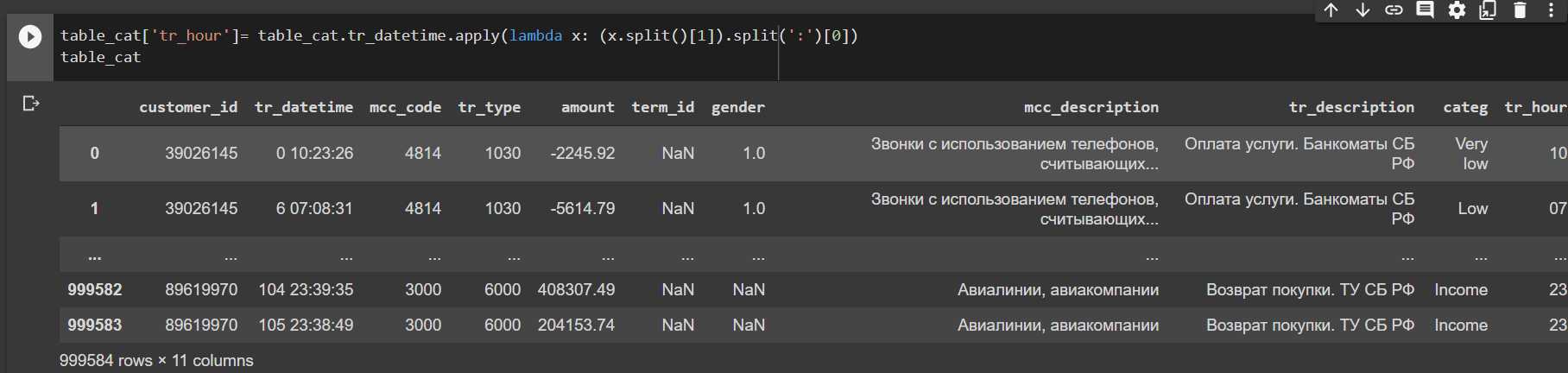
Добавляем в начальную табличку колонку с категориями и присваем ей колонку категорий для расходов

table\_cat['categ'] = table\_cat.categ.cat.add\_categories('Income')

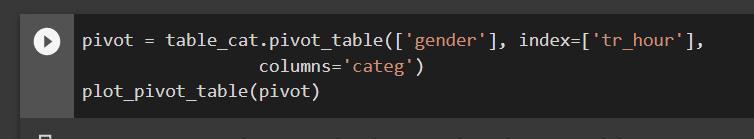
Добавляем для этой колонки категорию Income

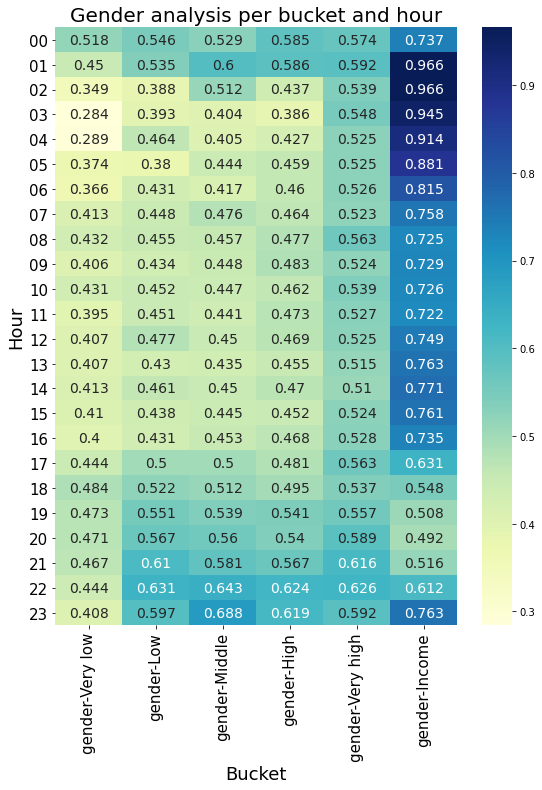
table\_cat['categ'] = table\_cat.categ.fillna("Income")

Все nan соответствовали прибыли, поэтому присваиваем им эту категорию



Вычленяем час из tr\_datetime, применяя к каждой записи этой колонки два раза сплит – первый раз по пробелу (получаем в [1] время) далее по двоеточию (получаем в [0] час). Заносим результат в новую колонку tr\_hour





pivot = table\_cat.pivot\_table(['gender'], index=['tr\_hour'],

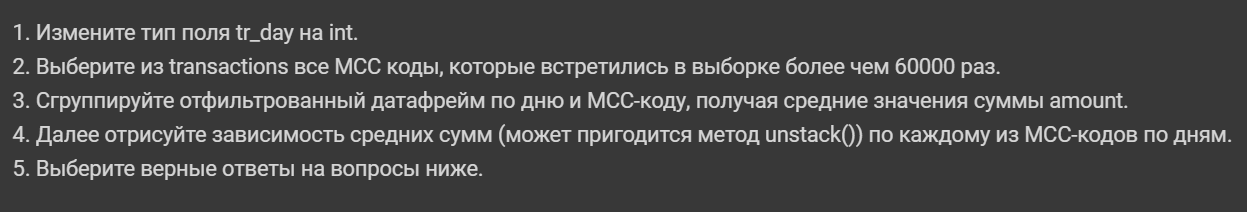
                 columns='categ')

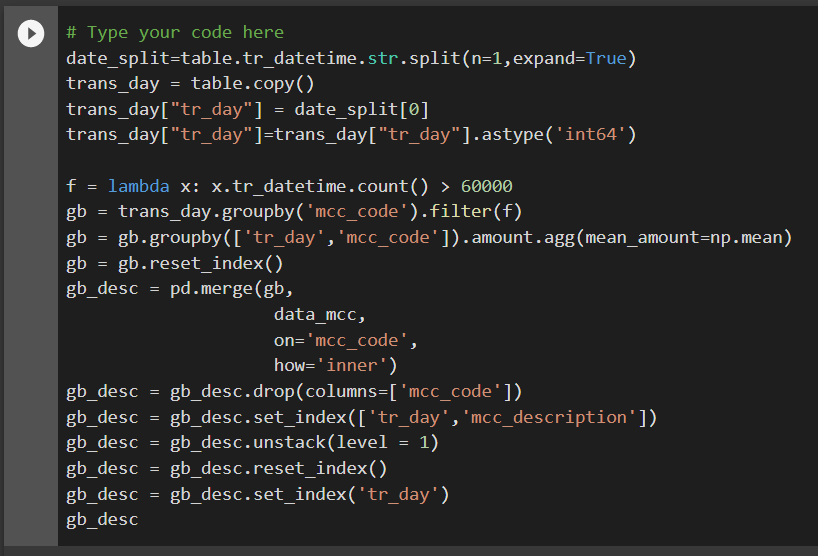
Строим сводную таблицу, значениями в которой является пол gender, индексы - tr\_hour, столбцы - amount\_bucket.

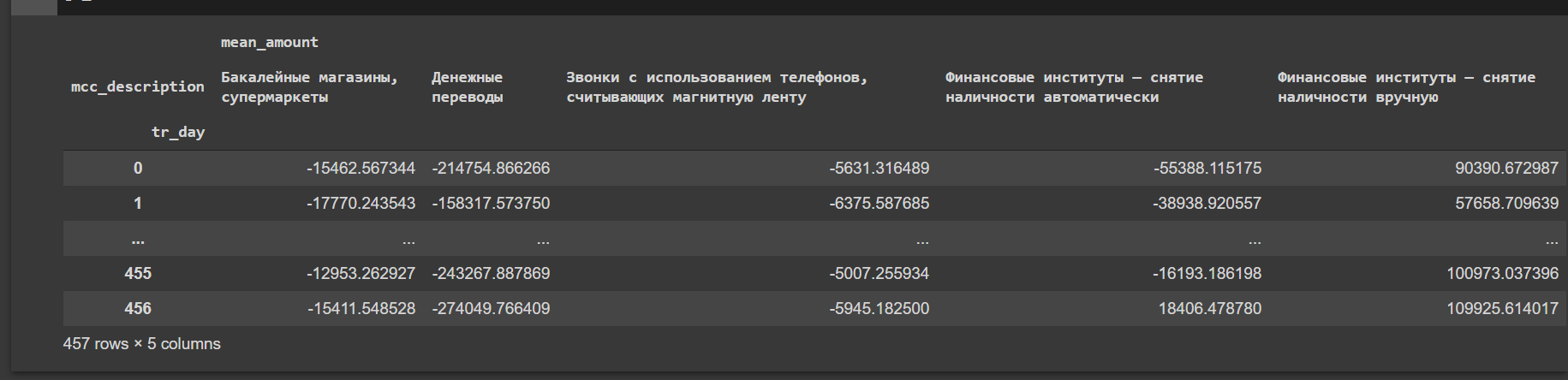
plot\_pivot\_table(pivot)

Отрисовываем полученные результаты, передав их в функцию plot\_pivot\_table, которая дана в задании

**TASK 6**

****





date\_split=table.tr\_datetime.str.split(n=1,expand=True)

trans\_day = table.copy()

trans\_day["tr\_day"] = date\_split[0]

trans\_day["tr\_day"]=trans\_day["tr\_day"].astype('int64')

Выделяем день из tr\_datetime. Переводим день из строки в int64.

f = lambda x: x.tr\_datetime.count() > 60000

gb = trans\_day.groupby('mcc\_code').filter(f)

gb = gb.groupby(['tr\_day','mcc\_code']).amount.agg(mean\_amount=np.mean)

Выделяем только записи, где mcc\_code встречался более 60000 раз

gb\_desc = pd.merge(gb,

                    data\_mcc,

                    on='mcc\_code',

                    how='inner')

gb\_desc = gb\_desc.drop(columns=['mcc\_code'])

Мерджим с таблицей mcc\_code, чтобы получим описания транзакций, убираем mcc\_code (чистим таблицу, для построения графика)

gb\_desc = gb\_desc.set\_index(['tr\_day','mcc\_description'])

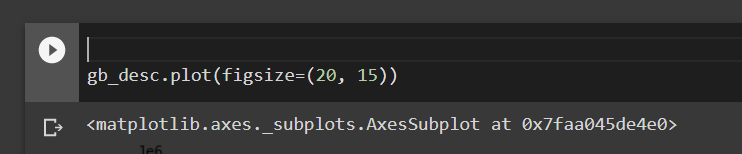
gb\_desc = gb\_desc.unstack(level = 1)

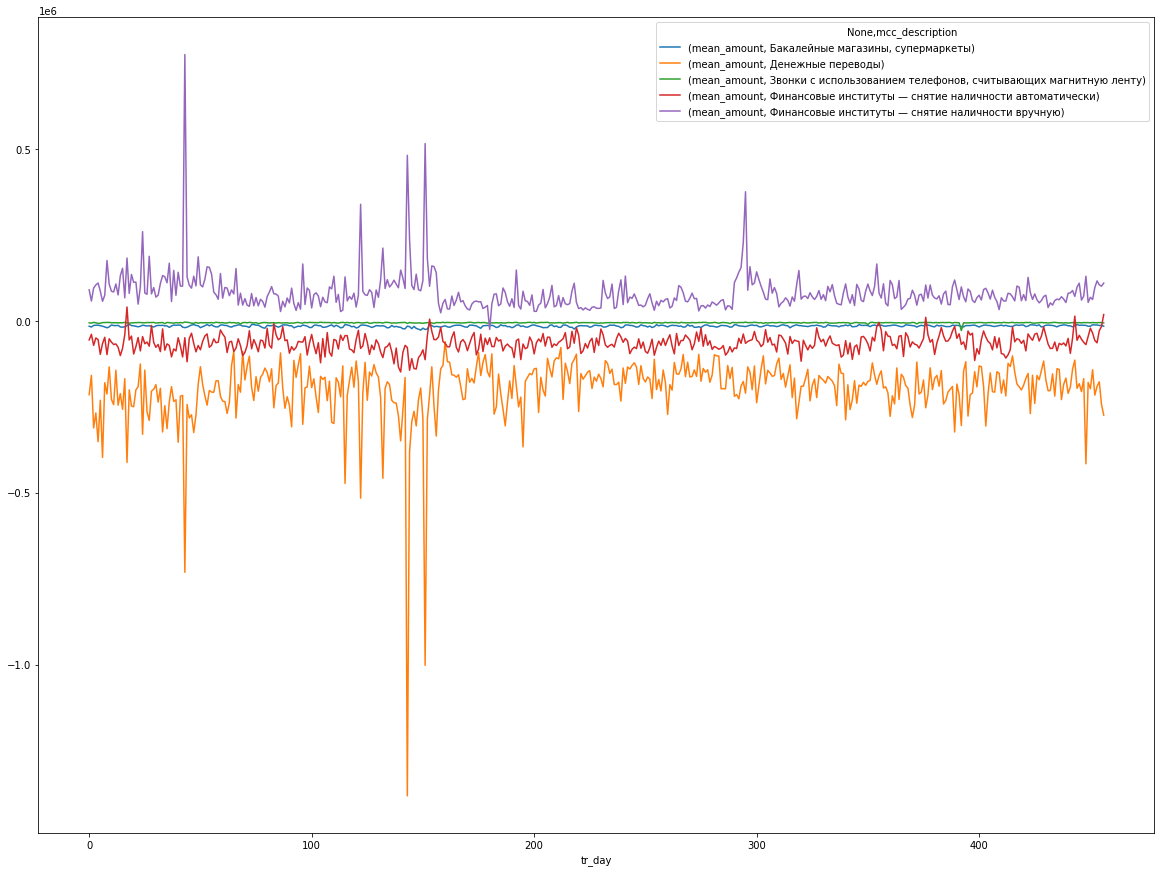
gb\_desc = gb\_desc.reset\_index()

gb\_desc = gb\_desc.set\_index('tr\_day')

Индексируем по парам tr\_day и mcc\_description. Развертываем таблицу, перенося в столбцы разновидности mcc\_description (в итоге их будет пять штук)

В качестве индекса используем tr\_day.





gb\_desc.plot(figsize=(20, 15))

Строим график. Tr\_day – ось X, amount – ось Y. Разные столбцы (транзакции с определенными описанием) стали разными графиками