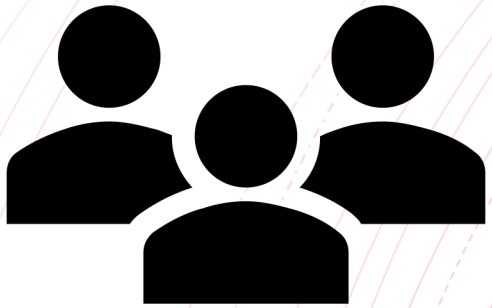
# Simplicity of Complexity Programming Collegium

Data classification and enrichment application based on telephone provider data.

# Состав нашей команды



Захаров Георгий (**analytics**)

ФПМИ МФТИ 2 курс

ВсОШ по математике (**2x** абсолютный победитель)

Харисов Тимур (**manager**)

ФПМИ МФТИ 2 курс

ВсОШ по математике (призер)

Каиров Константин (**analytics**)

ФПМИ МФТИ 2 курс, **ex Huawei RRI**

ВсОШ по математике (победитель)

Мясников Константин (**backend**)

ФПМИ МФТИ 2 курс, **ex Huawei RRI**

**Romanian Masters – Gold Medal**

Алекссев Максим (**backend**)

ФПМИ МФТИ 2 курс, **ex Yandex**

ВсОШ по информатике (призер)

# В чем заключается проблема?

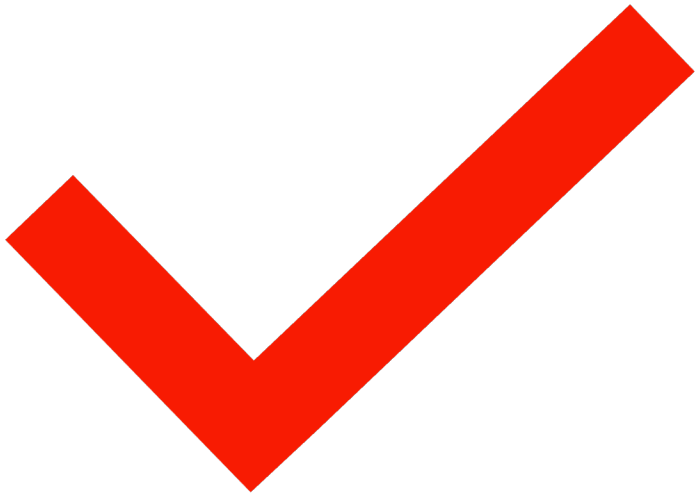
- Есть большие базы данных с информацией, в которых долго разбираться. Маркетологам нужно получить уже обработанную информацию: когда, кому и как посылать. И дополнительная информация о данном пользователе.

# Решение проблемы с помощью нашего продукта

Необходимо отсортировать данные о всех операциях, чтобы из них получить информацию о потреблении абонентом интернет трафика, исходящих и входящих звонков в месяц.

Помимо этого нужно создать систему, которая по **real time** токенам событий типа “вход пользователем в приложение”, выдает рекомендуемое время взаимодействия и тип взаимодействия.

Потом на основании этих данных выдаем готовую таблицу маркетологам в виде итоговой таблицы.



# Преимущества

- Наш проект содержит, как нам кажется следующие преимущества по сравнению с остальными конкурентами.

# Скорость работы

Обработка месячных данных о пользователях происходит за **O(agg_size + par_size)** времени, где **agg_size = size(agg_usage), par_size = size(parrent_operator).**

Алгоритм: Здесь в основном применяются неасимптотические оптимизации.

```
def get_client_profile(data_traffic: DataFrame, voice_traffic: DataFrame) -> DataFrame:
    merged_frame = data_traffic.join(voice_traffic, ['client_id', 'time_key'], 'full')
    merged_frame = merged_frame.withColumn("time_month", SparkFunctions.col("time_key")[0:7])
    df_client_profile = None
    all_columns = merged_frame.columns
    all_columns.remove("client_id")
    all_columns.remove("time_month")
    all_columns.remove("time_key")
    for column in all_columns:
        one_column = merged_frame.groupBy("client_id", "time_month").agg(
            SparkFunctions.sum(column).alias(column))
        if df_client_profile is None:
            df_client_profile = one_column
        else:
            df_client_profile = df_client_profile.join(one_column, ['client_id', 'time_month'], 'full')
    return df_client_profile
```

# Скорость работы

```python
def from_date_to_mobile_consumers(date: str) -> float:
    """turns date into time"""
    hours_f = float(date[11:13])
    minutes_f = float(date[14:16])
    mobile_consumers_f = float(date[17:19])
    mmobile_consumers_f = float(date[20:23]) / 1000
    return mmobile_consumers_f + mobile_consumers_f + minutes_f * 60 + hours_f * 3600


def is_diff_more_than_5(date1: str, date2: str) -> bool:
    """checks if time differs at least in five minutes"""
    mobile_consumers1 = from_date_to_mobile_consumers(date1)
    mobile_consumers2 = from_date_to_mobile_consumers(date2)
    if mobile_consumers1 < mobile_consumers2:
        mobile_consumers1, mobile_consumers2 = mobile_consumers2, mobile_consumers1
    if mobile_consumers1 - mobile_consumers2 > 300:
        return True
    return False
```

Обработка информации о взаимодействии с **web** и **mobile** происходят за следующее время: **O(n + m + N * k)**, где **n = size(mobile_client)**, **m = size(web_client)**, **N** - среднее количество запросов в день, **k** - количество дней. Можно также заметить, что **O(n + m + N * K) = O(n + m + n + m) = O(n + m)**, поскольку **N * k = n + m**.

Алгоритм: проход 2 указателями по **mobile_client** и **web_client**. Во время прохода делим этапы на дни. Для каждого дня создаем аналог **hash_table**, который за **O(N)** позволит хранить, словарь - какому пользователю, что лучше отправить.

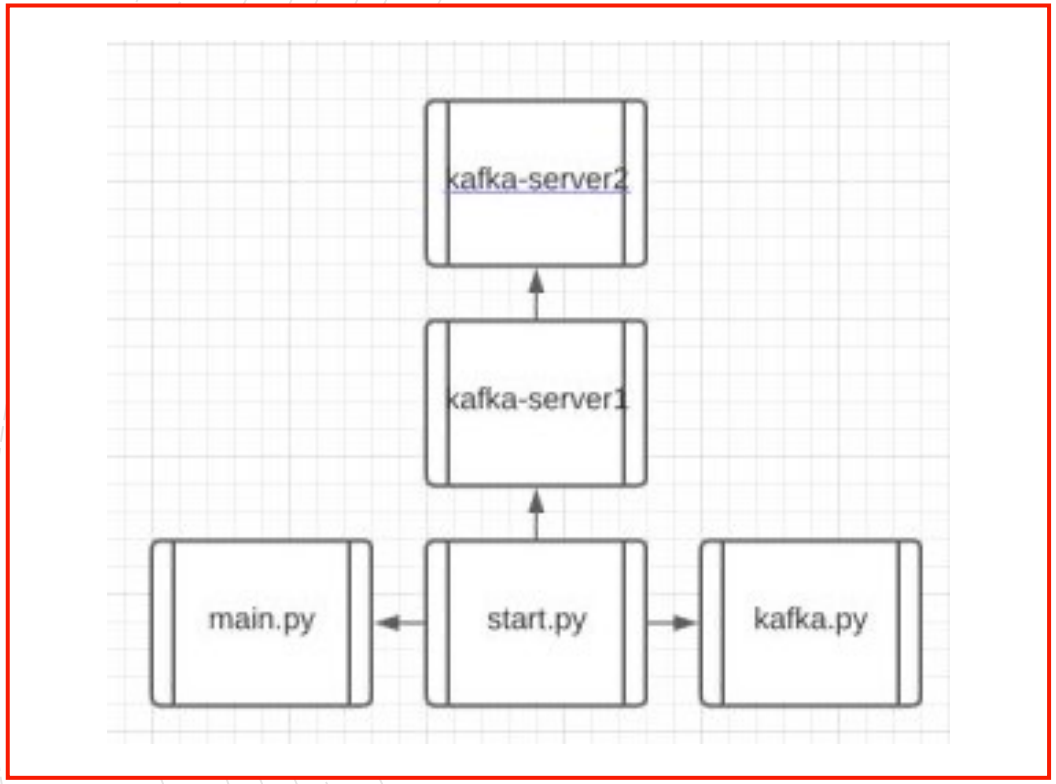


# Использование open-source libraries и python

- **Apache Spark** - для хранения и обработки данных (перекладывали данные из одного файла в другой)
- **Apache Kafka** - для трансляции информации в топики с возможностью дальнейшей работы с ней

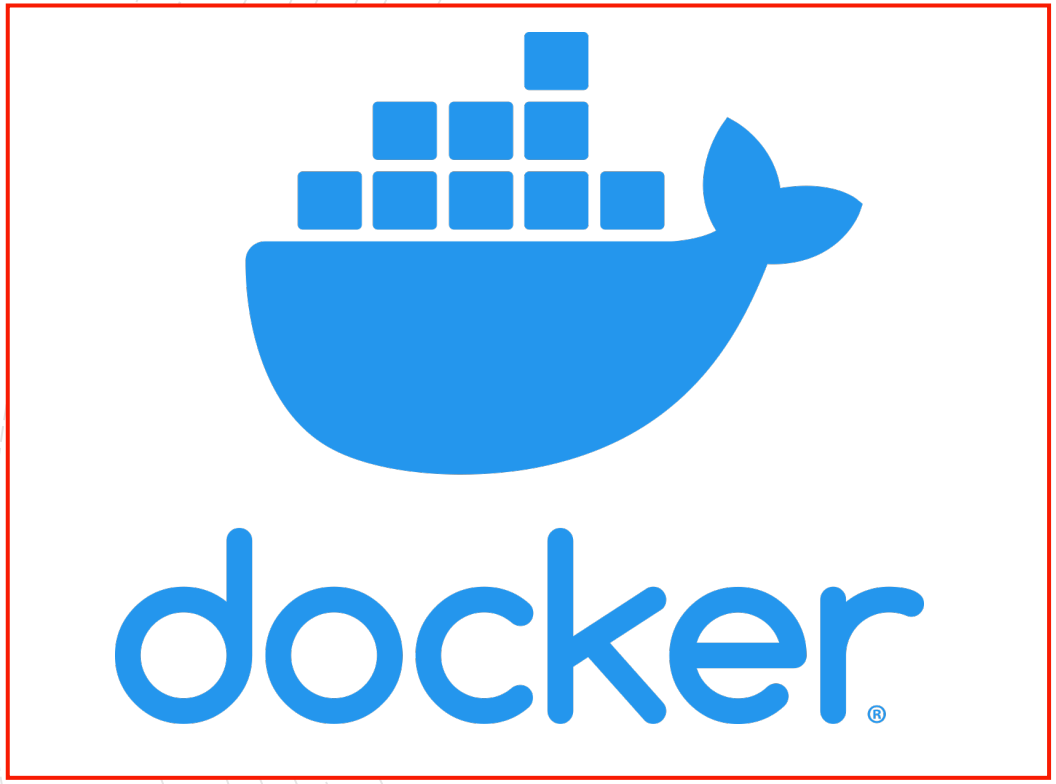
# Демонстрируем как работать с продуктом.

# Информация по запуску продукта в достаточной мере указана в README гитхаба.




- Сам продукт разделен на некоторые вспомогательные файлы для запуска, например, **dockerfile** и **requirments.txt** и основного кода, лежащего в **src:**
  **start.py -** запускает программу.
- **main.py -** основной код для второй задачи, использующий **spark** для перекладывания данных из изначальных табличек в новую **(client_profile).**
- **kafka.py -** набор функций для удобной работы с **apache kafka,** который двумя указателями в **merge-sort-style** проходит по двум топикам **web_client** и **mobile_client** на ходу и поддерживая информацию за последние пять минут определяют наилучший вариант посредством анализа использования клиентом **web** и **mobile** приложений. Благодаря этому отправляет топик **out_cm** для 3-го задания и создает базу **out_cm.json** для четвертого.

# Запускаем Docker-Compose и даем ему необходимые данные




- собираем:

**$ docker build -t SoCPC-image SoCPC-hackaton/**

- запускаем:

**$ docker run --mount src=$(pwd)/data",target="/data/",type=bind --cpus 4 -m 8000M --name SoCPC-image**

- подтягиваем результаты:

**$ docker cp --name SoCPC-image:data/out_cm.json .**

# Получаем итоговую табличку out_cm.json

По сути некоторые шаги из документации.
Полученные данные будут лежать в **docker container** по адресу **/data** Чтобы их скопировать можно воспользоваться командой:

```
$ docker cp --name SoCPC-image:data/out_cm.json .
```

```
{"client_id": "3703718", "eventTime": "2017-10-27T00:10:19.272+03:00", "channel": "sms"}
{"client_id": "4104123", "eventTime": "2017-10-27T00:10:19.272+03:00", "channel": "sms"}
{"client_id": "3976327", "eventTime": "2017-10-27T00:10:19.274+03:00", "channel": "push"}
{"client_id": "1416695", "eventTime": "2017-10-27T00:10:19.289+03:00", "channel": "push"}
{"client_id": "2081766", "eventTime": "2017-10-27T00:10:19.296+03:00", "channel": "sms"}
{"client_id": "2920362", "eventTime": "2017-10-27T00:10:19.296+03:00", "channel": "sms"}
{"client_id": "3729078", "eventTime": "2017-10-27T00:10:19.293+03:00", "channel": "push"}
{"client_id": "0170017", "eventTime": "2017-10-27T00:10:19.296+03:00", "channel": "push"}
{"client_id": "0227266", "eventTime": "2017-10-27T00:10:19.300+03:00", "channel": "sms"}
{"client_id": "3788277", "eventTime": "2017-10-27T00:10:19.301+03:00", "channel": "sms"}
{"client_id": "3203754", "eventTime": "2017-10-27T00:10:19.301+03:00", "channel": "push"}
{"client_id": "3924911", "eventTime": "2017-10-27T00:10:19.310+03:00", "channel": "push"}
{"client_id": "0874027", "eventTime": "2017-10-27T00:10:19.310+03:00", "channel": "sms"}
{"client_id": "0823436", "eventTime": "2017-10-27T00:10:19.314+03:00", "channel": "sms"}
{"client_id": "2991393", "eventTime": "2017-10-27T00:10:19.316+03:00", "channel": "sms"}
{"client_id": "2136961", "eventTime": "2017-10-27T00:10:19.316+03:00", "channel": "sms"}
{"client_id": "4095558", "eventTime": "2017-10-27T00:10:19.321+03:00", "channel": "sms"}
{"client_id": "2146292", "eventTime": "2017-10-27T00:10:19.324+03:00", "channel": "sms"}
{"client_id": "1582671", "eventTime": "2017-10-27T00:10:19.326+03:00", "channel": "push"}
{"client_id": "3032342", "eventTime": "2017-10-27T00:10:19.331+03:00", "channel": "sms"}
{"client_id": "1278416", "eventTime": "2017-10-27T00:10:19.333+03:00", "channel": "sms"}
{"client_id": "1740732", "eventTime": "2017-10-27T00:10:19.339+03:00", "channel": "push"}
{"client_id": "1898649", "eventTime": "2017-10-27T00:10:19.346+03:00", "channel": "push"}
{"client_id": "2593431", "eventTime": "2017-10-27T00:10:19.347+03:00", "channel": "sms"}
{"client_id": "1045700", "eventTime": "2017-10-27T00:10:19.349+03:00", "channel": "push"}
{"client_id": "3798350", "eventTime": "2017-10-27T00:10:19.356+03:00", "channel": "push"}
{"client_id": "1155279", "eventTime": "2017-10-27T00:10:19.359+03:00", "channel": "sms"}
{"client_id": "1007827", "eventTime": "2017-10-27T00:10:19.361+03:00", "channel": "sms"}
{"client_id": "3038302", "eventTime": "2017-10-27T00:10:19.362+03:00", "channel": "sms"}
{"client_id": "2301538", "eventTime": "2017-10-27T00:10:19.365+03:00", "channel": "push"}
{"client_id": "1387307", "eventTime": "2017-10-27T00:10:19.368+03:00", "channel": "sms"}
{"client_id": "0238245", "eventTime": "2017-10-27T00:10:19.372+03:00", "channel": "sms"}
{"client_id": "1020199", "eventTime": "2017-10-27T00:10:19.372+03:00", "channel": "sms"}
{"client_id": "3492202", "eventTime": "2017-10-27T00:10:19.374+03:00", "channel": "sms"}
{"client_id": "1275124", "eventTime": "2017-10-27T00:10:19.379+03:00", "channel": "sms"}
{"client_id": "1039766", "eventTime": "2017-10-27T00:10:19.383+03:00", "channel": "push"}
{"client_id": "3046304", "eventTime": "2017-10-27T00:10:19.384+03:00", "channel": "sms"}
{"client_id": "3365423", "eventTime": "2017-10-27T00:10:19.392+03:00", "channel": "push"}
```


| client_id | time_month | data_all_mb | voice_in_sec | voice_out_sec |
|---|---|---|---|---|
| 0000001 | P201710 | 1.8205888295092132E9 | 407.0 | 1177.0 |
| 0000002 | P201710 | 1.672766867442339... | 4282.0 | 4435.0 |
| 0000003 | P201710 | 1.487477766756704... | 3147.0 | 8452.0 |
| 0000005 | P201710 | 2.313080439514262E10 | 2034.0 | 1617.0 |
| 0000006 | P201710 | null | 192.0 | 208.0 |
| 0000008 | P201710 | null | 4276.0 | 2967.0 |
| 0000009 | P201710 | null | 1079.0 | 192.0 |
| 0000010 | P201710 | 4.7823301912166847E7 | 3022.0 | 3266.0 |
| 0000011 | P201710 | null | 3025.0 | 576.0 |
| 0000013 | P201710 | null | 14111.0 | 8284.0 |
| 0000015 | P201710 | 4.0604434321157746E9 | 818.0 | 1035.0 |
| 0000016 | P201710 | 1.580830817353946... | 2657.0 | 3317.0 |
| 0000018 | P201710 | 2.6664082322742467E9 | 195.0 | 96.0 |
| 0000019 | P201710 | 5.039180609168422E9 | 3610.0 | 9239.0 |
| 0000020 | P201710 | 1.620904351051044E10 | 5008.0 | 10344.0 |
| 0000021 | P201710 | 9.577043783255394E9 | 5960.0 | 5654.0 |
| 0000022 | P201710 | 1.9021193985822606E8 | 1380.0 | 528.0 |
| 0000023 | P201710 | 9.214331479376513E9 | 1152.0 | 1347.0 |
| 0000024 | P201710 | 1.0272974532247381E8 | 6418.0 | 718.0 |
| 0000025 | P201710 | 6.742942269064984E8 | 11741.0 | 3586.0 |

# Итоги

Продукт быстро, легко и удобно транслирует данные и обогащает их, давая высококачественные возможности для дальнейшего анализа и изучения поставленных данных.

Достигается такая эффективность уникальными алгоритмическими решениями (поддержка линейности двумя указателями) и использованием соответствующих фреймворков (**apache kafka**, **apache spark**).