empenoso
20 янв в 03:25

Python и нечеткое сопоставление: решение проблемы разнобоя в адресах

Простой 10 мин 4.6K

Python*, Open source*, Геоинформационные сервисы*

Кейс Сезон Open source

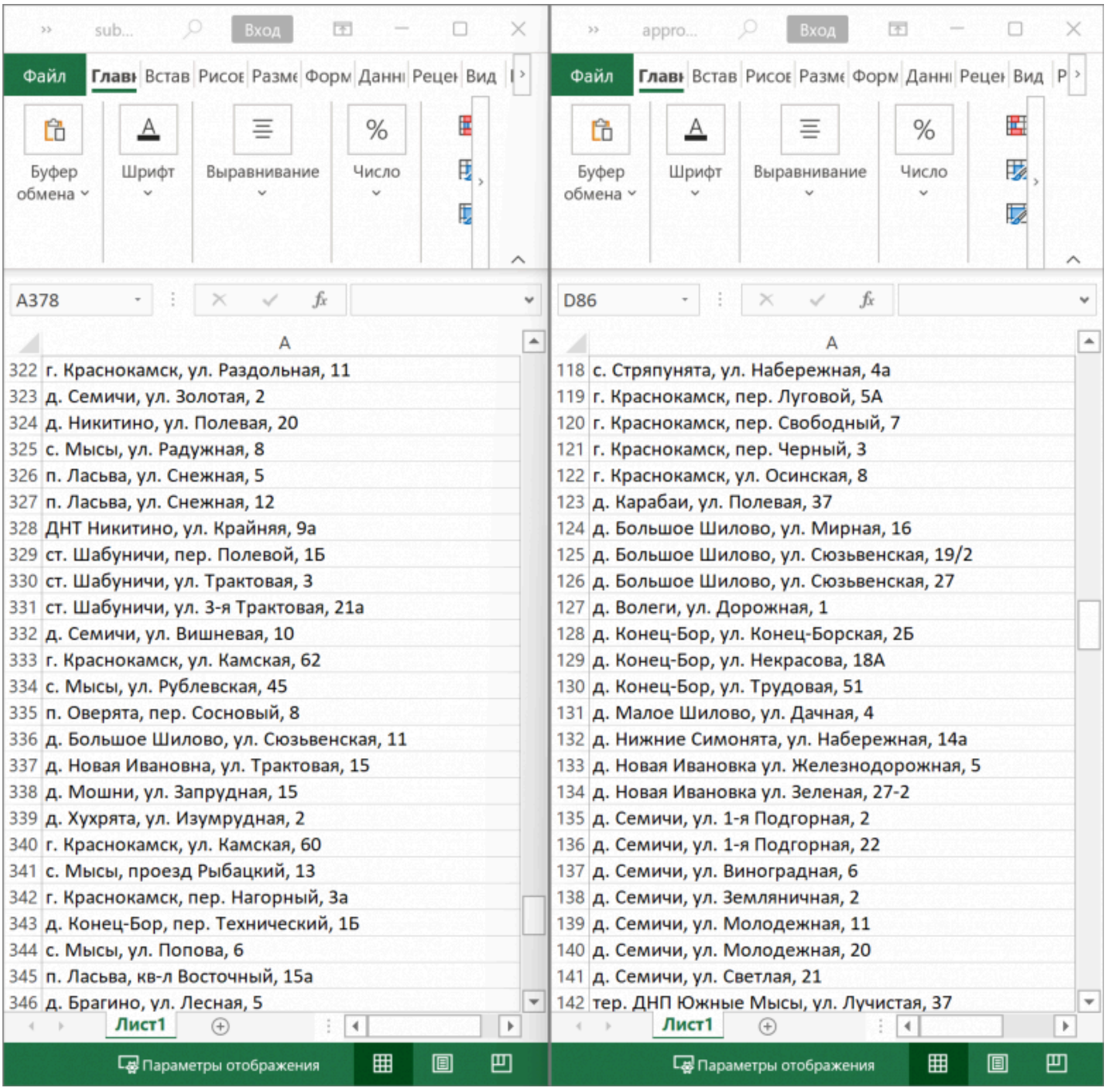
Иногда приходится заниматься сравнением больших списков адресов, в которых адреса записаны совершенно по разному без внятных идентификаторов вроде номера объекта - есть только адрес. Один и тот же адрес может фигурировать в различных списках следующим образом:

- "д. Малое Шилово, ул. Березовая, д. 7" и "Березовая 7_М Шилово".
- "п. Ласьва, ул. Весенняя, д. 5" и "Весенняя 5_Ласьва".
- "Луговой пер 5, Краснокамск г" и "г. Краснокамск, пер. Луговой, 5".
- "д. Новая Ивановка, ул. Солнечная, 18" и "д.Новая Ивановка, ул.Солнечная, 18".

Уже выделенные отдельно адреса могут выглядеть как на скриншоте Экселя ниже. А пример поставленной задачи может звучать так: «**В реестре поданных объектов отметить все согласованные объекты (из общего списка согласованных)**».

Если отбросить вариант ручного исполнения и обратиться к скриптам, то мне видится всего два решения:

1. Использовать алгоритмы нечёткого сопоставления.
2. Использовать геокодинг адресов.



Варианты решения этой задачи

Первый вариант – использование алгоритмов нечёткого сопоставления (fuzzy matching). Эти алгоритмы позволяют сравнивать строки, учитывая возможные опечатки, разные порядок слов и сокращения. В нашем случае, алгоритм сможет распознать "д. Малое Шилово, ул. Березовая, д. 7" и "Березовая 7_М Шилово" как варианты одного и того же адреса, несмотря на различия в формате и сокращения. Fuzzy matching оценивает

Это делает данный метод весьма эффективным для обработки больших списков адресов с вариативностью написания.

Не прямо в тему, но наглядно. Источник: pub.aimind.so

Второй подход – геокодинг. Этот метод преобразует текстовое описание адреса в географические координаты. Получив координаты для каждого адреса в обоих списках, можно сравнивать их близость и таким образом находить соответствия. Геокодинг полезен для проверки корректности адресов и выявления дубликатов, записанных по-разному. Однако, этот метод имеет существенные ограничения в контексте данной задачи. Во-первых, не все адреса могут быть найдены на картах. Если объект ещё строится, то адрес еще не внесен в картографические сервисы. Во-вторых, геокодинг может быть неточным, особенно в сельской местности. Таким образом, полагаться исключительно на геокодинг в данном случае рискованно.

Иллюстрация геокодинга. Источник: pubnub.com

Для нашей задачи, где требуется сравнить большие списки адресов с высокой вариативностью написания и наличием потенциально «несуществующих» адресов, алгоритмы нечёткого сопоставления представляются более подходящим решением. Они не требуют наличия адреса на карте и способны эффективно обрабатывать различные варианты написания одного и того же адреса. Гибкость настройки позволяет подобрать оптимальный баланс между точностью и полнотой поиска соответствий, минимизируя как ложноположительные, так и ложноотрицательные результаты. В то время как геокодинг может служить дополнительным инструментом для верификации результатов, основным методом сравнения адресов в данном случае следует выбрать fuzzy matching.

Подготовка данных

Прежде чем приступить к сравнению адресов, необходимо привести их к единому формату. Это значительно повысит точность алгоритмов нечёткого сопоставления. Различия в регистре, сокращениях, пунктуации и лишние пробелы могут помешать алгоритму правильно идентифицировать одинаковые адреса. Например, "д. Малое Шилово" и "малое шилово" будут рассматриваться как разные адреса, если не провести предварительную обработку.

fuzzywuzzy. pandas предоставляет удобные инструменты для работы с табличными

данными, openpyxl позволяет читать и записывать файлы Excel, а fuzzywuzzy реализует алгоритмы нечёткого сопоставления.

```
def clean_address(address):
    print(f"Очистка адреса: {address}")  # Вывод текущего адреса для очистки
    if pd.isnull(address):  # Проверяем, является ли адрес пустым значением
        return None

    # Приведение к нижнему регистру
    address = address.lower()

    # Список замен с сохранением структуры
    replacements = [
        (r"\bп/ст\b", ""),              # Убираем "п/ст"
        (r"\bднт\b", ""),               # Убираем "ДНТ"
        (r"\bснт\b", ""),               # Убираем "СНТ"
        (r"\bднп\b", ""),               # Убираем "ДНП"
        (r"\bкв-л\b", ""),              # Убираем "кв-л"
        (r"\bпроезд\b", ""),            # Убираем "проезд"
        (r"\bквартал\b", ""),           # Убираем "квартал"
        (r"\bд\.\s?", ""),              # Убираем "д." с пробелом
        (r"\bг\.\s?", ""),              # Убираем "г." с пробелом
        (r"\bпер\.\s?", ""),            # Убираем "пер." с пробелом
        (r"\bул\s?", ""),               # Убираем "ул" с пробелом
        (r"\bп\.\s?", ""),              # Убираем "п." с пробелом
        (r"\bс\.\s?", ""),              # Убираем "с." с пробелом
        (r"\bст\.\s?", ""),             # Убираем "ст." с пробелом
        (r"\bпр-д\b", "")               # Убираем "пр-д"
    ]

    # Применение замен
    for pattern, replacement in replacements:
        address = re.sub(pattern, replacement, address)

    # Удаление текста в скобках
    address = re.sub(r"\([^)]*\)", "", address)  # Убираем текст в скобках

    # Удаление лишних символов, но с сохранением структуры
    address = re.sub(r"[.,]", "", address)       # Убираем точки и запятые
    address = address.strip()                    # Убираем пробелы по краям
```

```
    print(f"Очищенный адрес: {address}")  # Вывод очищенного адреса
    return address
```

Для приведения адресов к единому формату используем функцию `clean_address`, представленную в коде выше. Она приводит адрес к нижнему регистру, удаляет сокращения (например, "д.", "ул.", "г."), текст в скобках, лишние пробелы и знаки препинания. Применение регулярных выражений обеспечивает гибкость и эффективность очистки. Функция также включает вывод исходного и очищенного адресов для контроля процесса обработки.

Перед началом работы необходимо установить упомянутые библиотеки. Это можно сделать с помощью pip:

```
pip install pandas openpyxl fuzzywuzzy
```

После установки библиотек и подготовки данных можно переходить к реализации алгоритма нечёткого сопоставления.

Основы работы с fuzzywuzzy

Библиотека fuzzywuzzy предоставляет несколько функций для сравнения строк, основанных на алгоритме Левенштейна. Этот алгоритм вычисляет минимальное количество операций (вставка, удаление, замена символов), необходимых для преобразования одной строки в другую. Чем меньше операций требуется, тем больше сходство между строками.

fuzzywuzzy предлагает три основные функции:

- **fuzz.ratio**: Сравнивает строки целиком, учитывая порядок слов. Например, fuzz.ratio("ул. Ленина 10", "Ленина ул 10") вернёт относительно низкий балл, несмотря на то, что слова одинаковые, но расположены в разном порядке.
- **fuzz.partial_ratio**: Ищет наиболее похожую подстроку. Полезно, когда одна строка является частью другой. Например, fuzz.partial_ratio("ул. Ленина 10", "г. Москва, ул. Ленина 10, кв 5") вернёт высокий балл, так как первая строка полностью содержится во

- **fuzz.token_sort_ratio**: Сначала сортирует слова в строках по алфавиту, а затем сравнивает их с помощью fuzz.ratio. Это позволяет игнорировать порядок слов. В нашем примере fuzz.token_sort_ratio("ул. Ленина 10", "Ленина ул 10") выдаст высокий балл, поскольку после сортировки строки станут идентичными.

```
# Функция для поиска совпадений с помощью fuzzy matching
def match_address(row, approved_addresses):
    cleaned_address = row["cleaned_address"]
    if not cleaned_address:  # Проверка, если адрес пустой (None или пустая строка)
        print("Пропущен пустой адрес")
        return None

    # Извлекаем цифры из текущего адреса
    current_digits = set(re.findall(r'\d+', cleaned_address))
    if not current_digits:
        print(f"Адрес без цифр пропущен: {cleaned_address}")
        return None

    # Отфильтровываем список одобренных адресов, оставляя только те, где есть совпадающ
    filtered_addresses = [
        addr for addr in approved_addresses
        if current_digits & set(re.findall(r'\d+', addr))
    ]

    if not filtered_addresses:
        print(f"Совпадений по цифрам не найдено для адреса: {cleaned_address}")
        return None

    print(f"Поиск совпадения для адреса: {cleaned_address}")  # Лог текущего адреса
    result = process.extractOne(cleaned_address, filtered_addresses, scorer=fuzz.token_

    if result:  # Если совпадение найдено
        match, score = result
        print(f"Найдено совпадение: {match} с оценкой {score}")  # Вывод найденного сов
        return match if score > 70 else None  # Возвращаем совпадение только при достат
    else:
        print("Совпадений не найдено")
        return None
```

Использую fuzz.token_sort_ratio в сочетании с предварительной фильтрацией по совпадающим цифрам в адресах. Это позволяет существенно ускорить процесс и повысить точность сопоставления, так как сравниваются только те адреса, номера которых потенциально могут совпадать.

Порог сходства установлен на 70, что означает, что совпадение считается найденным, только если оценка fuzz.token_sort_ratio превышает это значение. Это позволяет отсеять ложные совпадения.

Скрипт для сопоставления списков разных адресов

Скрипт вначале загружает данные из файлов Excel с помощью библиотеки pandas, после загрузки скрипт очищает адреса в обоих списках, используя функцию clean_address, приводя их к единому формату.

Затем начинается процесс сопоставления. Для каждого адреса из реестра поданных объектов скрипт ищет соответствие в реестре согласованных объектов с помощью библиотеки fuzzywuzzy. Функция process.extractOne, используемая в коде, позволяет эффективно находить совпаденич в большом списке, применяя алгоритм token_sort_ratio. Предварительная фильтрация по совпадающим цифрам в адресах значительно ускоряет обработку больших списков.

Результаты сопоставления, включая найденный адрес и отметку о согласованности "➕" или нет "❌", добавляются в исходный реестр поданных объектов. Окончательный результат сохраняется в новый файл Excel.

Полный код:

```
# pip install pandas openpyxl fuzzywuzzy

# Подробнее: https://habr.com/ru/articles/873242/

"""
Иногда приходится заниматься сравнением больших списков адресов, в которых адреса запис

 "д. Малое Шилово, ул. Березовая, д. 7" и "Березовая 7 М Шилово".
```

```
Уже выделенные отдельно адреса могут выглядеть как на скриншоте Экселя. А пример постав

Если отбросить вариант ручного исполнения и обратиться к скриптам, то мне видится всего

✅ Использовать алгоритмы нечёткого сопоставления.
✅ Использовать геокодинг адресов.
"""

import sys
sys.stdout.reconfigure(encoding='utf-8')

import re
import pandas as pd
from fuzzywuzzy import fuzz, process

def clean_address(address):
    print(f"Очистка адреса: {address}")  # Вывод текущего адреса для очистки
    if pd.isnull(address):  # Проверяем, является ли адрес пустым значением
        return None

    # Приведение к нижнему регистру
    address = address.lower()

    # Список замен с сохранением структуры
    replacements = [
        (r"\bп/ст\b", ""),              # Убираем "п/ст"
        (r"\bднт\b", ""),               # Убираем "ДНТ"
        (r"\bснт\b", ""),               # Убираем "СНТ"
        (r"\bднп\b", ""),               # Убираем "ДНП"
        (r"\bкв-л\b", ""),              # Убираем "кв-л"
        (r"\bпроезд\b", ""),            # Убираем "проезд"
        (r"\bквартал\b", ""),           # Убираем "квартал"
        (r"\bд\.\s?", ""),              # Убираем "д." с пробелом
        (r"\bг\.\s?", ""),              # Убираем "г." с пробелом
        (r"\bпер\.\s?", ""),            # Убираем "пер." с пробелом
        (r"\bул\s?", ""),               # Убираем "ул" с пробелом
        (r"\bп\.\s?", ""),              # Убираем "п." с пробелом
        (r"\bс\.\s?", ""),              # Убираем "с." с пробелом
        (r"\bст\.\s?", ""),             # Убираем "ст." с пробелом
```

```
    # Применение замен
    for pattern, replacement in replacements:
        address = re.sub(pattern, replacement, address)

    # Удаление текста в скобках
    address = re.sub(r"\([^)]*\)", "", address)  # Убираем текст в скобках

    # Удаление лишних символов, но с сохранением структуры
    address = re.sub(r"[.,]", "", address)       # Убираем точки и запятые
    address = re.sub(r"\s{2,}", " ", address)    # Убираем множественные пробелы
    address = re.sub(r"[\"]", "", address)       # Убираем кавычки
    address = address.strip()                    # Убираем пробелы по краям

    print(f"Очищенный адрес: {address}")  # Вывод очищенного адреса
    return address

# Функция для поиска совпадений с помощью fuzzy matching
def match_address(row, approved_addresses):
    cleaned_address = row["cleaned_address"]
    if not cleaned_address:  # Проверка, если адрес пустой (None или пустая строка)
        print("Пропущен пустой адрес")
        return None

    # Извлекаем цифры из текущего адреса
    current_digits = set(re.findall(r'\d+', cleaned_address))
    if not current_digits:
        print(f"Адрес без цифр пропущен: {cleaned_address}")
        return None

    # Отфильтровываем список одобренных адресов, оставляя только те, где есть совпадающ
    filtered_addresses = [
        addr for addr in approved_addresses
        if current_digits & set(re.findall(r'\d+', addr))
    ]

    if not filtered_addresses:
        print(f"Совпадений по цифрам не найдено для адреса: {cleaned_address}")
        return None

    print(f"Поиск совпадения для адреса: {cleaned_address}")  # Лог текущего адреса
```

```
        match, score = result
        print(f"Найдено совпадение: {match} с оценкой {score}")  # Вывод найденного сов
        return match if score > 70 else None  # Возвращаем совпадение только при достат
    else:
        print("Совпадений не найдено")
        return None

# Загружаем данные из Excel-файлов
print("Загрузка данных...")
submitted_df = pd.read_excel("submitted.xlsx")  # Реестр поданных объектов
approved_df = pd.read_excel("approved.xlsx")  # Реестр согласованных объектов

# Очистка адресов в обоих реестрах
print("Очистка адресов в таблицах...")
submitted_df["cleaned_address"] = submitted_df["address"].apply(clean_address)
approved_df["cleaned_address"] = approved_df["address"].apply(clean_address)

# Формируем список очищенных адресов из реестра согласованных объектов
approved_addresses = approved_df["cleaned_address"].dropna().tolist()

# Ищем совпадения и добавляем их в реестр поданных объектов
print("Сопоставление адресов...")
submitted_df["matched_address"] = submitted_df.apply(
    match_address, approved_addresses=approved_addresses, axis=1
)

# Добавляем отметку о согласованности
print("Добавление отметки о согласованности...")
# Проверяем наличие совпадения и добавляем соответствующий символ
submitted_df["is_approved"] = submitted_df["matched_address"].notnull().apply(
    lambda x: "➕" if x else "❌"
)

# Сохраняем результат в новый Excel-файл
print("Сохранение результатов...")
submitted_df.to_excel("submitted_with_matches_v2.xlsx", index=False)

print("Готово! Результаты сохранены в 'submitted_with_matches_v2.xlsx'.")
```

Заключение

Автоматизация процесса сопоставления адресов с помощью Python позволяет значительно сэкономить время и исключить ошибки, связанные с человеческим фактором. Вместо утомительной ручной проверки скрипт быстро и точно обрабатывает большие объемы данных. Более того, представленный скрипт легко адаптируется под похожие задачи, требующие сравнения текстовых строк, например, сопоставление наименований товаров или данных клиентов.

Для повышения точности сопоставления можно рассмотреть комбинирование fuzzy matching с геокодингом. Если адрес можно успешно геокодировать, то координаты служат дополнительным критерием для подтверждения совпадения.

Буду рад обсудить возможные улучшения и ответы на ваши вопросы в комментариях.

Автор: Михаил Шардин
🔗 Моя онлайн-визитка
📢 Telegram «Умный Дом Инвестора»

20 января 2025 г.

Теги: алгоритм, Алгоритм нечёткого сопоставления, карта, fuzzywuzzy, сезон open source

Хабы: Python, Open source, Геоинформационные сервисы

189 Карма 17.5 Рейтинг

Михаил Шардин @empenoso

Автоматизация / Данные / Финансы / Умные дома

Подписаться

Хабр Карьера Сайт Сайт Github

Комментарии 31

Сезон Open source

Поделись опытом и выиграй приз

ЛУЧШИЕ ЗА СУТКИ ПОХОЖИЕ

Exosphere
15 часов назад

Ещё 10 ошибок авторов Хабра

11 мин 3.2K

+93 35 64

vital_pavlenko
22 часа назад

Больше нет входа в IT. Только выход

2 мин 77K

+49 78 324

duran-duran
21 час назад

Трамплин в интернет: как мы ускорили запуск Яндекс Браузера

6 мин 3.2K

+42 9 29

Nickmob
16 часов назад

Введение в Angie: краткая история и отличия от Nginx

Простой 7 мин 2.8K

Ретроспектива

+37 29 8

Myskat_90
21 час назад

Сложный 14 мин 1.8K

Туториал

+33 46 0

slava_rumin
15 часов назад

Мое производство электрощитов приносит 40 млн в год. Спасибо нейросетям и СССР за конструкторскую школу

Простой 14 мин 21K

Интервью

+28 46 53

AlexeyNadezhin
23 часа назад

Важное обновление BatteryTest 2

Простой 3 мин 2.5K

+27 37 9

ru_vds
16 часов назад

Как создавались вокальные эффекты Daft Punk

Средний 13 мин 836

Обзор Перевод

+23 5 1

GordienkoAnd
19 часов назад

Как настраивать сети: готовые решения Selectel для максимальной отказоустойчивости

+21 28 0

alizar
20 часов назад

Почему из технологий делают культы

Простой 8 мин 1.3K

Обзор

+21 7 6

Кто вы, мистер багхантер? Что показал опрос на Хабре

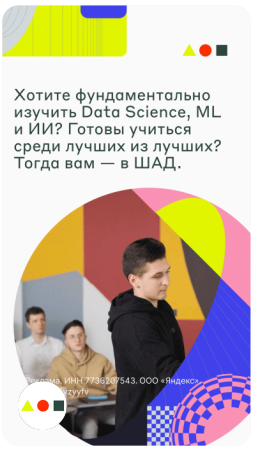
Турбо

Показать еще

ИСТОРИИ

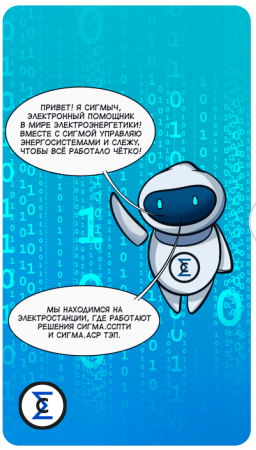
Старт гонки разработчиков

Торопись в сезон Open source



Открыт приём в Школу анализа данных

С Днём радио!

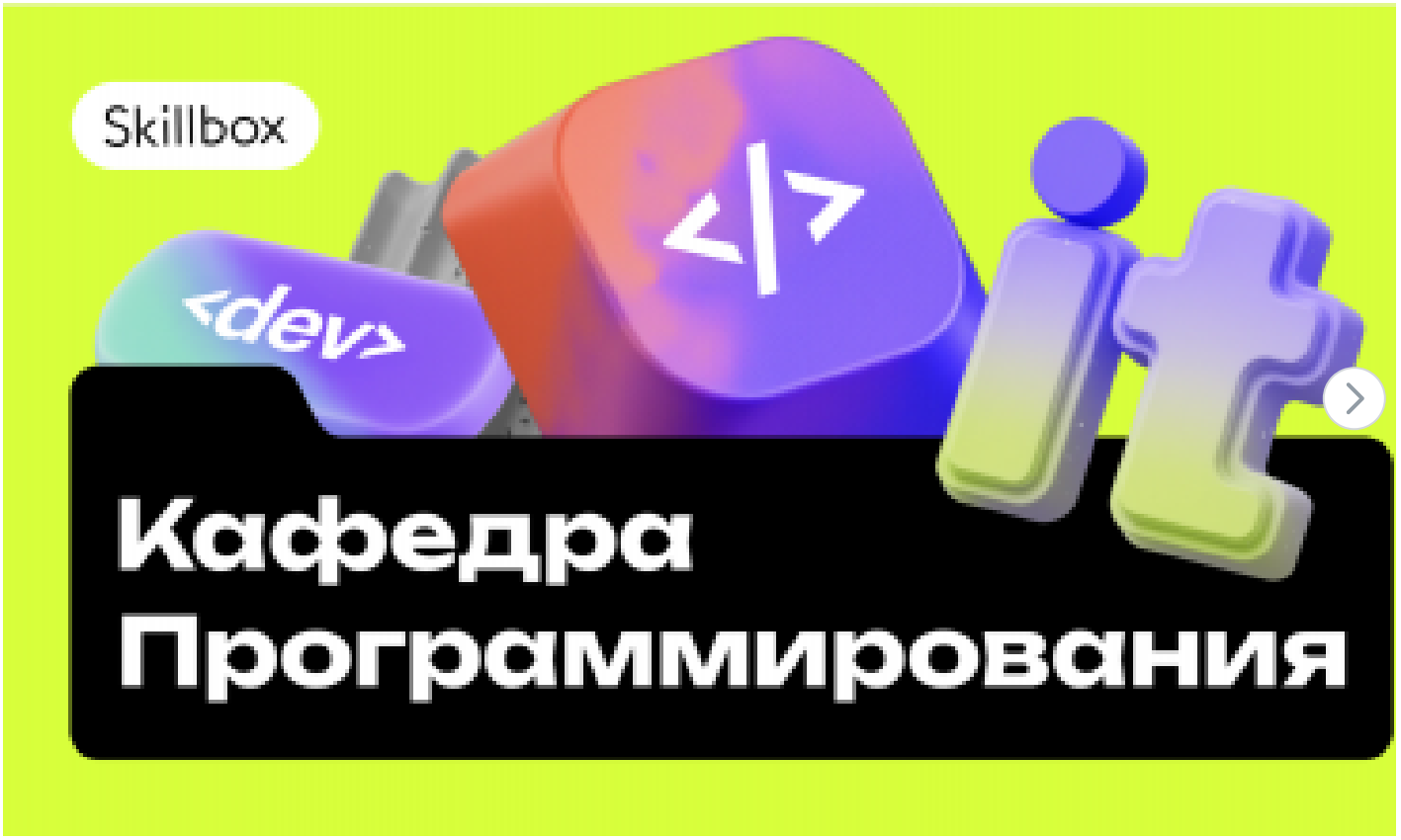


HELLO, WORLD — теперь на 110 киловольтах

Буду лучи

КУРСЫ

Сезон Open source
Поделись опытом и выиграй приз



17 апреля – 29 мая

Серия бесплатных офлайн-конференций «Кафедра Программировани от Skillbox

Москва

Разработка

Больше событий в календаре

Хабр

Сезон Open source

Поделись опытом и выиграй приз

Техническая поддержка