Карьерный цех
3-й конкурс Аналитиков весна 2022
Задание 1

Цель: составить для руководителя отдела продаж отчет с тремя показателями, обеспечить ежедневную автоматическую рассылку в телеграм бот для оперативного получения информации.

Задачи:

- настроить подключение к БД Postgres
- определить актуальные метрики для ежедневного анализа
- выгрузить необходимые данные из таблиц БД
- обработать данные и рассчитать метрики
- построить графики за выбранный интервал (последние 7 дней)
- написать телеграм бота, который будет представлять обработанные данные
- настроить периодичность отправки сообщений
- обработать возможные ошибки

Ограничения:

При выгрузке данных из БД была выявлена неконсистентность данных между таблицами events и payment (последняя запись в events 2022-03-14 17:43:58, последняя запись в payments 2022-03-15 11:43:41). Поэтому результирующий отчет сформирован на последнюю дату, по которой есть все необходимые данные для расчета метрик – 14.03.2022.

Используемые методы и инструменты:

Matplotlib, pandas, psycopg2, python-dotenv, python-telegram-bot.

Решение:

Выбранные для ежедневного анализа метрики и их обоснование:

1. Ежедневная выручка (Daily revenue)

Метрика позволяет оценить текущий уровень продаж за расчетный период (день), а в совокупности с предыдущими показателями на графике еще и оценить динамику продаж: как менялся показатель выручки, например, за неделю.

Метрика рассчитывается как: сумма всех проведенных за день транзакций

2. Средний доход от каждого клиента в день (ARPU - Average revenue per user)

Данная метрика позволяет понять, какой доход приносит каждый клиент за конкретный период (день). Не путать со средним чеком (AOV), т. к. в нашем случае средний чек покажет только средний доход от одной транзакции в день.

ARPU показывает, сколько клиенты готовы платить за услугу. Например, если в компании несколько продуктов по разным ценовым предложениям, ARPU продемонстрирует, какое ценовое предложение предпочитает большинство клиентов. На основании этих данных можно строить маркетинговые стратегии по привлечению клиентов и выстраивать линейку тарифов.

Метрика рассчитывается как: общая выручка за день /число уникальных клиентов за день

Дополнительно рассчитано количество оплативших клиентов. Данный показатель может быть полезен в дальнейшем при расчете конверсии.

3. Количество клиентов, создавших заявку(-и) за день (SOC – созданные возможности продаж (Sales opportunities created))

Данная метрика фиксирует заинтересованных клиентов, помогает оценить эффективность процесса продаж, воронки, а также прогнозировать выручку. Позволяет ответить на вопросы:

- Эффективны ли каналы продаж? Можно оценить количество поступающих заявок от клиентов и сравнить показатели за определенный интервал.

- Эффективна ли команда продвижения? Можно сопоставить количество созданных заявок/возможностей с количеством звонков на одного менеджера (для расчета необходимо понимать количество менеджеров в компании).
- Эффективен ли используемый скрипт продаж? Можно сравнить число созданных заявок/возможностей с числом состоявшихся сделок (в дальнейшем можно будет посчитать конверсию лидов в сделки за месяц/год, но в данной задаче расчет конверсии за интервал в 1 день не актуален, т. к. этапы продаж зачастую занимают более длительное время).

Метрика рассчитывается как: количество уникальных клиентов за день, создавших заявку(-и)

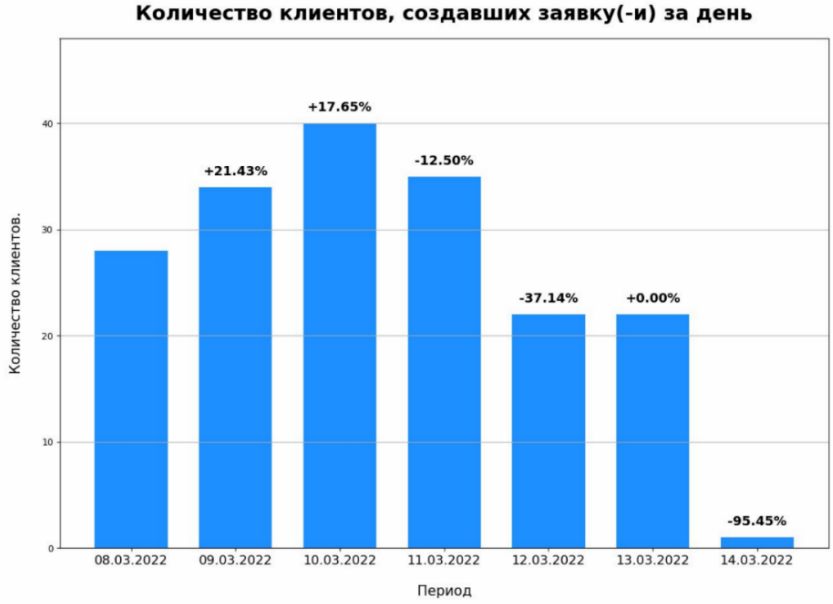
В качестве этапа, определяющего начало взаимодействия с клиентом, был определен этап «Создание заявки» (event_id == 3), т. к., исходя из представленных данных (таблицы БД events и events_dict), 99.77% всех пользователей, обратившихся в компанию, проходят через этот этап.

Результат отправки сообщения с метриками и графиками в телеграм:

В телеграм бот графики отправляются в групповом сообщении, при открытии они масштабируются под экран устройства. Графики составлены за последние 7 дней (данный параметр опционален), по которым в БД представлена информация. На графиках отражена динамика прироста/падения показателя в процентах к предыдущему дню.

Вывод: полученные метрики позволяют оперативно и наглядно оценить текущую ситуацию в продажах, посмотреть динамику за последнюю неделю, а также могут быть использованы в дальнейшем для более глубокого анализа и построения статистических отчетов за длительный период времени.

Приложения: графики и листинг кода на Python



Требования к запуску скрипта

Интерпретатор:
Python 3.7.9

Служебные пакеты:
matplotlib==3.5.1
pandas==1.3.5
psycopg2-binary==2.9.3
python-dotenv==0.19.2
python-telegram-bot==13.11

Определить значения переменных окружения в .env:
TELEGRAM_TOKEN - Токен телеграм бота
PG_DATABASE - Имя базы данных PostgreSQL
PG_HOST - Имя хоста PostgreSQL
PG_PORT - Порт PostgreSQL
PG_USER - Пользователь PostgreSQL
PG_PASSWORD - Пароль пользователя PostgreSQL
INTERVAL - Интервал за какой период от актуальной даты необходимо отразить данные на графиках
RETRY_TIME - Интервал получения и отправки данных

```
import os
import textwrap
import time
from typing import Optional

import matplotlib.pyplot as plt
import pandas as pd
import psycopg2
import telegram
from dotenv import load_dotenv
from telegram.ext import Filters, MessageHandler, Updater
from telegram import TelegramError
load_dotenv()

# Токен телеграм бота
TELEGRAM_TOKEN = os.getenv('TELEGRAM_TOKEN')

# Учетные данные подключения к БД
PG_DATABASE: Optional[str] = os.getenv('PG_DATABASE')
PG_HOST: Optional[str] = os.getenv('PG_HOST')
PG_PORT: Optional[str] = os.getenv('PG_PORT')
PG_USER: Optional[str] = os.getenv('PG_USER')
PG_PASSWORD: Optional[str] = os.getenv('PG_PASSWORD')

# Интервал, за какой период от актуальной даты
# необходимо отразить данные на графиках
INTERVAL: Optional[str] = os.getenv('INTERVAL')

# Интервал получения данных
RETRY_TIME: Optional[str] = os.getenv('RETRY_TIME')

# Запрос данных из таблиц БД за последний INTERVAL
events: str = (
```

```
    f"""select * from events where to_date(happened_at,'YYYY-MM-DD') >=
    to_date((SELECT MAX(happened_at) from events),'YYYY-MM-DD') -
    interval '{INTERVAL} days' order by happened_at""")

payments: str = (
    f"""select *
    from payments where to_date(transaction_created_at,'YYYY-MM-DD') >=
    to_date((SELECT MAX(transaction_created_at) from payments),'YYYY-MM-
DD') -
    interval '{INTERVAL} days' order by transaction_created_at""")

# Структура хранения подписчиков бота
chat_list: list = []


def bot_initialize(update, context) -> None:
    """Отправка сообщения после подписки на бота"""
    chat = update.effective_chat
    chat_list.append(chat.id)
    context.bot.send_message(
        chat_id=chat.id,
        text=f'Отправка метрик происходит раз в {RETRY_TIME} сек. '
             f'Данный параметр опционален и может быть '
             f'установлен в любое требуемое значение.')


def connect_database():
    """Подключение к БД"""
    try:
        engine = psycopg2.connect(
            f'dbname={PG_DATABASE} \
            user={PG_USER} \
            password={PG_PASSWORD} \
            host={PG_HOST} port={PG_PORT} \
            connect_timeout=10'
        )
        return engine
    except psycopg2.OperationalError:
        print(textwrap.dedent("""
              Произошла ошибка при подключении к базе данных.
              Переподключение через 60 сек."""))
        time.sleep(60)
        connect_database()
    except psycopg2.Error:
        print(textwrap.dedent("""
              Произошла ошибка при подключении к базе данных.
              Переподключение через 60 сек."""))
        time.sleep(60)
        connect_database()


def select(sql, engine) -> pd.core.frame.DataFrame:
    """Функция выборки из базы."""
    try:
        result = pd.read_sql(sql, engine)
        return result
    except Exception:
```

```
        print("Ошибка при выполнении SQL запроса")


def check_tokens() -> bool:
    """Функция проверки токенов возвращает True или False."""
    tokens = [TELEGRAM_TOKEN,
              PG_DATABASE,
              PG_HOST,
              PG_PORT,
              PG_USER,
              PG_PASSWORD,
              INTERVAL,
              RETRY_TIME]
    if not all(tokens):
        message = 'Переменные окружения не определены в ENV'
        print(message)

    return all(tokens)


def get_plot(x, y, title, xlabel, ylabel, fig_name, bar_color) -> None:
    """Создание графика."""
    plt.figure(figsize=(15, 10))
    bars = plt.bar(x, y, width=0.7, color=bar_color)
    plt.title(title, fontsize=22,
              fontweight="bold", pad=20)
    if int(INTERVAL) > 7:
        plt.xticks(fontsize=13, rotation=45)
    else:
        plt.xticks(fontsize=14, rotation=0)

    plt.bar_label(bars, [''] + [f'{(y1 - y0) / y0 * 100:+.2f}%' for y0,
                                y1 in zip(y[:-1], y[1:])],
                  fontsize=14, padding=10, fontweight="bold")
    plt.grid(axis='y')
    plt.xlabel(xlabel, fontsize=15, labelpad=20)
    plt.ylabel(ylabel, fontsize=15, labelpad=20)
    plt.margins(0.05, 0.2)
    plt.savefig(f'{fig_name}.png')


def get_metrics(payments_df, events_df) -> str:
    """Получение метрик."""

    # 1 метрика - ежедневная выручка
    payments_df['transaction_created_at'] = pd.to_datetime(
        payments_df['transaction_created_at'])

    payments_df = payments_df.astype({'amount': 'float'})

    payments_df['date_of_day'] = \
        payments_df.transaction_created_at.dt.strftime('%d.%m.%Y')

    payments_metrics = payments_df\
        .groupby('date_of_day', as_index=False)\
        .agg({'amount': 'sum', '_user_id': pd.Series.nunique})\
        .sort_values('date_of_day', ascending=False)\
```

```
        .rename(
            columns={'amount': 'revenue',
                     '_user_id': 'count_unique_users_with_payment'})

    # 2 метрика - средний доход от каждого клиента в день
    payments_metrics['ARPU'] = \
        round(payments_metrics['revenue'] /
              payments_metrics['count_unique_users_with_payment'], 2)
    payments_metrics.sort_values('date_of_day')

    # 3 метрика - количество созданных заявок
    # по уникальным пользователям в день
    events_df['happened_at'] = pd.to_datetime(events_df['happened_at'])
    events_df['date_of_day'] =
events_df.happened_at.dt.strftime('%d.%m.%Y')
    count_unique_users_with_new_orders = events_df\
        .query('event_id == "3"')\
        .groupby('date_of_day', as_index=False)\
        .agg({'_user_id': pd.Series.nunique})\
        .rename(columns={'_user_id':
'count_unique_users_with_new_orders'})

    # Результирующий датасет
    metrics_df = payments_metrics.merge(
        count_unique_users_with_new_orders, on='date_of_day',
how='left')

    # Сортировка
    metrics_df = metrics_df.sort_values('date_of_day')

    # Исключим NaN из датафрейма на основании того,
    # что представленные данные не консистентны по времени выгрузки
    metrics_df = metrics_df[
        metrics_df['count_unique_users_with_new_orders'].notna()]

    # получение метрики revenue
    revenue: str = metrics_df['revenue'].iloc[-1]
    revenue: str = "{:,.2f}".format(revenue).replace(',', ' ')

    # получение метрики count_unique_users_with_payment
    count_unique_users_with_payment: str = \
        int(metrics_df['count_unique_users_with_payment'].iloc[-1])

    # получение метрики  ARPU
    arpu: str = metrics_df['ARPU'].iloc[-1]
    arpu: str = "{:,.2f}".format(arpu).replace(',', ' ')

    # получение метрики count_unique_users_with_new_orders
    count_unique_users_with_new_orders: str = \
        int(metrics_df['count_unique_users_with_new_orders'].iloc[-1])

    last_value_date_of_day: str = \
        metrics_df['date_of_day'].iloc[-1]

    result: str = f"""
    Показатели за *{last_value_date_of_day}* составили:
```

```
    - Дневная выручка: *{revenue}* руб.
    - Средний доход от каждого клиента: *{arpu}*  руб.
      (количество клиентов, произведших оплату:
*{count_unique_users_with_payment}*)
    - Количество клиентов, создавших заявку(-и):
*{count_unique_users_with_new_orders}*
    """

    # Настройка графика revenue
    x = metrics_df['date_of_day']
    y = metrics_df['revenue']
    title = 'Выручка в млн. руб.'
    xlabel = 'Период'
    ylabel = 'Выручка в млн. руб.'
    fig_name = 'revenue'
    bar_color = 'darkviolet'
    get_plot(x, y, title, xlabel, ylabel, fig_name, bar_color)

    # Настройка графика ARPU
    x = metrics_df['date_of_day']
    y = metrics_df['ARPU']
    title = 'Средний доход от каждого клиента в день'
    xlabel = 'Период'
    ylabel = 'ARPU, руб.'
    fig_name = 'ARPU'
    bar_color = 'darkturquoise'
    get_plot(x, y, title, xlabel, ylabel, fig_name, bar_color)

    # Настройка графика возможности продаж
    x = metrics_df['date_of_day']
    y = metrics_df['count_unique_users_with_new_orders']
    title = 'Количество клиентов, создавших заявку(-и) за день'
    xlabel = 'Период'
    ylabel = 'Количество клиентов.'
    fig_name = 'unique_users'
    bar_color = 'dodgerblue'
    get_plot(x, y, title, xlabel, ylabel, fig_name, bar_color)

    return textwrap.dedent(result)


def main() -> None:
    updater = Updater(token=TELEGRAM_TOKEN)
    while True:
        engine = connect_database()
        if engine:
            # Получение данных
            events_df = select(events, engine)
            payments_df = select(payments, engine)

            # Обработка и получение результирующих метрик
            message: str = get_metrics(payments_df, events_df)

            # Получение идентификаторов подписчиков бота
            updater.dispatcher.add_handler(
                MessageHandler(Filters.text, bot_initialize))
            updater.start_polling()
```

```
            # Отправка текстовой части сообщение
            for chat_item in chat_list:
                try:
                    updater.bot.send_message(chat_id=chat_item,
                                             text=message,
parse_mode='Markdown')
                except TelegramError:
                    print("Произошла ошибка при отправке сообщения в
телеграм")

                # Отправка графиков
                try:
                    ARPU = open('ARPU.png', 'rb')
                    revenue = open('revenue.png', 'rb')
                    unique_users = open('unique_users.png', 'rb')

                    medias = [
                        telegram.InputMediaPhoto(
                            revenue, caption='Выручка в млн. руб.'),
                        telegram.InputMediaPhoto(
                            ARPU,
                            caption='Средний доход от каждого клиента в
день'),
                        telegram.InputMediaPhoto(
                            unique_users,
                            caption='''Количество клиентов,
                            создавших заявку(-и) за день''')
                    ]
                    try:
                        updater.bot.send_media_group(chat_id=chat_item,
                                                     media=medias)
                    except TelegramError:
                        print("Произошла ошибка при отправке сообщения в
телеграм")

                except IOError:
                    print("Файлы не доступны")

            time.sleep(int(RETRY_TIME))


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