Отчет по анализу репозиториев GitHub

1. Скрипт получения трендовых репозиториев (`parsing_script.py`)

Принцип работы:
Скрипт комбинирует веб-скрейпинг (Web Scraping) и работу с официальным GitHub API. Поскольку у GitHub нет публичного API для получения списка "Trending", скрипт парсит HTML-страницу `github.com/trending/{language}` .

Алгоритм:

1. **Сбор кандидатов:** Библиотека `BeautifulSoup` извлекает список репозиториев со страницы трендов за период `monthly` . Это позволяет находить устоявшиеся популярные проекты, а не однодневные хайпы.
2. **Валидация через API:** Для каждого кандидата происходит проверка через GitHub API с использованием токена (для обхода лимитов).

Фильтры и их назначение:

1. **Доминация языка (`is_language_dominant`):**
   - *Логика:* Проверяется, что выбранный язык занимает более 50-60% кодовой базы (настраивается в `LANGUAGE_SETTINGS` , например, для C++ порог 40%).
   - *Зачем:* Отсеивает "Awesome-листы" (коллекции ссылок), книги, переводы документации и Jupyter Notebooks, где основной контент — не код, а текст или разметка.
2. **Количество участников (`has_enough_contributors` , min 3):**
   - *Зачем:* Исключает личные проекты-одиночки. Цель анализа — изучить командную разработку.
3. **Количество коммитов (`has_enough_commits`):**
   - *Логика:* Минимум 200–1500 коммитов (зависит от языка).
   - *Зачем:* Гарантирует наличие достаточной истории для статистического анализа. Новые репозитории с 10 коммитами не дадут репрезентативных данных.

---

2. Скрипт сбора данных о коммитах (`dataset_collection.cpp`)

Принцип работы:
Это высокопроизводительный модуль на C++, использующий низкоуровневую библиотеку

`libgit2`. Он клонирует репозитории локально и напрямую читает Git-историю (без использования медленного HTTP API для каждого файла).

Механики и Фильтры:

1. **Фильтрация файлов (`is_valid_file`):**
   - **По расширению:** Анализируются только файлы, относящиеся к целевому языку (например, для Python учитываются `.py`, но игнорируются `.md`, `.json`, `.yml`).
   - **Черный список путей:** Игнорируются папки `vendor/`, `node_modules/`, `docs/`, `tests/`.
   - *Цель:* Оценить метрики именно *продуктового кода*, исключив влияние зависимостей, автогенерации и документации.
2. **Игнорирование Merge-коммитов:**
   - Код пропускает коммиты, у которых более 1 родителя (`git_commit_parentcount > 1`).
   - *Цель:* Мердж-коммиты часто содержат "чужие" изменения и могут исказить статистику огромным количеством строк, которые автор мерджа не писал.
3. **Таймзоны:**
   - Время коммита корректируется на смещение часового пояса автора (`author->when.offset`), чтобы получить реальное "локальное" время работы (утро/день/вечер), а не UTC.
4. **Анализ Diff (Tree-to-Tree):**
   - Скрипт сравнивает состояние дерева файлов текущего коммита с родительским, вычисляя точное количество добавленных и удаленных строк внутри отфильтрованных файлов.

---

3. Скрипт визуализации (`metrics.py`)

Принцип работы:
Python-скрипт с использованием `pandas` и `seaborn` агрегирует сырые JSON-данные, полученные от C++ сборщика. Он строит индивидуальные графики для каждого репозитория и сводные графики для каждого языка.

Рассчитываемые метрики (для каждого репозитория):

1. **Commit Size Distribution (Распределение размера коммита):**
   - *Метрика:* Гистограмма количества измененных строк (Lines Changed = Added + Deleted) во всех коммитах. Также считается среднее арифметическое.

- *Нюанс:* Показывает "вес" типичного изменения. Если график смещен влево — коммиты маленькие и частые. Если есть пики справа — в проекте приняты большие "выкатки".

2. **Activity Heatmap (Карта активности):**
   - *Метрика:* Матрица, где по оси X — часы (0-23), по оси Y — дни недели. Цвет ячейки показывает количество коммитов в это время.
   - *Нюанс:* Позволяет увидеть рабочий график команды. Например, активность в выходные или по ночам может указывать на энтузиастов или переработки.
3. **Top Authors (Топ авторов):**
   - *Метрика:* Топ-10 авторов по количеству совершенных коммитов.
   - *Нюанс:* Показывает "Bus Factor" — зависимость проекта от ключевых разработчиков. Если один человек сделал 80% коммитов, проект держится на нем.
4. **Churn Stats (Коэффициент переписывания):**
   - *Метрика:* Среднее количество добавленных строк vs среднее количество удаленных строк. Считается отношение `Avg Deleted / Avg Added` (Churn Ratio).
   - *Нюанс:*
     - **Ratio < 1:** Код в основном пишется "в плюс" (накопление функционала).
     - **Ratio > 1:** Код чаще удаляют или переписывают, чем пишут новый (рефакторинг или удаление устаревшего).
5. **Changed Files per Commit (Файлов в коммите):**
   - *Метрика:* Гистограмма количества затронутых файлов в одном коммите.
   - *Нюанс:* Характеризует связность кода. Если один коммит меняет в среднем 1 файл — код модульный. Если 10+ файлов — изменение одной части системы требует правок во многих других местах.
6. **Top Modified Files (Самые изменяемые файлы):**
   - *Метрика:* Топ-5 файлов, которые фигурировали в коммитах чаще всего.
   - *Нюанс:* Выявляет "горячие точки" проекта. Файл, который меняют каждый день, часто является кандидатом на рефакторинг, так как в нем сосредоточена слишком большая логика.

Агрегация:
Скрипт также создает сводный DataFrame (`global_stats`), усредняя показатели (Average Commit Size, etc.) по всем репозиториям для конкретного языка, чтобы сравнить языки между собой.

---

4. Анализ итоговых метрик по языкам

1. Средний размер коммита (Average Commit Size)

- Наибольший средний размер коммита зафиксирован у **Python** (~390 строк), **Java** (~360 строк) и **C** (~350 строк).
- Наименьший средний размер коммита у **Haskell** (~75 строк).
- **Kotlin** и **Javascript** имеют показатели в диапазоне 140–160 строк.
- **C++**, **Rust** и **Go** находятся в диапазоне 190–260 строк.

2. Количество измененных файлов (Average Files Changed)

- Лидером по количеству затрагиваемых файлов является **Java** (более 7 файлов за коммит).
- На втором месте **Kotlin** (~4.5 файла) и **Rust** (~4.1 файла).
- Минимальное количество файлов меняется в **Haskell** (~2.3) и **Javascript** (~2.6).
- Остальные языки (**C++**, **Python**, **Go**, **C**, **Swift**) показывают результат в диапазоне 3.3 – 4.0 файла.

3. Коэффициент удаления/добавления (Delete/Add Ratio)

- У **Rust** зафиксировано самое высокое значение коэффициента — около 6.0.
- Значения заметно выше 1.0 (удаляется больше, чем добавляется) наблюдаются у **Java**, **Haskell**, **C**, **Javascript** (около 2.0).
- Значение ниже 1.0 (добавляется больше, чем удаляется) зафиксировано у **Python** (~0.7) и **Kotlin** (~0.9).

4. Объем данных (Total Commits Analyzed)

- Наибольшее количество коммитов было проанализировано для **Java** (более 31 000) и **C** (около 28 000).
- Для **Haskell** проанализировано около 21 000 коммитов.
- Наименьшее количество коммитов в итоговой выборке у **Python** (менее 1 000).