Отчёт о продажах в Телеграм

Вы работаете аналитиком в стартапе в области обучения взрослых английскому языку. Компания растет быстро, целый штат маркетологов и продактов придумывает новые механики привлечения пользователей, улучшая коммерческие продукты.

Ещё есть отдел продаж. В нём сотрудники звонят потенциальным клиентам и продают пакеты уроков. Только вот маркетинг не дружит с продажами и красивой сквозной аналитики у Руководителя отдела продаж нет. Нужно ему помочь.

Никакого централизованного DWH или сложного BI - у компании нет, но бизнесу точно нужны основные метрики, причем завтра. Данные об основных событиях CJM пользователя записываются в Postgres.

Руководитель отдела продаж просит вас каждый день присылать в его telegram метрики или графики. Они должны помочь ему понимать как идут дела в отделе, так сказать держать руку на пульсе.

Данные

Все таблицы с данными в PostgresSQL. Параметры для подключения:

db_name = "quest-db",
db_login = "rouser",
db_passwd = "ZI6MVnmi",
db_host = "178.62.242.91",
db_port = 5433

Задачи

1. Выберите 3 метрики, которые помогут руководителю отдела продаж контролировать ситуацию ежедневно — всё ли идёт нормально. Объясните свой выбор.
2. Напишите Телеграм Бота (скрипт), который будет отправлять ежедневный отчёт по этим метрикам в Телеграм руководителю. Чтобы показать, как работает бот вставьте его код в файл с ответом и прикрепите скриншот отправленного им сообщения, чтобы было видно от кого это. Под названием отправителя должно быть написано Бот, как на скриншоте ниже.

In [1]:
```
import numpy as np
import pandas as pd

import seaborn as sns
import matplotlib.pyplot as plt

from sqlalchemy import create_engine
from sqlalchemy import text

import retentioneering

from sklearn.metrics import silhouette_score
```

In [2]:
```
postgres_uri = 'postgresql://rouser:ZI6MVnmi@178.62.242.91:5433/quest-db'
engine = create_engine(postgres_uri)
```

```
retentioneering.config.update({
    'user_col': 'user_id',
    'event_col':'event_type',
    'event_time_col':'event_time',
})
```

Загрузка данных

In [8]:

```
query = """
select
        e._user_id as user_id,
        e.happened_at as event_time,
        ed._description as event_type
from events e
left join events_dict ed
        on e.event_id = ed.id

union all

select
        p._user_id as user_id,
        p.transaction_created_at as happened_at,
        'Покупка' as event_type
from
payments p
"""

with engine.connect() as con:
    df = pd.read_sql(query, con)

df.head()
```

Out[8]:

	user_id	event_time	event_type
0	12426747	2022-01-21 17:01:22	Создание заявки
1	12426754	2021-06-10 11:07:07	Создание заявки
2	12426754	2021-06-10 11:08:18	Создание заявки
3	12426779	2021-06-10 11:10:03	Создание заявки
4	12427176	2021-06-10 11:48:09	Создание заявки

Рассмотрим каждую транзакцию по оплате, как отдельное событие.

Метрики

Бизнес процессы

1. Вводный урок — основная воронка
2. Демо урок — экпериментальная воронка
3. Первая линия — поддержка
4. Вторая линия — поддержка
5. WA — экпериментальная поддержка

Анализировать будем только основные бизнес процессы: вводный урок, первая линия, вторая линия. Нет смысла анализировать эксперименты по демо уроку и wa в рамках общего ежедневного отчета. Для экспериментов нужен отдельный отчет, тк нам не известны ни параметры, ни кол-во экспериментов.

Исключим пользователей, участвующих в экспериментах, из выборки.

In [9]:
```
main_flow = ['Создание заявки', 'Покупка']
demo_flow = ['Переход на ДУ (ДУ начался)', 'ДУ завершен']
intro_lesson_flow = ['Назначение ВУ', 'Выход МВУ на ВУ', 'Успешный ВУ']
operator_1_flow = ['Назначение задачи на звонок 1Л', 'Ученик ответил на звонок оператора 1
operator_2_flow = ['Назначена задача на вторую линию', 'Дозвон 2Л']
wa_flow = ['Отправка сообщения WA', 'У ответил на сообщение WA']

main_events = [*main_flow, *intro_lesson_flow, *operator_1_flow, *operator_2_flow]
```

In [11]:
```
df['is_ab'] = df.event_type.isin([demo_flow, wa_flow])
users = df.groupby('user_id').is_ab.max()
users_not_ab = users[users == False].index
df = df[df.user_id.isin(users_not_ab)][['user_id', 'event_time', 'event_type']]
df.event_time = pd.to_datetime(df.event_time)
```

In [5]:
```
df['is_ab'] = df.event_type.isin([*demo_flow, *wa_flow])
users = df.groupby('user_id').is_ab.max()
users_not_ab = users[users == False].index
df = df[df.user_id.isin(users_not_ab)][['user_id', 'event_time', 'event_type']]
df.event_time = pd.to_datetime(df.event_time)

df.head()
```

Out[5]:

	user_id	event_time	event_type
0	12446294	2021-06-13 06:58:29	Назначение задачи на звонок 1Л
1	12446328	2021-06-13 07:28:54	Назначение задачи на звонок 1Л
2	12446536	2021-06-13 08:22:04	Назначение задачи на звонок 1Л
3	12446929	2021-06-13 11:39:33	Назначение задачи на звонок 1Л
4	12446941	2021-06-13 10:03:22	Назначение задачи на звонок 1Л

Незакрытые бизнесс процессы

Работу отдела можно свести к 4 основным бизнес процессам (БП):

- Обработка пользовательских заявок:
  - Начало: "Создания заявки"
  - Конец: "Назначение ВУ" или "Назначение задачи на звонок 1Л", "Назначена задача на вторую линию"
- Оплата:
  - Начало: "Успешный ВУ"
  - Конец: "Оплата", "Назначена задача на вторую линию" (если пользователь не оплатил сразу)
- Первая линия поддержки:
  - Начало: "Назначение задачи на звонок 1Л"
  - Конец: "Ученик ответил на звонок оператора 1л"
- Вторая линия поддержки:

- Начало: "Назначена задача на вторую линию"
- Конец: "Дозвон 2Л"

Предположим, что на выполнение каждого БП отведено 24 часа. Тогда все БП, которые не были закрыты за это время будут помечены как проблемные — блокированы, или нехватка ресурсов.

Что показывает метрика:

- Нагрузка на отдел
- Узкие места в БП

In [12]:
```
def get_day_boundaries(date):
    template = '%Y-%m-%d %H:%M:%S'
    t1 = pd.to_datetime(f'{date} 00:00:00', format=template)
    t2 = pd.to_datetime(f'{date} 23:59:59', format=template)

    return t1, t2


def get_daily_raw(data, date):
    t1, t2 = get_day_boundaries(date)
    time_mask = (data['event_time'] > t1) & (data['event_time'] < t2)

    data = data[time_mask]

    lead = df.sort_values(by=['user_id', 'event_time']).groupby('user_id').shift(-1)
    data = df.join(lead.rename(columns=lambda x: 'next_' + x))

    return data[time_mask]


def get_daily(data, date):
    t1, t2 = get_day_boundaries(date)
    time_mask = (data['event_time'] > t1) & (data['event_time'] < t2)

    return data[time_mask]


def get_daily_pivot(data, date, columns=None):
    data = get_daily(data, date)
    data = data.groupby(['user_id', 'event_type']).event_time.agg(max).unstack()

    for col_name in columns:
        if col_name not in data.columns:
            data[col_name] = pd.NaT

    return data
```

In [13]:
```
date = '2022-01-30'

pipelines = {
    'Открытые заявки': {
        'start': 'Создание заявки',
        'end': [*intro_lesson_flow, *operator_1_flow, *operator_2_flow],
    },
    'Первая линия': {
        'start': 'Назначение задачи на звонок 1Л',
        'end': ['Ученик ответил на звонок оператора 1л'],
    },
    'Вторая линия': {
        'start': 'Назначена задача на вторую линию',
```

```
        'end': ['Дозвон 2Л'],
    },
    'Оплата ВУ': {
        'start': 'Успешный ВУ',
        'end': ['Покупка', 'Назначена задача на вторую линию'],
    },
}

full_report = {}
for name, pipeline in pipelines.items():
    pivot = get_daily_pivot(df, date, columns=main_events)

    start_event, end_events = pipelines[name].values()

    start_mask = pivot[start_event].notna()
    any_end_mask = pivot[end_events].notna().any(axis=1)

    full_report[name] = len(pivot[start_mask & ~any_end_mask])

fig = pd.DataFrame(full_report, index=[0]).T.sort_values([0]).plot.barh(
    title=f'Незакрытые бизнес процессы, {date}', legend=False)
```

Пример:
2022-01-30 осталось более 20 открытых заявок, необработанных в течение дня. Но серезной нагрузки на первую линию поддержки нет — повод разобраться почему висят завяки.

Воронка конверсии

Наш идеальный сценарий:

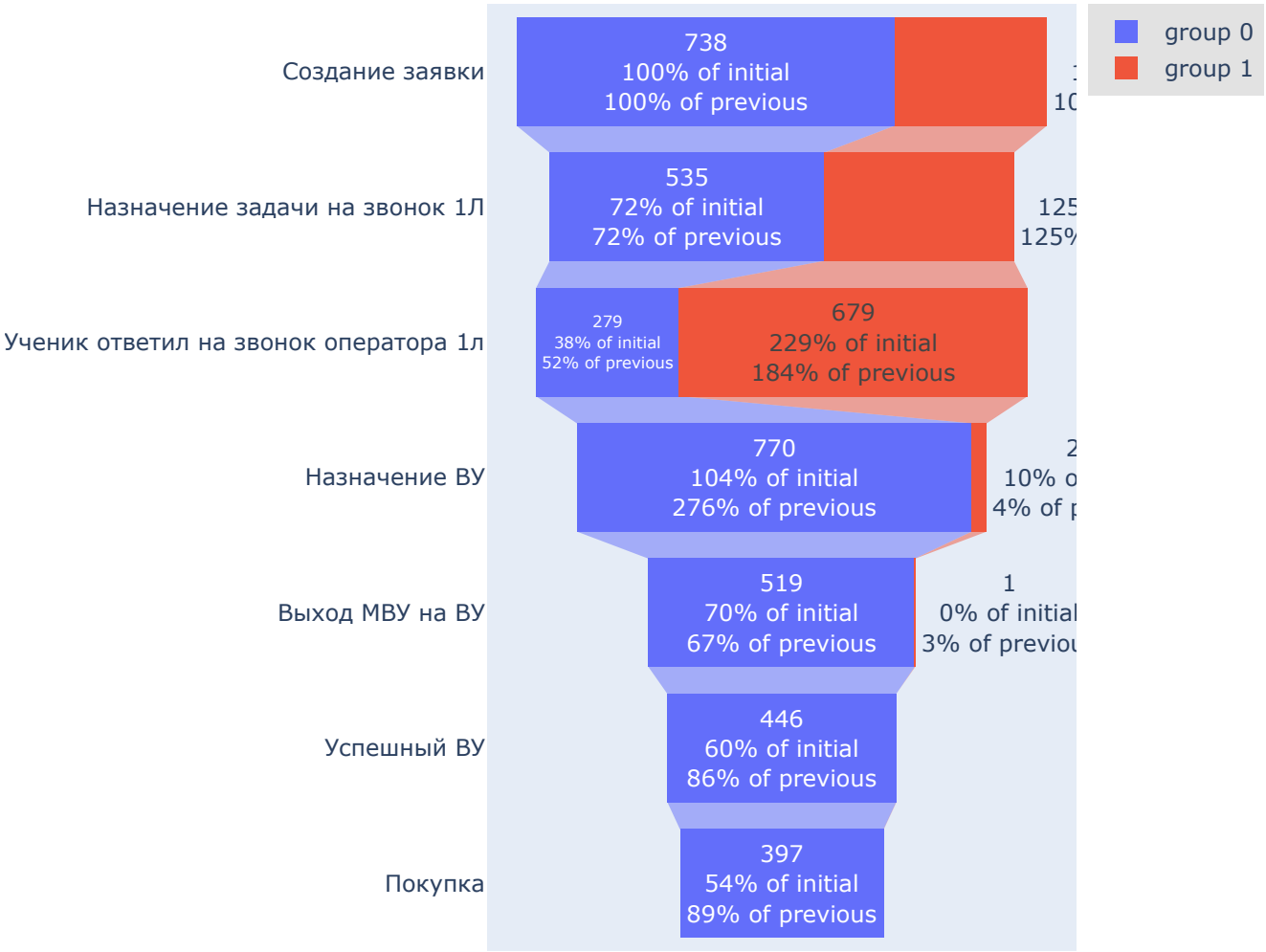
1. Создание завяки пользователем
2. Создание задачи для первой линии
3. Дозвон первой линии до пользователя
4. Назначение вводного урока
5. Выход методиста на вводный урок
6. Успешный вводный урок
7. Покупка

Произведем кластеризацию пользователей, и на каждом этапе посчитаем конверсию для кластеров. Результат виуализируем в виде воронки.

Что показывает метрика:

- Отток пользователей на каждом этапе в разрезе кластеров

In [14]:
```
date = '2021-06-12'

flow = ['Создание заявки', *operator_1_flow, *intro_lesson_flow, 'Покупка']

data = get_daily(df, date)

n_clusters = 2

data.rete.get_clusters(method='kmeans',
                       feature_type='tfidf',
                       n_clusters=n_clusters,
                       ngram_range=(1, 2),
                       targets=flow)

clusters_ids = [data.rete.cluster_mapping[i] for i in range(n_clusters)]
data.rete.funnel(targets=flow, groups=clusters_ids)
```



Количество новых платных пользователей

Для мониторинга роста используем простую метрику кол-ва новых платных пользователей. Метрика просто интерпретируются, и тем самым подходит для ежедневного отчета.

In [15]:
```
data = get_daily(df, '2022-02-12')

report_date, _ = get_day_boundaries('2021-06-12')
```

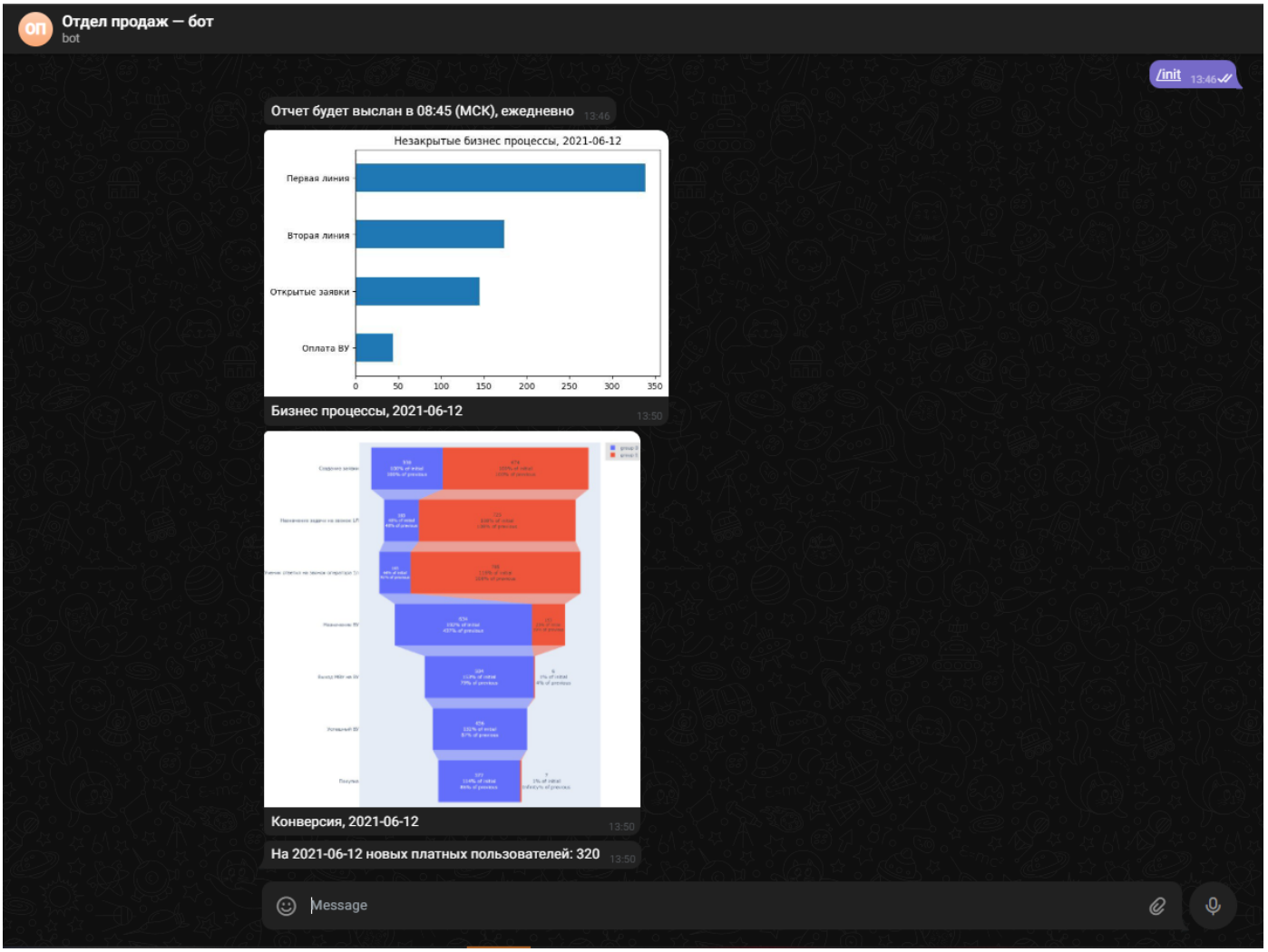
```
today_payed_users = data[data.event_type == 'Покупка'].user_id.unique()
before_payed_users = df[(df.event_type == 'Покупка') & (df.event_time < report_date)].user
new_payed_users = [x for x in today_payed_users if x not in before_payed_users]
len(new_payed_users)
```

Out[15]: 14

Бот

@sales_dep_report_bot

Скриншоты



Код

sales_dep_report_bot/

- reports/
  - _ *init* _.py
  - conversion_funnel.py
  - hangin_workflows.py
  - new_paing_user.py
  - utils.py

    - workflows.py
- .env
- db.py
- main.py

main.py

In [ ]:

```
# @sales_dep_report_bot

from dotenv import load_dotenv
load_dotenv()

import os

import logging
import datetime
import pytz

from telegram import Update
from telegram.ext import Updater, CommandHandler, CallbackContext

from reports.utils import load_data
from reports import hanging_workflows, conversion_funnel, new_paying_users


logging.basicConfig(
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s', level=logging.INFO
)

logger = logging.getLogger(__name__)


def report(context: CallbackContext) -> None:
    """Формирует отчеты."""
    job = context.job

    date = datetime.date.today()
    date = '2021-06-12'
    data = load_data()

    hanging_img = open(hanging_workflows.get_plot(data, date), 'rb')
    funnel_img = open(conversion_funnel.get_plot(data, date), 'rb')
    new_paying_count = new_paying_users.get_report(data, date)

    context.bot.send_photo(job.context, photo=hanging_img, caption=f'Бизнес процессы, {dat
    context.bot.send_photo(job.context, photo=funnel_img, caption=f'Конверсия, {date}')
    context.bot.send_message(job.context, text=f'На {date} новых платных пользователей: {r


def remove_job_if_exists(name: str, context: CallbackContext) -> bool:
    """Удаляет джоб из расписания."""
    current_jobs = context.job_queue.get_jobs_by_name(name)
    if not current_jobs:
        return False
    for job in current_jobs:
        job.schedule_removal()
    return True


def start(update: Update, context: CallbackContext) -> None:
    """Оправляет приветствие и инструкции дл начала работы"""
    update.message.reply_text('Отчет отдела продажа. Для начала работы введите /init')
```

```
def init(update: Update, context: CallbackContext) -> None:
    """Запускает джоб."""
    chat_id = update.message.chat_id
    remove_job_if_exists(str(chat_id), context)
    time = datetime.time(hour=8, minute=45, tzinfo=pytz.timezone('Europe/Moscow'))
    context.job_queue.run_daily(report, time, context=chat_id, name=str(chat_id))

    update.message.reply_text('Отчет будет выслан в 08:45 (МСК), ежедневно')


def main() -> None:
    updater = Updater(os.getenv('BOT_TOKEN'))

    dispatcher = updater.dispatcher

    dispatcher.add_handler(CommandHandler("start", start))
    dispatcher.add_handler(CommandHandler('init', init))

    updater.start_polling()
    updater.idle()


if __name__ == '__main__':
    main()
```

db.py

In [ ]:

```
import os
from sqlalchemy import create_engine
import pandas as pd


engine = create_engine(os.getenv('DB_URI'))


def query(sql):
    with engine.connect() as con:
        return pd.read_sql(sql, con)
```

reports/conversion_funnel.py

In [ ]:

```
import reports.workflows as flow
from reports.utils import get_daily

import retentioneering

retentioneering.config.update({
    'user_col': 'user_id',
    'event_col': 'event_type',
    'event_time_col': 'event_time',
})

targets = ['Создание заявки', *flow.operator_1_flow, *flow.intro_lesson_flow, 'Покупка']


def get_plot(df, date):
    df = get_daily(df, date)
```

```
    n_clusters = 2

    df.rete.get_clusters(method='kmeans',
                         feature_type='tfidf',
                         n_clusters=n_clusters,
                         ngram_range=(1, 2),
                         targets=targets)

    clusters_ids = [df.rete.cluster_mapping[i] for i in range(n_clusters)]
    fig = df.rete.funnel(targets=targets, groups=clusters_ids)

    path = 'conversion_funnel.jpeg'
    fig.write_image(path, engine='kaleido', width=1000, height=1000, scale=2)

    return path
```

reports/hanging_workflows.py

In [ ]:

```
import pandas as pd

import reports.workflows as flow
from reports.utils import get_daily_pivot

pipelines = {
    'Открытые заявки': {
        'start': 'Создание заявки',
        'end': [*flow.intro_lesson_flow, *flow.operator_1_flow, *flow.operator_2_flow],
    },
    'Первая линия': {
        'start': 'Назначение задачи на звонок 1Л',
        'end': ['Ученик ответил на звонок оператора 1л'],
    },
    'Вторая линия': {
        'start': 'Назначена задача на вторую линию',
        'end': ['Дозвон 2Л'],
    },
    'Оплата ВУ': {
        'start': 'Успешный ВУ',
        'end': ['Покупка', 'Назначена задача на вторую линию'],
    },
}


def get_plot(df, date):
    """
    Формирует отчет по незавершенным бизнес процессам
    :param df: Фрейм с пользовательскими событиями
    :param date: День на кторый формируется отчет ('2022-03-14')
    :return: Путь к файлу с графиком отчета
    """
    full_report = {}
    for name, pipeline in pipelines.items():
        pivot = get_daily_pivot(df, date, columns=flow.main_events)

        start_event, end_events = pipelines[name].values()

        start_mask = pivot[start_event].notna()
        any_end_mask = pivot[end_events].notna().any(axis=1)

        full_report[name] = len(pivot[start_mask & ~any_end_mask])

    ax = pd.DataFrame(full_report, index=[0]).T.sort_values([0]).plot.barh(
        title=f'Незакрытые бизнес процессы, {date}', legend=False)
```

```
    path = 'hanging_workflows.jpeg'
    ax.get_figure().savefig(path, bbox_inches='tight')

    return path
```

reports/new_paing_user.py

In [ ]:

```
from reports.utils import get_day_boundaries, get_daily


def get_report(df, date):
    today_df = get_daily(df, date)

    report_date, _ = get_day_boundaries(date)

    today_payed_users = today_df[today_df.event_type == 'Покупка'].user_id.unique()
    before_payed_users = df[(df.event_type == 'Покупка') & (df.event_time < report_date)].
    new_payed_users = [x for x in today_payed_users if x not in before_payed_users]

    return len(new_payed_users)
```

reports/utils.py

In [ ]:

```
import reports.workflows as flow
from db import query
import pandas as pd


def load_data():
    sql = """
    select
        e._user_id as user_id,
        e.happened_at as event_time,
        ed._description as event_type
    from events e
    left join events_dict ed
        on e.event_id = ed.id

    union all

    select
        p._user_id as user_id,
        p.transaction_created_at as happened_at,
        'Покупка' as event_type
    from
    payments p
    """

    df = query(sql)

    df['is_ab'] = df.event_type.isin([*flow.demo_flow, *flow.wa_flow])
    users = df.groupby('user_id').is_ab.max()
    users_not_ab = users[users == False].index
    df = df[df.user_id.isin(users_not_ab)][['user_id', 'event_time', 'event_type']]
    df.event_time = pd.to_datetime(df.event_time)

    return df


def get_day_boundaries(date):
```

```
    template = '%Y-%m-%d %H:%M:%S'
    t1 = pd.to_datetime(f'{date} 00:00:00', format=template)
    t2 = pd.to_datetime(f'{date} 23:59:59', format=template)

    return t1, t2


def get_daily(data, date):
    t1, t2 = get_day_boundaries(date)
    time_mask = (data['event_time'] > t1) & (data['event_time'] < t2)

    return data[time_mask]


def get_daily_pivot(data, date, columns=None):
    data = get_daily(data, date)
    data = data.groupby(['user_id', 'event_type']).event_time.agg(max).unstack()

    for col_name in columns:
        if col_name not in data.columns:
            data[col_name] = pd.NaT

    return data
```

reports/workflows.py

In [ ]:

```
main_flow = ['Создание заявки', 'Покупка']
demo_flow = ['Переход на ДУ (ДУ начался)', 'ДУ завершен']
intro_lesson_flow = ['Назначение ВУ', 'Выход МВУ на ВУ', 'Успешный ВУ']
operator_1_flow = ['Назначение задачи на звонок 1Л', 'Ученик ответил на звонок оператора 1
operator_2_flow = ['Назначена задача на вторую линию', 'Дозвон 2Л']
wa_flow = ['Отправка сообщения WA', 'У ответил на сообщение WA']

main_events = [*main_flow, *intro_lesson_flow, *operator_1_flow, *operator_2_flow]
```