Выполнили: Скворцов Иван, Никулина Евгения, Кордзахия Натела, Татаринов Артем (БЭК-181)

Используемые пакеты

```
begin
    using Pkg
    Pkg.add("DataFrames")
    Pkg.add("CSV")
    Pkg.add("Plots")
end
```

```
begin
    using Dates
    using CSV
    using DataFrames
    using Plots
end
```

Вводные слова

В данном обзоре функционала Julia мы будем использовать две таблицы, содержащие обезличенные данные одной известной сети магазинов одежды.

Таблица `transactions_footfall` содержит данные по количеству посетителей и транзакций в магазинах сети.

Таблица `events` содержит количество кликов (событий) в приложении, которое продавцы-консультанты используют для работы с клиентами.

Целью нашей работы станет исследование зависимости между **частотой использования приложения** и **конверсией** (отношением кол-ва транзакций к общему числу клиентов).

Подгрузка датасетов

Данные хранятся в формате `csv`. Для их корректной подгрузки мы задаем формат дат (переменная `dftm`), а также тип первого столбца – `String`, поскольку ID магазина является категориальной переменной.

```
dfmt = dateformat"yyyy-mm-dd"
```

transactions_footfall =

	store_id	date	transactions	footfall
1	"2174"	2020-11-01	14.0	366.0
2	"2174"	2020-11-02	14.0	missing
3	"2174"	2020-11-03	14.0	252.0
4	"2174"	2020-11-04	14.0	420.0
5	"2174"	2020-11-05	7.0	255.0
6	"2174"	2020-11-06	7.0	213.0
7	"2174"	2020-11-07	14.0	369.0
8	"2174"	2020-11-08	21.0	372.0
9	"2174"	2020-11-09	7.0	147.0
10	"2174"	2020-11-10	14.0	207.0

more

2860 "3889" 2020-11-30 7.0 147.0

```
transactions_footfall = CSV.read("transactions_footfall.csv", DataFrame, types=Dict(1=>String), dateformat=dfmt) # объект Dict в аргументе types задает желательные форматы столбцов
```

usage =

	store_id	date	events
1	"3048"	2020-11-01	214
2	"3048"	2020-11-02	116
3	"3048"	2020-11-03	97
4	"3048"	2020-11-04	207
5	"3048"	2020-11-05	285
6	"3048"	2020-11-06	220
7	"3048"	2020-11-07	334
8	"3048"	2020-11-08	156
9	"3048"	2020-11-09	171
10	"3048"	2020-11-10	97

more

3201 "3539" 2020-11-30 50

```
usage = CSV.read("usage.csv", DataFrame, types=Dict(1=>String), dateformat=dfmt)
```

Первичное исследование данных

При помощи функции `describe` выясняем, что имеющиеся данные захватывают период с 1 по 30 ноября 2020 года.

	variable	mean	min	median	max	nmissing	eltype
1	:store_id	nothing	"2011"	nothing	"4981"	0	String
2	:date	nothing	2020-11-01	nothing	2020-11-30	0	Date
3	:transactions	16.0706	0.0	14.0	91.0	0	Float64
4	:footfall	294.125	0.0	237.0	1635.0	32	Union{Missing, Fl

```
describe(transactions_footfall)
```

	variable	mean	min	median	max	nmissing	eltype
1	:store_id	nothing	"2027"	nothing	"4981"	0	String
2	:date	nothing	2020-11-01	nothing	2020-11-30	0	Date
3	:events	134.333	2	102.0	970	0	Int64

```
describe(usage)
```

При этом данные по пользованию приложением есть не для всех магазинов: скорее всего, в некоторых из них оно не использовалось вовсе. Это дает нам возможность посмотреть на работу различных типов объединения данных: *left join*, *right join* и *inner join*.

93

```
length(unique(transactions_footfall.store_id))
```

77

```
length(unique(usage.store_id))
```

При помощи операций над множествами мы можем выяснить, насколько полными будут данные после их объединения. Для этого зададим множества значений столбца `store_id` в для обоих датасетов:

```
Set{String} with 77 elements:
  "3852"
  "2394"
```

```
  "3196"
  "2362"
  "2452"
  "3757"
  "2678"
  "4715"
  "2987"
  "2732"
  "4095"
  "4549"
  "2423"
  ⋮
```

```
begin
    stores_in_tf = Set(transactions_footfall.store_id)
    stores_in_ug = Set(usage.store_id)
end
```

В частности, для 76 из 77 магазинов, использовавших приложение в ноябре, имеются данные по трафику и транзакциям:

```
Set{String} with 76 elements:
  "4601"
  "2614"
  "3922"
  "4619"
  "4606"
  "2732"
  "2666"
  "4057"
  "2922"
  "4071"
  "2064"
  "2736"
  "3745"
  ⋮
```

```
intersect(stores_in_tf, stores_in_ug)
```

17 магазинов не использовали приложение, но обслуживали посетителей:

```
Set{String} with 17 elements:
  "2011"
  "2083"
  "3630"
  "4254"
  "4359"
  "3591"
  "4560"
  "2979"
  "2596"
  "4222"
  "4003"
  "3311"
  "3160"
  ⋮
```

```
setdiff(stores_in_tf, stores_in_ug)
```

Опять же, у 1 магазина из использовавших приложение нет данных по числу посетителей (возможно, он не работал, а приложение использовалось для внутренних процессов).

```
Set{String} with 1 element:
  "4659"
```

```
setdiff(stores_in_ug, stores_in_tf)
```

Можем вывести данные по пользованию для этого магазина. Видим, что приложение использовалось всего четыре дня.

	store_id	date	events
1	"4659"	2020-11-01	19
2	"4659"	2020-11-02	4
3	"4659"	2020-11-03	5
4	"4659"	2020-11-12	2

```
usage[usage[!, "store_id"] .== "4659", :]
```

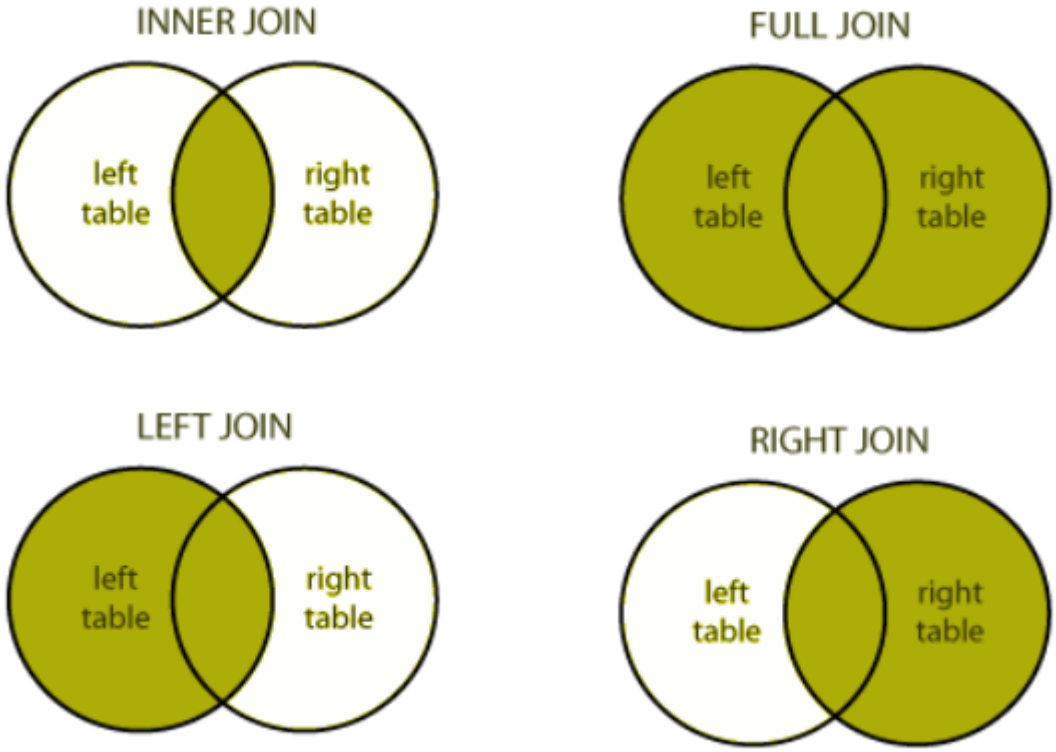
Наконец, в целом две имеющиеся таблицы покрывают 94 магазина сети:

```
Set{String} with 94 elements:
  "4601"
  "2614"
  "3922"
  "4619"
  "4606"
  "3630"
  "2732"
  "2666"
  "4222"
  "4057"
  "2922"
  "4071"
  "2064"
  ⋮
```

```
union(stores_in_ug, stores_in_tf)
```

Объединение таблиц (join)

Объединение таблиц обычно понимается в смысле join'ов в том виде, в каком они представлены в языке SQL. Ниже представлено схематическое представление четырех основных типов объединений. В этом разделе мы разберем каждый из них на базе функционала Julia DataFrames.



Синтаксис

Объединение таблиц в Julia производится при помощи набора функций, полный перечень которых представлен в **документации**. Разберем общую логику их синтаксиса.

Функции вида `*join(df1, df2, on = ...)` принимают три обязательных аргумента: *левая (первая) таблица*, *правая (вторая) таблица* и *столбцы-индексы (key columns)*.

Основная сложность может возникнуть при задании столбцов-индексов. Разберем основные случаи, которые встречаются при работе с реальными данными:

- Названия столбцов-индексов **совпадают** в обеих таблицах.
  - Один индекс: `on = :idx_col_name`
  - Несколько индексов: `on = [:idx_col_name1, :idx_col_name2, ...]`
- Названия столбцов-индексов **не совпадают** в обеих таблицах. В таком случае нужно задать соответствие между названиями столбцов в датасетах:
  - Первый способ, кортежи: `on = [(:idx_1_df1, :idx_1_df2), (:idx_2_df1, :idx_2_df2), ...]`
  - Второй способ, словари: `on = [:idx_1_df1 => :idx_1_df2, :idx_2_df1 => :idx_2_df2, ...]`

Иногда бывает необходимо проверить, являются ли сочетания индексов уникальными в каждом из датасетов (например, убедиться, что в таблице встречается лишь одно значение переменной

для пары store_id-date). В таком случае используется дополнительный аргумент функции `*join` – `validate`, принимающий кортеж из двух логических значений true/false. Первое значение отвечает за проверку уникальности в левом датасете, второе значение – за проверку уникальности в правом датасете. Если будут обнаружены повторяющиеся индексы, функция выдаст ошибку типа

```
ERROR: ArgumentError: Merge key(s) are not unique in both df1 and df2. First duplicate in df1 at 3. First duplicate in df2 at 3
```

Inner join

Этот тип объединения подразумевает, что в итоговый датасет включаются только те наблюдения, индексы которых содержатся в обеих таблицах. В нашем случае мы имеем дело с двойным индексом: нам необходимо объединять данные по столбцам *store_id* и *date* одновременно.

```
inner =
```

	store_id	date	events	transactions	footfall
1	"3048"	2020-11-01	214	7.0	99.0
2	"3048"	2020-11-02	116	14.0	126.0
3	"3048"	2020-11-03	97	14.0	156.0
4	"3048"	2020-11-04	207	14.0	144.0
5	"3048"	2020-11-05	285	7.0	123.0
6	"3048"	2020-11-06	220	14.0	147.0
7	"3048"	2020-11-07	334	14.0	153.0
8	"3048"	2020-11-08	156	7.0	90.0
9	"3048"	2020-11-09	171	21.0	132.0
10	"3048"	2020-11-10	97	21.0	147.0
	more				
3396	"3539"	2020-11-30	60	7.0	84.0

```
inner = innerjoin(usage, transactions_footfall, on = [:store_id, :date])
```

Исходя из нашего знания данных, полученного в предыдущем разделе, мы ожидаем, что inner join позволит получить объединенный датасет для **76 магазинов**, у которых имеются данные и по пользованию, и по посетителям. Проверим, что это действительно так:

```
Set{String} with 76 elements:
  "3852"
  "2394"
  "3196"
  "2362"
```

```
  "2452"
  "3757"
  "2678"
  "4715"
  "2987"
  "2732"
  "4095"
  "4549"
  "2423"
  ⋮
```

```
Set(inner.store_id)
```

Full join

Этот тип объединения сохранит все индексы, которые имеются в двух таблицах. При этом недостающие данные будут заполнены пустыми значениями. В Julia соответствующая функция называется `outerjoin` (не путать с настоящим outer join, сохраняющим только наблюдения, индексы которых встречаются лишь в одном из датасетов!)

`full =`

	store_id	date	events	transactions	footfall
1	"3048"	2020-11-01	214	7.0	99.0
2	"3048"	2020-11-02	116	14.0	126.0
3	"3048"	2020-11-03	97	14.0	156.0
4	"3048"	2020-11-04	207	14.0	144.0
5	"3048"	2020-11-05	285	7.0	123.0
6	"3048"	2020-11-06	220	14.0	147.0
7	"3048"	2020-11-07	334	14.0	153.0
8	"3048"	2020-11-08	156	7.0	90.0
9	"3048"	2020-11-09	171	21.0	132.0
10	"3048"	2020-11-10	97	21.0	147.0

more

3809 "3889" 2020-11-24 missing 7.0 201.0

```
full = outerjoin(usage, transactions_footfall, on = [:store_id, :date])
```

Как и следовало ожидать, полученный датасет включает данные по 94 магазинам:

```
Set{String} with 94 elements:
  "3852"
  "2394"
  "3196"
  "2362"
  "2452"
  "3757"
```

```
  "2678"
  "4715"
  "2011"
  "2987"
  "2732"
  "2083"
  "4095"
  ⋮
```

```
Set(full.store_id)
```

Left join

Этот тип объединения сохранит все индексы, которые содержатся в левой (первой) таблице. Недостающие данные из правой таблицы будут заполнены пустыми значениями.

left =

	store_id	date	events	transactions	footfall
1	"3048"	2020-11-01	214	7.0	99.0
2	"3048"	2020-11-02	116	14.0	126.0
3	"3048"	2020-11-03	97	14.0	156.0
4	"3048"	2020-11-04	207	14.0	144.0
5	"3048"	2020-11-05	285	7.0	123.0
6	"3048"	2020-11-06	220	14.0	147.0
7	"3048"	2020-11-07	334	14.0	153.0
8	"3048"	2020-11-08	156	7.0	90.0
9	"3048"	2020-11-09	171	21.0	132.0
10	"3048"	2020-11-10	97	21.0	147.0
more					
3303	"3539"	2020-11-30	50	7.0	84.0

```
left = leftjoin(usage, transactions_footfall, on = [:store_id, :date])
```

Разумеется, полученная таблица содержит данные по 77 магазинам:

```
Set{String} with 77 elements:
  "3852"
  "2394"
  "3196"
  "2362"
  "2452"
  "3757"
  "2678"
  "4715"
  "2987"
  "2732"
  "4095"
```

```
  "4549"
  "2423"
  ⋮
```

```
Set(left.store_id)
```

Right join

Этот тип объединения сохранит все индексы, которые содержатся в правой (второй) таблице. Недостающие данные из левой таблицы будут заполнены пустыми значениями.

```
right =
```

	store_id	date	events	transactions	footfall
1	"2174"	2020-11-01	44	14.0	366.0
2	"2174"	2020-11-02	19	14.0	missing
3	"2174"	2020-11-03	27	14.0	252.0
4	"2174"	2020-11-04	78	14.0	420.0
5	"2174"	2020-11-05	120	7.0	255.0
6	"2174"	2020-11-06	41	7.0	213.0
7	"2174"	2020-11-07	21	14.0	369.0
8	"2174"	2020-11-08	112	21.0	372.0
9	"2174"	2020-11-09	13	7.0	147.0
10	"2174"	2020-11-10	9	14.0	207.0
more					
3893	"3889"	2020-11-24	missing	7.0	201.0

```
right = rightjoin(usage, transactions_footfall, on = [:store_id, :date])
```

Полученная таблица содержит данные по 93 магазинам:

```
Set{String} with 93 elements:
  "3852"
  "2394"
  "3196"
  "2362"
  "2452"
  "3757"
  "2678"
  "4715"
  "2011"
  "2987"
  "2732"
  "2083"
  "4095"
  ⋮
```

```
Set(right.store_id)
```

Поскольку left и right объединения относительны, right join также можно сделать, поменяв порядок таблиц внутри функции `leftjoin`.

right_alternative =

	store_id	date	transactions	footfall	events
1	"2174"	2020-11-01	14.0	366.0	44
2	"2174"	2020-11-02	14.0	missing	19
3	"2174"	2020-11-03	14.0	252.0	27
4	"2174"	2020-11-04	14.0	420.0	78
5	"2174"	2020-11-05	7.0	255.0	120
6	"2174"	2020-11-06	7.0	213.0	41
7	"2174"	2020-11-07	14.0	369.0	21
8	"2174"	2020-11-08	21.0	372.0	112
9	"2174"	2020-11-09	7.0	147.0	13
10	"2174"	2020-11-10	14.0	207.0	9
more					
3893	"3889"	2020-11-30	7.0	147.0	17

```
right_alternative = leftjoin(transactions_footfall, usage, on = [:store_id, :date])
```

```
Set{String} with 93 elements:
  "3852"
  "2394"
  "3196"
  "2362"
  "2452"
  "3757"
  "2678"
  "4715"
  "2011"
  "2987"
  "2732"
  "2083"
  "4095"
  ⋮
```

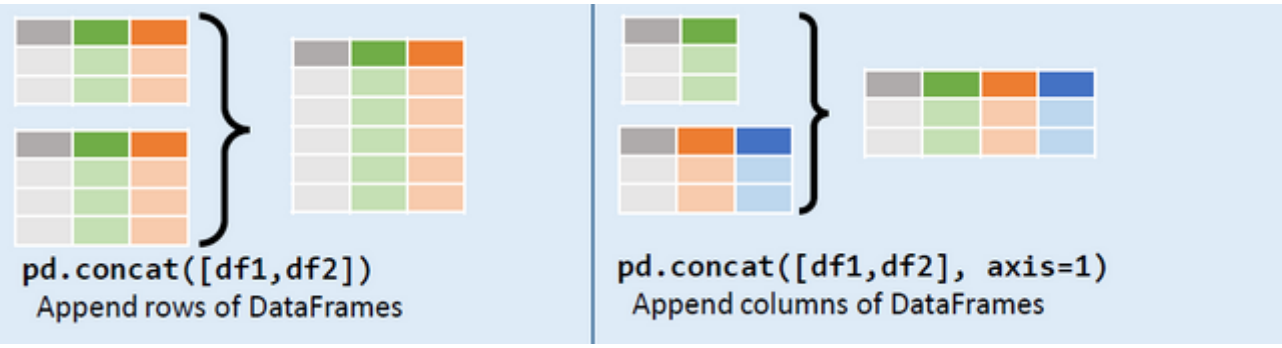
```
Set(right_alternative.store_id)
```

Более подробную информацию об объединениях таблиц в Julia DataFrames можно найти **здесь**.

Конкатенация таблиц

Конкатенация таблицы - это простое склеивание вдоль одной из осей. Ниже мы рассмотрим конкатенацию по горизонтали - `hcat()` и по вертикали - `vcat()`. Обе функции происходят от базовой функции `cat(..., dims = (m,n))`, которую удобно использовать, если надо объединить

таблицы с тремя и более измерениями. Для тех, кто знаком с питоном - эта фунция полностью аналогична `pd.concat()`



По вертикали

Конкатенация по вертикали позволяет объединить два датасета, если у них совпадает количество столбцов (в нашем примере три), при этом количество строк для объединения может быть любым:

`a =`

	store_id	date	events
1	"3048"	2020-11-01	214
2	"3048"	2020-11-02	116
3	"3048"	2020-11-03	97
4	"3048"	2020-11-04	207

```
a = usage[1:4, :]
```

`b =`

	store_id	date	events
1	"3048"	2020-11-05	285
2	"3048"	2020-11-06	220
3	"3048"	2020-11-07	334
4	"3048"	2020-11-08	156

```
b = usage[5:8, :]
```

`v_conc =`

store_id	date	events

	store_id	date	events
1	"3048"	2020-11-01	214
2	"3048"	2020-11-02	116
3	"3048"	2020-11-03	97
4	"3048"	2020-11-04	207
5	"3048"	2020-11-05	285
6	"3048"	2020-11-06	220
7	"3048"	2020-11-07	334
8	"3048"	2020-11-08	156

```
v_conc = vcat(a,b)
```

По горизонтали

Конкатенация по горизонтали позволяет объединить два датасета, если у них совпадает количество строк. При этом количество столбцов для объединения может быть любым:

```
c =
```

	store_id
1	"3048"
2	"3048"
3	"3048"
4	"3048"
5	"3048"
6	"3048"
7	"3048"
8	"3048"
9	"3048"
10	"3048"
	more
3301	"3639"

```
c = usage[:, 1:1]
```

d =

	date	events
1	2020-11-01	214
2	2020-11-02	116
3	2020-11-03	97
4	2020-11-04	207
5	2020-11-05	285
6	2020-11-06	220
7	2020-11-07	334
8	2020-11-08	156
9	2020-11-09	171
10	2020-11-10	97

more

```
d = usage[:, 2:3]
```

h_conc =

	store_id	date	events
1	"3048"	2020-11-01	214
2	"3048"	2020-11-02	116
3	"3048"	2020-11-03	97
4	"3048"	2020-11-04	207
5	"3048"	2020-11-05	285
6	"3048"	2020-11-06	220
7	"3048"	2020-11-07	334
8	"3048"	2020-11-08	156
9	"3048"	2020-11-09	171
10	"3048"	2020-11-10	97

more

3301 "3539" 2020-11-30 50

```
h_conc = hcat(c,d)
```

В **документации** можно подробнее прочитать о конкатенации в Julia.

Корреляционный анализ

Применяя полученные знания об объединении таблиц, выясним характер взаимосвязи между **частотой использования приложения** и **конверсией**.

Поскольку для этого нам нужны наиболее полные данные, будем использовать inner join для создания рабочего датасета:

data =

	store_id	date	events	transactions	footfall
1	"3048"	2020-11-01	214	7.0	99.0
2	"3048"	2020-11-02	116	14.0	126.0
3	"3048"	2020-11-03	97	14.0	156.0
4	"3048"	2020-11-04	207	14.0	144.0
5	"3048"	2020-11-05	285	7.0	123.0
6	"3048"	2020-11-06	220	14.0	147.0
7	"3048"	2020-11-07	334	14.0	153.0
8	"3048"	2020-11-08	156	7.0	90.0
9	"3048"	2020-11-09	171	21.0	132.0
10	"3048"	2020-11-10	97	21.0	147.0
	more				
3206	"3539"	2020-11-30	50	7.0	84.0

```
data = innerjoin(usage, transactions_footfall, on = [:store_id, :date])
```

Исключим наблюдения с пропущенными значениями (они встречаются только в столбце `footfall`):

	store_id	date	events	transactions	footfall
1	"3048"	2020-11-01	214	7.0	99.0
2	"3048"	2020-11-02	116	14.0	126.0
3	"3048"	2020-11-03	97	14.0	156.0
4	"3048"	2020-11-04	207	14.0	144.0
5	"3048"	2020-11-05	285	7.0	123.0
6	"3048"	2020-11-06	220	14.0	147.0
7	"3048"	2020-11-07	334	14.0	153.0
8	"3048"	2020-11-08	156	7.0	90.0
9	"3048"	2020-11-09	171	21.0	132.0

	store_id	date	events	transactions	footfall

```
dropmissing!(data)
```

Исключим наблюдения с нулевым количеством посетителей или транзакций: скорее всего, в такие дни магазин не работал.

	store_id	date	events	transactions	footfall
1	"3048"	2020-11-01	214	7.0	99.0
2	"3048"	2020-11-02	116	14.0	126.0
3	"3048"	2020-11-03	97	14.0	156.0
4	"3048"	2020-11-04	207	14.0	144.0
5	"3048"	2020-11-05	285	7.0	123.0
6	"3048"	2020-11-06	220	14.0	147.0
7	"3048"	2020-11-07	334	14.0	153.0
8	"3048"	2020-11-08	156	7.0	90.0
9	"3048"	2020-11-09	171	21.0	132.0
10	"3048"	2020-11-10	97	21.0	147.0
	more				
?????	"3539"	2020-11-30	50	7.0	84.0

```
begin
    filter!(row -> row.footfall != 0, data)
    filter!(row -> row.transactions != 0, data)
end
```

Зададим новые переменные: `conversion` (конверсия) и `CPP` (clicks per person, число кликов в приложении на посетителя).

```
Float64[2.16162, 0.920635, 0.621795, 1.4375, 2.31707, 1.4966, 2.18301, 1.73333, 1.2
```

```
begin
    data[!, "conversion"] = data.transactions ./ data.footfall
    data[!, "CPP"] = data.events ./ data.footfall
end
```

	store_id	date	events	transactions	footfall	conversion	CPP
1	"3048"	2020-11-01	214	7.0	99.0	0.0707071	2.16162

	store_id	date	events	transactions	footfall	conversion	CPP
2	"3048"	2020-11-02	116	14.0	126.0	0.111111	0.920635
3	"3048"	2020-11-03	97	14.0	156.0	0.0897436	0.621795
4	"3048"	2020-11-04	207	14.0	144.0	0.0972222	1.4375
5	"3048"	2020-11-05	285	7.0	123.0	0.0569106	2.31707
6	"3048"	2020-11-06	220	14.0	147.0	0.0952381	1.4966
7	"3048"	2020-11-07	334	14.0	153.0	0.0915033	2.18301
8	"3048"	2020-11-08	156	7.0	90.0	0.0777778	1.73333

```
data
```

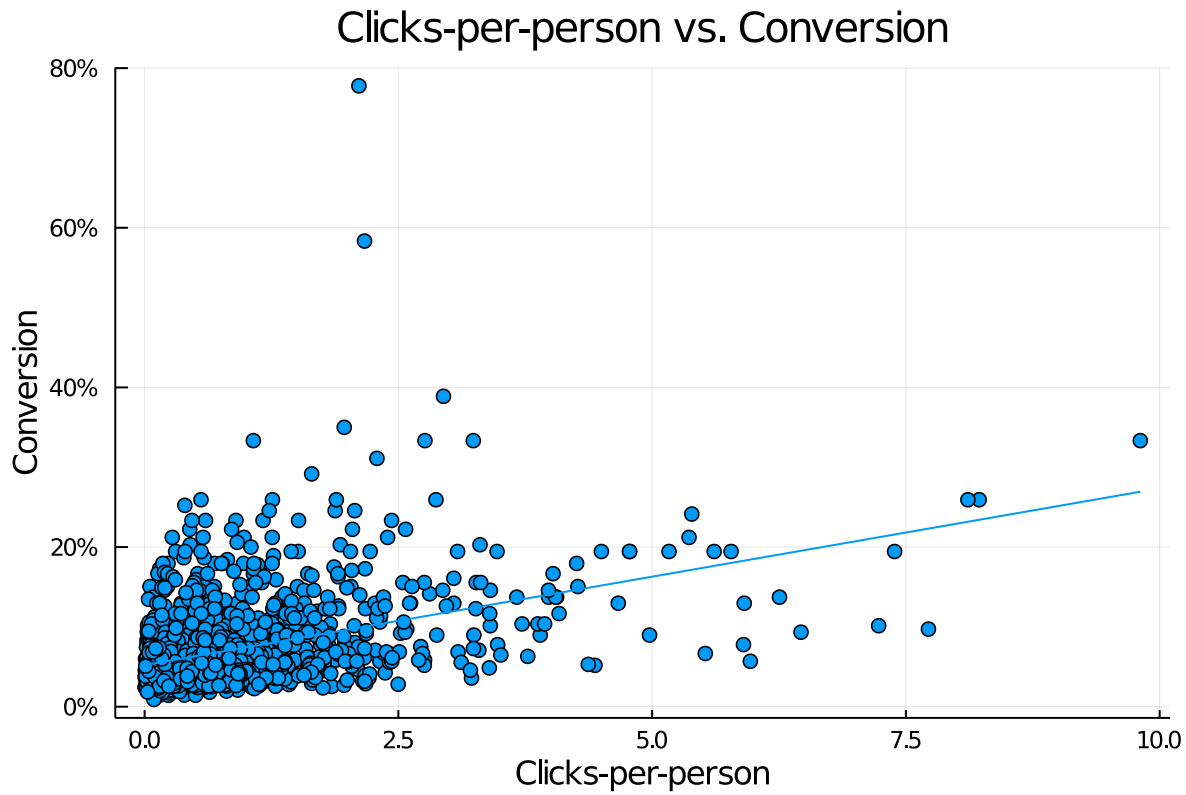
Отфильтруем наблюдения, руководствуясь тем, что конверсия не может превышать 1, а число кликов на посетителя вряд ли превысит 25.

```
data_clean =
```

	store_id	date	events	transactions	footfall	conversion	CPP
1	"3048"	2020-11-01	214	7.0	99.0	0.0707071	2.16162
2	"3048"	2020-11-02	116	14.0	126.0	0.111111	0.920635
3	"3048"	2020-11-03	97	14.0	156.0	0.0897436	0.621795
4	"3048"	2020-11-04	207	14.0	144.0	0.0972222	1.4375
5	"3048"	2020-11-05	285	7.0	123.0	0.0569106	2.31707
6	"3048"	2020-11-06	220	14.0	147.0	0.0952381	1.4966
7	"3048"	2020-11-07	334	14.0	153.0	0.0915033	2.18301
8	"3048"	2020-11-08	156	7.0	90.0	0.0777778	1.73333
9	"3048"	2020-11-09	171	21.0	132.0	0.159091	1.29545
10	"3048"	2020-11-10	97	21.0	147.0	0.142857	0.659864

more

```
data_clean = data[(data[!, "conversion"] .< 1) .& (data[!, "CPP"] .< 10), :]
```




```
plot(data_clean.CPP, data_clean.conversion, seriestype = :scatter, title = "Clicks-per-person vs. Conversion", dpi = 400, yformatter = y -> string(Int64(floor(y * 100)), "%"), smooth = true, label = "", xlabel = "Clicks-per-person", ylabel = "Conversion")
```

Можно заметить, что наблюдается положительная зависимость между интенсивностью пользования приложением и конверсией! Круто!

Гетероскедастичность – это отдельный разговор :)