

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 2

ГЛАВА 1: ОПИСАНИЕ ПРИЛОЖЕНИЯ И ТЕХНИЧЕСКОЙ ЗАДАЧИ..... 3

ГЛАВА 2: ФУНКЦИОНАЛЬНАЯ СПЕЦИФИКАЦИЯ..... 5

ГЛАВА 3: СТАТИЧЕСКИЙ АНАЛИЗ 9

1.1 Инструмент статического анализа..... 9

1.2 Процедура установки инструмента 9

1.3 Описание необходимых опций анализатора 11

1.4 Процедура запуска анализатора..... 12

1.5 Анализ ошибок 12

1.6 Выводы 14

ГЛАВА 4: МОДУЛЬНОЕ ТЕСТИРОВАНИЕ 16

ГЛАВА 5: ФОРМАЛИЗОВАННЫЙ ОБЗОР КОДА..... 23

ГЛАВА 6: ДИНАМИЧЕСКИЙ АНАЛИЗ 27

ЗАКЛЮЧЕНИЕ 32

ПРИЛОЖЕНИЕ 1. ИСХОДНЫЙ КОД 34

ВВЕДЕНИЕ

Краткое описание технической задачи. Укажите цели курсовой работы и перечислите методы проверок, выполненные для выбранного приложения.

ГЛАВА 1: ОПИСАНИЕ ПРИЛОЖЕНИЯ И ТЕХНИЧЕСКОЙ ЗАДАЧИ

В данной работе предлагается изучить задачу, решаемую с помощью нейронных сетей – анализ и обработку временных рядов данных от измерительных устройств, включая фильтрацию шумов, определение значимых событий (например, капель) и анализ временных интервалов между событиями в разных каналах.

Программа была реализована в виде набора модулей на языке Python. Мультитаргетные модели используются, когда необходимо предсказать несколько целевых переменных одновременно. В данном случае, модель обучается на временных разнице между различными парами каналов.

1. Из полученного датафрейма удаляются признаки, содержащие в себе записи по экспериментам.
2. Выделяются целевые колонки – расстояния между пиками.
3. Обозначаются цели таргета и исключаются столбцы, являющиеся лишними данными для обучения (к примеру сами цели таргета и id колонок), указывается пул таргета и обучающих данных.
4. Настраивается мультитаргетная модель CatBoostRegressor
5. Обучение и результат работы модели

ГЛАВА 2: ФУНКЦИОНАЛЬНАЯ СПЕЦИФИКАЦИЯ

Терминология (описываем специфические термины характерные для данной среды).

Нейронная сеть – это метод в искусственном интеллекте, который учит компьютеры обрабатывать данные таким же способом, как и человеческий мозг. Это тип процесса машинного обучения, называемый глубоким обучением,

который использует взаимосвязанные узлы или нейроны в слоистой структуре, напоминающей человеческий мозг.

Большие данные — это разнообразные данные, поступающие с более высокой скоростью, объем которых постоянно растет. Таким образом, три основных свойства больших данных — это разнообразие, высокая скорость поступления и большой объем.

Линейная регрессия — используемая в статистике регрессионная модель зависимости одной переменной y от другой или нескольких других переменных x с линейной функцией зависимости.

Машинное обучение — класс методов искусственного интеллекта, характерной чертой которых является не прямое решение задачи, а обучение за счёт применения решений множества сходных задач.

Полное описание функционала приложения (или отдельных компонент).

Сначала описываем поведение приложения в нулевой точке, а затем уже описываем реакцию приложения при подаче всего множества входных данных по пунктам соответственно

	Название функции	Описываем саму функцию	Описываем ожидаемый выход
1	apply_lowpass_filter	Применяет низкочастотный фильтр Баттерворта к входным данным. Функция предназначена для фильтрации шумов из сигнала по заданной частоте среза, порядку фильтра и частоте дискретизации.	Отфильтрованный сигнал

		Возвращает отфильтрованный сигнал.	
2	np_to_df	Преобразует массив NumPy в DataFrame pandas, объединяя его с существующим DataFrame, если он предоставлен. Эта функция может использоваться для объединения данных из разных источников в единый DataFrame для дальнейшего анализа.	Возвращение датафрейма в новом формате.
3	heigh_search	Определяет уровень амплитуды сигнала, ниже которого значения считаются слишком малыми для учета (например, слишком маленькие капли). Это делается путем вычисления квантиля амплитуды положительных значений в столбце 'Channel A'.	Функция возвращает значение высоты, которое может использоваться для фильтрации данных.
4	plotter_maker	Генерирует набор графиков для сравнения двух наборов данных	Функция не возвращает значение, но сохраняет

		(setup и setup2) по заданному имени столбца. Графики позволяют визуально сравнить результаты оптимизированных и неоптимизированных данных.	изображения графиков в файл.
5	dt_finder	Находит временные интервалы (dt) между пиками в разных каналах измерений. Функция работает с DataFrame, содержащим уникальные идентификаторы измерений (ID) и временные метки для каждого канала.	Результатом работы функции является новый DataFrame, содержащий вычисленные временные интервалы между каналами для каждого ID.
6	raindrops_and_peaks	Определяет пики в сигнале канала 'Channel A' и вырезает окна вокруг этих пиков для всех каналов. Окна могут быть отфильтрованы с использованием низкочастотного фильтра.	Возвращает массив окон вокруг пиков.

7	subpeaks	<p>Нормализация данных канала.</p> <p>Определение первичного пика с высотой 1.</p> <p>Поиск вторичных пиков в сигнале с определенными параметрами, такими как высота и prominence.</p> <p>Возвращение временных различий между первичным пиком и найденными вторичными пиками.</p>	<p>Возвращение временных различий между первичным пиком и найденными вторичными пиками.</p>
8	semi_max	<p>Вычисляется полуширина пика путем вызова другой функции semi_width.</p> <p>Если результат является кортежем, извлекается полуширина пика.</p> <p>Затем вычисляется и возвращается коэффициент, используя формулу: $(\text{максимальное значение канала} - \text{полусреднее максимальное значение канала}) / \text{полуширина пика}$.</p>	<p>Возвращается значение полуширины пика</p>

9	mean_median	Вычисляется среднее значение (mean) и медиана (median) для данных канала. Возвращается разница между средним значением и медианой.	Возвращается разница между средним значением и медианой.
10	q_10	Используя функцию pr.npquantile из библиотеки NumPy, вычисляется квантиль 10% для данных канала.	Результат, то есть значение квантиля 10%, возвращается из функции.

ГЛАВА 3: СТАТИЧЕСКИЙ АНАЛИЗ КОДА ПРИЛОЖЕНИЯ

3.1 Описание выбранного инструмента статического анализа.

Pylint - это мощный инструмент статического анализа кода на языке программирования Python. Его основная задача - обнаруживать потенциальные проблемы и ошибки в коде на ранних этапах разработки, что позволяет улучшить его качество, соблюдать стандарты кодирования и предотвращать ошибки выполнения программы. Вот более подробное описание его функционала:

Проверка стиля кодирования: Pylint анализирует стиль вашего кода в соответствии с рекомендациями PEP 8, официальным руководством по стилю написания кода на Python, а также другими стандартами. Он предупреждает о несоответствиях стандартам, таким как неправильное именование переменных, отступы, длина строки и т.д.

Анализ потенциальных ошибок: Pylint выявляет потенциальные ошибки в вашем коде, такие как использование неопределенных переменных, неправильное использование операторов, некорректное обращение к атрибутам объектов и т.д. Это помогает предотвратить ошибки выполнения программы.

Предупреждения о неэффективном использовании кода: Инструмент также предупреждает о возможных проблемах производительности или неэффективном использовании кода, таких как лишние импорты, циклы слишком большой глубины, ненужные операции и т.д.

Поддержка конфигурационных параметров: Pylint предлагает широкий спектр настроек, которые позволяют настраивать его поведение в соответствии с требованиями проекта или команды разработчиков. Это включает в себя возможность отключения или настройки конкретных проверок, изменения уровня строгости и т.д.

Генерация отчетов о качестве кода: Инструмент позволяет генерировать различные отчеты о качестве кода, которые содержат информацию о найденных проблемах, статистику по коду и другую полезную информацию для анализа и улучшения качества кодовой базы.

Выбор Pylint обосновывается его широкой популярностью в сообществе Python-разработчиков, активной поддержкой разработчиков и пользователями, а также его мощным и гибким функционалом, который помогает автоматизировать и упростить процесс анализа кода.

```
C:\Users\Professional>pip install pylint
Collecting pylint
  Downloading pylint-3.1.0-py3-none-any.whl.metadata (12 kB)
Requirement already satisfied: platformdirs>=2.2.0 in c:\users\professional\appdata\local\programs\python\
site-packages (from pylint) (3.10.0)
Collecting astroid<=3.2.0-dev0,>=3.1.0 (from pylint)
  Downloading astroid-3.1.0-py3-none-any.whl.metadata (4.5 kB)
Collecting isort!=5.13.0,<6,>=4.2.5 (from pylint)
  Downloading isort-5.13.2-py3-none-any.whl.metadata (12 kB)
Collecting mccabe<0.8,>=0.6 (from pylint)
  Downloading mccabe-0.7.0-py2.py3-none-any.whl.metadata (5.0 kB)
Collecting tomlkit>=0.10.1 (from pylint)
  Downloading tomlkit-0.12.4-py3-none-any.whl.metadata (2.7 kB)
Collecting dill>=0.3.6 (from pylint)
  Downloading dill-0.3.8-py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: colorama>=0.4.5 in c:\users\professional\appdata\local\programs\python\pyth
-packages (from pylint) (0.4.6)
Downloading pylint-3.1.0-py3-none-any.whl (515 kB)
----- 515.6/515.6 kB 363.6 kB/s eta 0:00:00
Downloading astroid-3.1.0-py3-none-any.whl (275 kB)
----- 275.6/275.6 kB 585.8 kB/s eta 0:00:00
Downloading dill-0.3.8-py3-none-any.whl (116 kB)
----- 116.3/116.3 kB 616.3 kB/s eta 0:00:00
Downloading isort-5.13.2-py3-none-any.whl (92 kB)
----- 92.3/92.3 kB 656.6 kB/s eta 0:00:00
Downloading mccabe-0.7.0-py2.py3-none-any.whl (7.3 kB)
Downloading tomlkit-0.12.4-py3-none-any.whl (27 kB)
```

Рисунок 1 – Инсталляция Pylint

3.2 Настройка и объяснение правил анализатора Pylint

Группа правил "Стиль кода":

C0103 - Несоответствие стандарту именования переменных:

Это правило проверяет, соответствует ли имя переменной или функции стандартам именования PEP 8. Например, оно предупреждает о использовании имени переменной, не начинающегося с буквы в нижнем регистре или содержащего нижние подчеркивания в начале или конце.

C0301 - Максимальная длина строки кода:

Данное правило проверяет, не превышает ли длина строки кода максимальное допустимое значение (обычно 79 символов согласно PEP 8). Следует разбивать длинные строки на более короткие для повышения читаемости кода.

Группа правил "Потенциальные ошибки":

W0612 - Неиспользуемый импорт:

Это правило предупреждает о наличии импортированных модулей, которые не используются в коде. Неиспользуемые импорты могут увеличить размер исполняемого файла и усложнить понимание структуры проекта.

W0101 - Использование raise без указания исключения:

Данное правило проверяет, используется ли оператор raise без указания конкретного исключения. Это может привести к ошибкам выполнения программы из-за неопределенности исключительной ситуации.

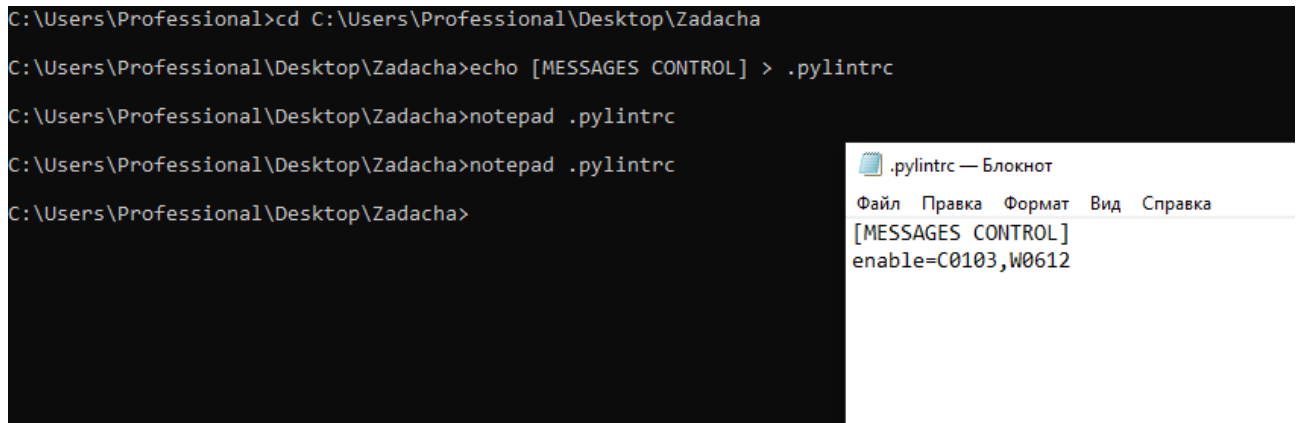


Рисунок 2 – Настройка анализатора

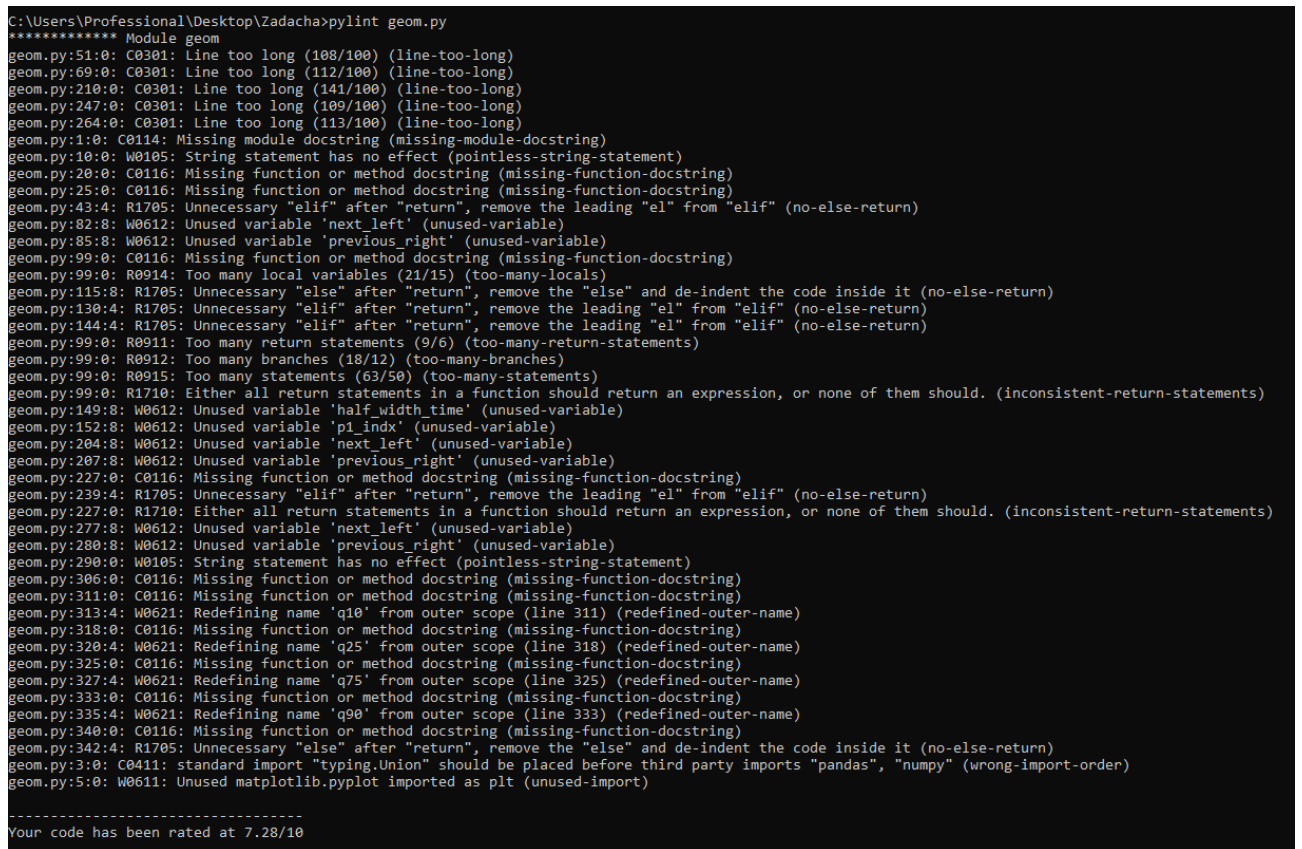


Рисунок 3 – Результат работы статического анализатора

В результате анализа с помощью pylint было обнаружено несколько типов ошибок и предупреждений в модуле geom.py. Вот краткое описание этих ошибок:

Line too long (C0301):

Данный вид ошибки возникает, когда длина строки превышает заданный предел символов. Это может затруднять чтение кода и его поддержку, особенно если строка содержит длинные выражения или комментарии. Рекомендуется разбивать длинные строки на несколько более коротких или использовать продолжение строки.

Missing module docstring (C0114):

Это предупреждение указывает на отсутствие документации модуля. Документирование модуля помогает другим разработчикам быстрее понять его назначение, интерфейс и использование.

Missing function or method docstring (C0116):

Это предупреждение указывает на отсутствие документации для функций или методов. Документирование функций и методов помогает понять их назначение, ожидаемые параметры и возвращаемые значения.

Unnecessary "elif" after "return" (R1705):

Это предупреждение указывает на лишний оператор elif после оператора return в условных конструкциях. Если оператор return встречается внутри условного блока, то код после него может быть недостижимым и его можно удалить.

Unused variable (W0612):

Это предупреждение указывает на неиспользуемые переменные в коде.

Неиспользуемые переменные могут быть признаком лишнего или ненужного кода, который следует удалить для повышения читаемости и поддерживаемости.

Redefining name from outer scope (W0621):

Это предупреждение указывает на переопределение имени переменной или функции из внешнего контекста. Переопределение имени может привести к путанице и ошибкам в коде, поэтому лучше избегать такой практики.

Too many local variables (R0914):

Это предупреждение указывает на слишком большое количество локальных переменных в функции. Большое количество переменных может быть признаком сложности функции и затруднять её понимание и тестирование.

Too many return statements (R0911):

Это предупреждение указывает на слишком большое количество операторов return в функции. Большое количество операторов return может сделать код менее читаемым и усложнить его понимание.

```
geom.py:50:0: C0301: Line too long (137/100) (line-too-long)
geom.py:80:0: C0301: Line too long (107/100) (line-too-long)
geom.py:97:0: C0301: Line too long (113/100) (line-too-long)
geom.py:115:0: C0301: Line too long (141/100) (line-too-long)
geom.py:153:0: C0305: Trailing newlines (trailing-newlines)
geom.py:1:0: C0114: Missing module docstring (missing-module-docstring)
geom.py:7:0: C0116: Missing function or method docstring (missing-function-docstring)
geom.py:10:0: C0116: Missing function or method docstring (missing-function-docstring)
geom.py:21:4: R1705: Unnecessary "elif" after "return", remove the leading "el" from "elif" (no-else-return)
geom.py:41:0: C0116: Missing function or method docstring (missing-function-docstring)
geom.py:41:0: R0914: Too many local variables (18/15) (too-many-locals)
geom.py:53:8: R1705: Unnecessary "else" after "return", remove the "else" and de-indent the code inside it (no-else-return)
geom.py:65:4: R1705: Unnecessary "elif" after "return", remove the leading "el" from "elif" (no-else-return)
geom.py:78:4: R1705: Unnecessary "elif" after "return", remove the leading "el" from "elif" (no-else-return)
geom.py:81:0: R0911: Too many return statements (10/6) (too-many-return-statements)
geom.py:41:0: R0912: Too many branches (19/12) (too-many-branches)
geom.py:41:0: R0915: Too many statements (60/50) (too-many-statements)
geom.py:80:8: W0612: Unused variable 'half_width_time' (unused-variable)
geom.py:134:0: C0116: Missing function or method docstring (missing-function-docstring)
geom.py:137:0: C0116: Missing function or method docstring (missing-function-docstring)
geom.py:138:4: W0621: Redefining name 'q10' from outer scope (line 137) (redefined-outer-name)
geom.py:141:0: C0116: Missing function or method docstring (missing-function-docstring)
geom.py:142:4: W0621: Redefining name 'q25' from outer scope (line 141) (redefined-outer-name)
geom.py:145:0: C0116: Missing function or method docstring (missing-function-docstring)
geom.py:146:4: W0621: Redefining name 'q75' from outer scope (line 145) (redefined-outer-name)
geom.py:149:0: C0116: Missing function or method docstring (missing-function-docstring)
geom.py:150:4: W0621: Redefining name 'q90' from outer scope (line 149) (redefined-outer-name)
geom.py:3:0: C0411: standard import "typing.Union" should be placed before third party imports "pandas", "numpy" (wrong-import-order)
geom.py:5:0: W0611: Unused matplotlib.pyplot imported as plt (unused-import)

-----
Your code has been rated at 7.24/10 (previous run: 7.32/10, -0.08)
```

Рисунок 4 – Результат статического анализа после исправления некоторых ошибок

ГЛАВА 4: МОДУЛЬНОЕ ТЕСТИРОВАНИЕ

4.1 Отчет о модульном тестировании.

Для модульного тестирования мною был выбран модуль **geometricals**, содержащий в себе функции для анализа временных рядов.

Этот модуль содержит в себе 350 строк кода.

Для написания и выполнения модульных тестов (unit tests) я использую библиотеку **unittest** в Python, которая является частью стандартной библиотеки Python. Версия **unittest** соответствует версии Python, которую использую. Например, для Python 3.8 будет использоваться **unittest** версии 3.8.

Основные возможности **unittest** включают:

Организация тестов: **unittest** предоставляет возможность организации тестов в виде классов, что упрощает группировку связанных тестов.

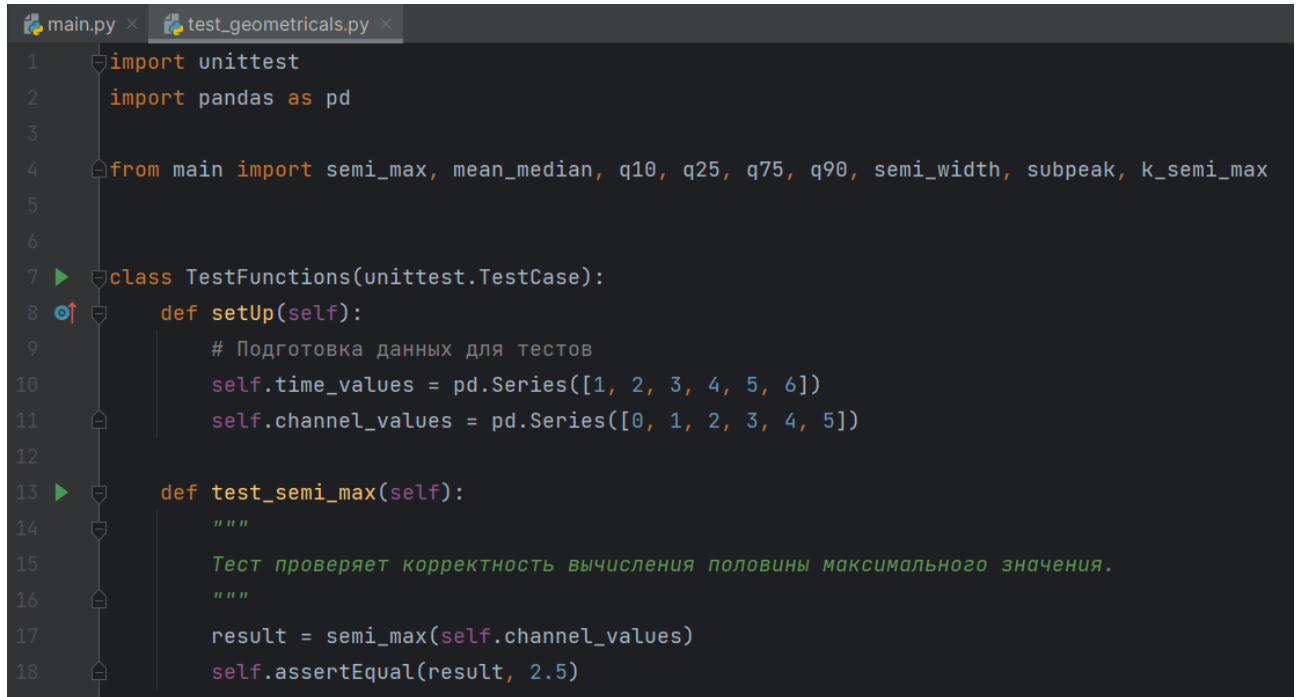
Accept: **unittest** предоставляет различные методы для проверки ожидаемых результатов ваших тестов. Это включает в себя методы `assertEqual`, `assertTrue`, `assertFalse` и многие другие.

Фикстуры: **unittest** поддерживает использование фикстур для предварительной подготовки данных перед выполнением тестов и очистки после их завершения. Это делает тесты более независимыми и предсказуемыми.

Тестовые сценарии: unittest позволяет определять тестовые сценарии, которые могут включать в себя несколько тестовых методов, что упрощает организацию сложных тестовых случаев.

Все тесты, написанный мной, были помещены в отдельный модуль

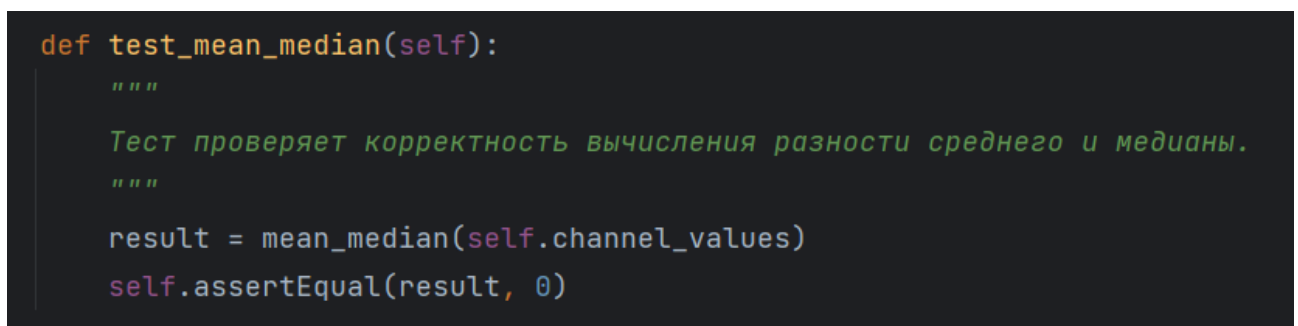
test_geometricals



```
1 import unittest
2 import pandas as pd
3
4 from main import semi_max, mean_median, q10, q25, q75, q90, semi_width, subpeak, k_semi_max
5
6
7 class TestFunctions(unittest.TestCase):
8     def setUp(self):
9         # Подготовка данных для тестов
10         self.time_values = pd.Series([1, 2, 3, 4, 5, 6])
11         self.channel_values = pd.Series([0, 1, 2, 3, 4, 5])
12
13     def test_semi_max(self):
14         """
15         Тест проверяет корректность вычисления половины максимального значения.
16         """
17         result = semi_max(self.channel_values)
18         self.assertEqual(result, 2.5)
```

Рисунок 5 – Создание класса для тестирования

Ниже будут приведены некоторые примеры тестов.



```
def test_mean_median(self):
    """
    Тест проверяет корректность вычисления разности среднего и медианы.
    """
    result = mean_median(self.channel_values)
    self.assertEqual(result, 0)
```

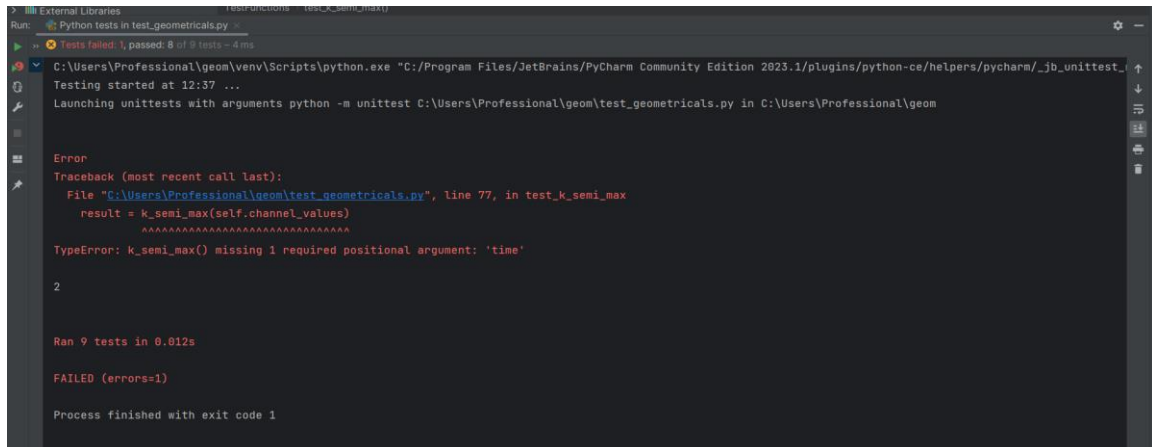
Рисунок 6 – test_mean_median

```
def test_subpeak(self):
    """
    Тест проверяет корректность вычисления сабпиков.
    """
    result = subpeak(self.channel_values, self.time_values)
    # Добавляем проверку ожидаемого результата
    expected_result = (0, 0, 0, 0, 0)
    self.assertEqual(result, expected_result)
```

Рисунок 7 – test_subpeak

Я выполнил мой программный проект вместе с тестами и получил следующие результаты:

1. Общее количество тестов – 9
2. Прошло успешно – 8
3. Провалено – 1



```
Run: Python tests in test_geometricals.py
Tests failed: 1, passed: 8 of 9 tests - 4 ms
C:\Users\Professional\geom\venv\Scripts\python.exe "C:/Program Files/JetBrains/PyCharm Community Edition 2023.1/plugins/python-ce/helpers/pycharm/_jb_unittest_
Testing started at 12:37 ...
Launching unittests with arguments python -m unittest C:\Users\Professional\geom\test_geometricals.py in C:\Users\Professional\geom

Error
Traceback (most recent call last):
  File "C:\Users\Professional\geom\test_geometricals.py", line 77, in test_k_semi_max
    result = k_semi_max(self.channel_values)
             ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
TypeError: k_semi_max() missing 1 required positional argument: 'time'

2

Ran 9 tests in 0.012s

FAILED (errors=1)

Process finished with exit code 1
```

Рисунок 8 – Результат тестирования

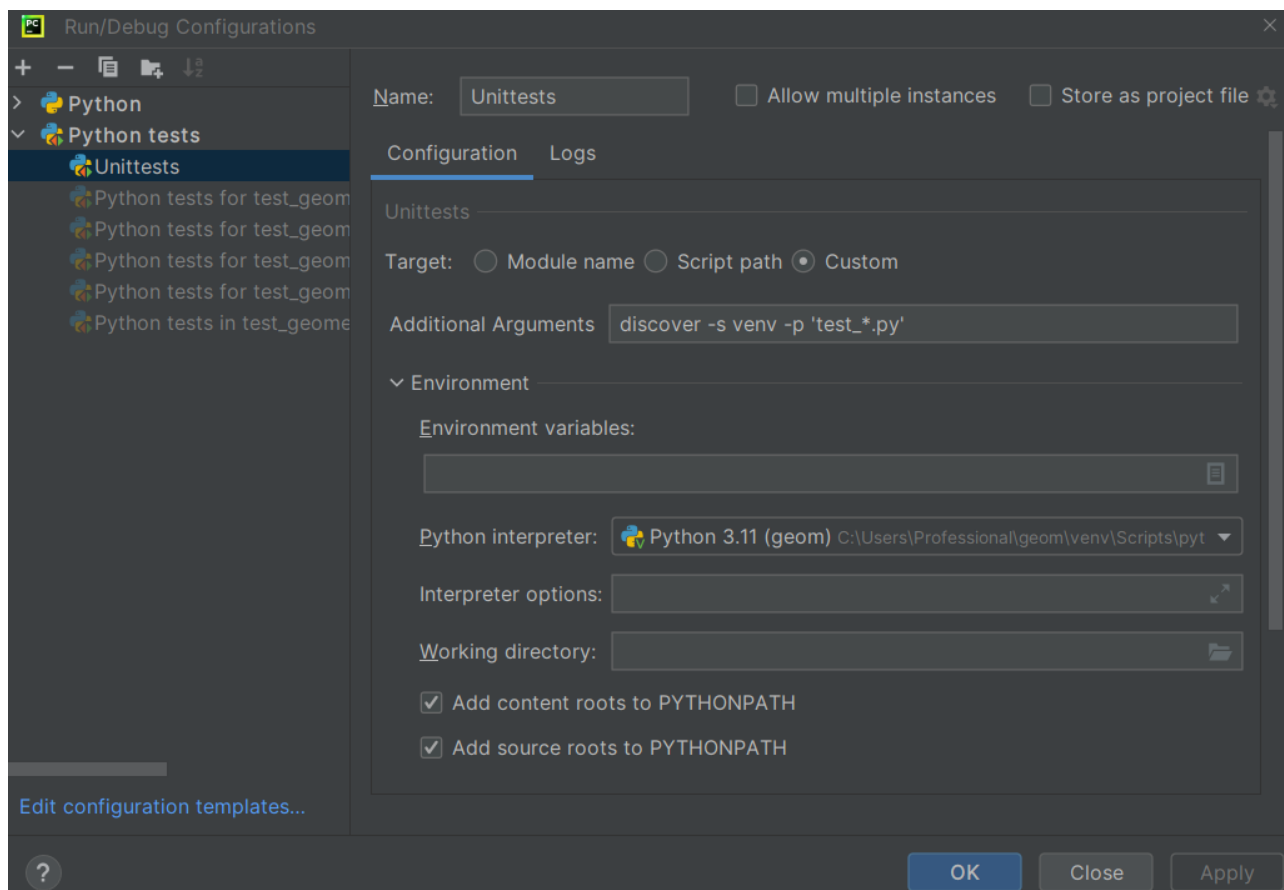


Рисунок 9 – Создание конфигурации для автоматического тестирования

Для получения информации о покрытии кода тестами был использован инструмент **coverage**

```
C:\Users\Professional>pip install coverage
Requirement already satisfied: coverage in c:\users\professional\appdata\local\programs\python\python311\lib\site-packages (7.4.4)
```

Рисунок 10 – Установка coverage

```
C:\Users\Professional\geom>coverage report
Name                               Stmts  Miss  Cover
-----
main.py                             162    115    29%
test_geometricals.py                 39      3    92%
TOTAL                               201    118    41%
```

Рисунок 11 – Отчет о покрытии кода тестами

Модульное тестирование оказалось эффективным методом проверки кода, позволяющим выявить и исправить ошибки на ранних стадиях разработки. Для повышения качества кода необходимо внедрять стратегию модульного тестирования, активно использовать хранение тестовых данных и регулярно проводить тестирование кода с помощью модульных тестов.

ГЛАВА 4: ФОРМАЛИЗОВАННЫЙ ОБЗОР КОДА

В данной главе будет произведен формальный обзор кода

https://github.com/Egowkins/test_raindrop/blob/TODO_branch_official/geometricals.py

Язык программирования: Python

Количество проверяемых строк исходного кода: 357

Количество участников: 3

4.1 Подготовительная фаза

Участники:

Автор: Алешковский Александр, 5140901/31502

Модератор: Росинский Александр, 5140901/31502

Рецензент: Подкина Анастасия, 5140901/31502

Материалы:

Данный код предназначен для вычисления различных геометрических характеристик спектограмм капель топлива.

На рисунке 12 приведены названия функций для формализованной проверки.

func semi_max
 func semi_width
 func subpeak
 func semi_width_subpekas
 func mean_median
 func q10
 func q25
 func q75
 func q90
 func k_semi_max

Рисунок 12 – Материалы, предоставляемые для проверки

```

99  def subpeak(channel, time, peak_return=False):
100
101      normalized_series = (channel - channel.min()) / (channel.max() - channel.min())
102      peak_indx = np.where(normalized_series == 1)[0]
103      # Берем первый пик с высотой 1
104      if peak_indx.size == 0:
105          return 0, 0, 0, 0, 0
106
107      peak_1 = channel.max()
108      #пик
109      #Пики по параметрам
110
111      peaks_2, _ = find_peaks(normalized_series, height=peak_1*0.3,
112                              prominence=(normalized_series.max()*0.2, normalized_series.max()*0.99))
113
  
```

Рисунок 13 – Фрагмент кода для формализованной проверки

Дата собрания: 25.04.2024

Количество участников: 3

Трудозатраты данной фазы: 1 чел-час.

4.2 Обзор кода

Дата собрания: 25.02.2024

Количество участников: 3

Трудозатраты данной фазы: 2 чел-час.

Цели обзора кода:

Ускорение работы некоторых функций

Увеличение отказоустойчивости кода

Соответствие кода стандартам PEP8

Журнал дефектов

Номер ошибки	Описание ошибки	Согласие	Статус ошибки
1	В функции semi_width возможно небольшое несоответствие в расчете. Может лучше убедиться, что right_index -1 и left_index+1 не превышают диапазон индексов time	Нет	Не признано ошибкой
2	В функции subpeak в случае, если пиков не найдено, лучше вернуть None вместо пустого кортежа	Да	Исправлено
3	В функции subpeak используется if peak is -1 , что является неправильным	Да	Исправлено

	способом сравнения. Лучше использовать if peak == -1		
4	В функции semi_width_subpeaks необходимо добавить проверку на случай, если крайние индексы пусты	Да	Исправлено
5	В функции k_semi_max возможно деление на ноль	Да	Исправлено

4.3 Исправление

Даты, в которые выполнялись исправления: 25-26.04.2024

Трудозатраты фазы: 2 чел-час

Дата собрания фазы верификации и завершения: 26.04.2024

Журнал дефектов

Номер ошибк и	Описание ошибки	Соглас ие	Статус ошибки
1	В функции semi_width	Нет	Не признано ошибкой

	<p>ВОЗМОЖНО небольшое несоответствие в расчете. Может лучше убедиться, что right_index -1 и left_index+1 не превышают диапазон индексов time</p>		
2	<p>В функции subpeak в случае, если пиков не найден, лучше вернуть None вместо пустого кортежа</p>	Да	<pre>def subpeak(channel, time, peak_return=False): normalized_series = (channel - channel.min()) / (channel.max() - channel.min()) peak_index = np.where(normalized_series == 1)[0] # Берем первый пик с высотой 1 if len(peaks_2) == 0: return None, None, None, None, None</pre>
3	<p>В функции subpeak используется if peak is -1, что является неправильным способом сравнения. Лучше</p>	Да	<pre>#лучший вариант - peaks_2: if len(peaks_2) == 0: return 0, 0, 0, 0, 0 elif len(peaks_2) == 1 and peaks_2[0]</pre>

	использовать if peak == -1		
4	В функции semi_width_subpeaks необходимо добавить проверку на случай, если крайние индексы пусты	Да	<pre>def semi_width_subpeaks(peak, channel: pd.Series, time: pd.Series): normalized = (channel - channel.min()) / (channel.max() - channel.m if peak is -1: return 0, 0, 0, 0, 0 if indices_left.any() and indices_right.any():</pre>
5	В функции k_semi_max возможно деление на ноль	Да	<pre>for element in time_difference: final_time[count] = element count += 1 if count == 0: break return final_time[:5]</pre>

Все участники оповещены о дате следующего собрания и исправленный код им предоставлен через систему управления версиями.

4.4 Верификация и завершение

Даты, в которые выполнялись исправления: 25-26.04.2024

Трудозатраты фазы: 2 чел-час

Дата собрания фазы верификации и завершения: 26.04.2024

Найденные ранее ошибки признаны исправленными, но обнаружена новая ошибка

Журнал дефектов

Номер ошибки	Описание ошибки	Согласие	Статус ошибки
1*	В функции subpeak в случае когда пики не найдены или найден только один пик, необходимо вернуть None вместо попытки обращения к максимальному пику	Да	Исправлено

Цели обзора кода достигнуты, указанные выше функции практически безошибочно исполняют свои задачи.

Итоговая статистика:

Всего на общий обзор кода потрачено 4 часа

Обзором кода были проверены 357 строк кода на языке программирования Python

Было найдено 5 ошибок и принято к исправлению 5, исправлено 5

Приложение 1 — Код программы.

```
import pandas as pd
import numpy as np
from typing import Union
from scipy.signal import find_peaks
import matplotlib.pyplot as plt

#channel = df['channel']
#insert into main code:
"""
o(n^2) - bad bad bad. maybe loc into ID in df. if means o(1)
for df in setup:
    for channel in df:
        if channel != 'time' :
            f'semi_max_{channel[-1]}' = mean_median(channel)
"""

#done
def semi_max(channel) -> Union[int, float]:
    return channel.max() / 2

# time = df['time'] where df with ID (means raindrop)
def semi_width(channel: pd.Series, time: pd.Series):
    normalized = (channel - channel.min()) / (channel.max() - channel.min())
```

```

peak_indx= np.where(normalized == normalized.max())[0]

if peak_indx.size == 0:
    return 0
# Берем первый пик с высотой 1
peak_index = peak_indx[0]

#r_plot = normalized.iloc[:peak_index].values <= 0.5
#l_plot = normalized.iloc[peak_index:].values <= 0.5
# Ищем точки слева и справа от пика, где амплитуда становится равной 0.5
indices_left = np.where(normalized.iloc[:peak_index].values <= 0.5)[0]
indices_right = np.where(normalized.iloc[peak_index:].values <= 0.5)[0]

if indices_left.any() and indices_right.size == 0:

    #если пик "упирается" справа
    left_index = indices_left[-1]
    print(left_index)
    half_width_time = (time.iloc[peak_index] - (time.iloc[left_index] +
        time.iloc[left_index])/2) * 2
    #plt.plot(time, normalized, label='Сигнал исходный')
    #plt.scatter(time.iloc[left_index], normalized.iloc[left_index], c='red',
marker='*', label='Слева')

# Добавляем легенду
plt.legend()

```

```

# Показываем график
plt.show()

return half_width_time, left_index

elif indices_left.size == 0 and indices_right.any():

    #если пик "упирается" слева
    right_index = peak_index + indices_right[0]
    half_width_time = ((time.iloc[right_index] + time.iloc[right_index - 1])/2 -
                       time.iloc[peak_index]) * 2
    plt.plot(time, normalized, label='Сигнал исходный')
    plt.scatter(time.iloc[right_index], normalized.iloc[right_index], c='blue',
marker='*', label='Справа')

# Добавляем легенду
plt.legend()

# Показываем график
plt.show()

return half_width_time, right_index

# Если индексы не пусты, продолжаем
elif indices_left.any() and indices_right.any():
    # Находим первый и последний индексы, соответствующие половинной
амплитуде
    left_index = indices_left[-1]
    next_left = left_index + 1

```

```

print(left_index)
right_index = peak_index + indices_right[0]
previous_right = right_index - 1

# Рассчитываем полуширину во времени
half_width_time = (time.iloc[right_index] + time.iloc[right_index - 1])/2 - \
    (time.iloc[left_index] + time.iloc[left_index + 1]) / 2

return half_width_time, right_index, left_index

else:
    print("Не удалось найти индексы для расчета полуширины.")
    return 0
#return half_width_time

```

```

def subpeak(channel, time, peak_return=False):

```

```

    normalized_series = (channel - channel.min()) / (channel.max() - channel.min())
    peak_indx = np.where(normalized_series == 1)[0]
    # Берем первый пик с высотой 1
    if peak_indx.size == 0:
        return 0, 0, 0, 0, 0

```

```

    peak_1 = channel.max()
    #пик
    #Пики по параметрам

```

```

    peaks_2, _ = find_peaks(normalized_series, height=peak_1*0.3,

```

```
prominence=(normalized_series.max()*0.2,  
normalized_series.max()*0.99))
```

```
if peak_return:  
    if peaks_2.any():  
        peaks = np.full(5, -1)  
        count = 0  
        for element in peaks_2:  
            peaks[count] = element  
            count += 1  
            if count >= 5:  
                break  
        return peaks  
    else:  
        return np.full(5, -1)
```

```
#лучший вариант - peaks_2:  
if len(peaks_2) == 0:  
    return 0, 0, 0, 0, 0  
elif len(peaks_2) == 1 and peaks_2[0] == normalized_series[normalized_series ==  
peak_1].index:  
    return 0, 0, 0, 0, 0
```

```
if peak_indx.size == 0:  
    return 0, 0, 0, 0, 0  
# Берем первый пик с высотой 1
```

```

peak_index = peak_indx[0]

indices_left = np.where(normalized_series.iloc[:peak_index].values <= 0.5)[0]
indices_right = np.where(normalized_series.iloc[peak_index:].values <= 0.5)[0]

if indices_left.any() and indices_right.size == 0:

    # если пик "упирается" справа
    left_index = indices_left[-1]
    print(left_index)
    half_width_time = (time.iloc[peak_index] - (time.iloc[left_index] +
                                                time.iloc[left_index]) / 2) * 2

    p1_indx = normalized_series[normalized_series == peak_1].index

    time_difference = np.abs(time.values[peaks_2] - time.values[peak_index])

    time_difference = np.delete(time_difference, np.where(time_difference == 0))
    time_difference = np.sort(time_difference)
    final_time = np.zeros(5)
    count = 0
    for element in time_difference:
        final_time[count] = element
        count += 1
        if count >= 5:
            break

    return final_time[:5]

```

```

elif indices_left.size == 0 and indices_right.any():

    # если пик "упирается" слева
    right_index = peak_index + indices_right[0]
    half_width_time = ((time.iloc[right_index] + time.iloc[right_index - 1]) / 2 -
                        time.iloc[peak_index]) * 2

time_difference = np.abs(time.values[peaks_2] - time.values[peak_index])

time_difference = np.delete(time_difference, np.where(time_difference == 0))
time_difference = np.sort(time_difference)
final_time = np.zeros(5)
count = 0
for element in time_difference:
    final_time[count] = element
    count += 1
    if count >= 5:
        break

return final_time[:5]

```



```

# Если индексы не пусты, продолжаем
elif indices_left.any() and indices_right.any():
    # Находим первый и последний индексы, соответствующие половинной
амплитуде
    left_index = indices_left[-1]
    next_left = left_index + 1
    print(left_index)
    right_index = peak_index + indices_right[0]
    previous_right = right_index - 1

    # Рассчитываем полуширину во времени
    half_width_time = (time.iloc[right_index] + time.iloc[right_index - 1]) / 2 -
(time.iloc[left_index] + time.iloc[left_index + 1]) / 2

    time_difference = np.abs(time.values[peaks_2] - time.values[peak_index])

    time_difference = np.delete(time_difference, np.where(time_difference == 0))
    time_difference = np.sort(time_difference)
    final_time = np.zeros(5)
    count = 0
    for element in time_difference:
        final_time[count] = element
        count += 1
        if count >= 5:
            break

```

```
return final_time[:5]
```

```
def semi_width_subpekas(peak, channel: pd.Series, time: pd.Series):  
    normalized = (channel - channel.min()) / (channel.max() - channel.min())  
    if peak is -1:  
        return 0, 0, 0, 0, 0  
  
    # Берем нужный сабпик  
    peak_index = peak  
  
    value = channel[peak]  
    indices_left = np.where(normalized.iloc[:peak_index].values <= value/2)[0]  
    indices_right = np.where(normalized.iloc[peak_index:].values <= value/2)[0]  
  
    if indices_left.any() and indices_right.size == 0:  
  
        # если пик "упирается" справа  
        left_index = indices_left[-1]  
        print(left_index)  
        half_width_time = (time.iloc[peak_index] - (time.iloc[left_index] +  
                                                    time.iloc[left_index]) / 2) * 2  
  
        # plt.plot(time, normalized, label='Сигнал исходный')  
        # plt.scatter(time.iloc[left_index], normalized.iloc[left_index], c='red',  
marker='*', label='Слева')  
  
        # Добавляем легенду  
        # plt.legend()
```

```

# Показываем график
# plt.show()

return half_width_time, left_index

elif indices_left.size == 0 and indices_right.any():

    # если пик "упирается" слева
    right_index = peak_index + indices_right[0]
    half_width_time = ((time.iloc[right_index] + time.iloc[right_index - 1]) / 2 -
                       time.iloc[peak_index]) * 2

    # plt.plot(time, normalized, label='Сигнал исходный')
    # plt.scatter(time.iloc[right_index], normalized.iloc[right_index], c='blue',
marker='*', label='Справа')

    # Добавляем легенду
    # plt.legend()

    # Показываем график
    # plt.show()

    return half_width_time, right_index

# Если индексы не пусты, продолжаем
elif indices_left.any() and indices_right.any():

    # Находим первый и последний индексы, соответствующие половинной
амплитуде
    left_index = indices_left[-1]
    next_left = left_index + 1
    print(left_index)

```

```
right_index = peak_index + indices_right[0]
```

```
previous_right = right_index - 1
```

```
# Рассчитываем полуширину во времени
```

```
half_width_time = (time.iloc[right_index] + time.iloc[right_index - 1]) / 2 - \
    (time.iloc[left_index] + time.iloc[left_index + 1]) / 2
```

```
return half_width_time, right_index, left_index
```

```
"""
```

```
plt.plot(time.values[peaks_1], normalized_series.values[peaks_1], 'x',
label='Пики_1')
```

```
plt.plot(time.values[subpeaks], normalized_series.values[subpeaks],
label='Пики_3')
```

```
plt.legend()
```

```
"""
```

```
#done
```

```
def mean_median(channel) -> Union[int, float]:  
    return channel.mean() - channel.median()
```

```
#done
```

```
def q10(channel) -> Union[int, float]:
```

```
    q10 = np.nanquantile(channel, 0.10)  
    return q10
```

```
#done
```

```
def q25(channel) -> Union[int, float]:
```

```
    q25 = np.nanquantile(channel, 0.25)  
    return q25
```

```
#done
```

```
def q75(channel) -> Union[int, float]:
```

```
    q75 = np.nanquantile(channel, 0.75)  
    return q75
```

```
#done
```

```
def q90(channel) -> Union[int, float]:
```

```

q90 = np.nanquantile(channel, 0.90)
return q90

#done
def k_semi_max(channel, time) -> Union[int, float]:
    semi_width_peak = semi_width(channel, time)
    if isinstance(semi_width_peak, tuple):
        semi_width_peak = semi_width_peak[0]
        print(f'{semi_width_peak} полуширина пика ')

    return (channel.max() - semi_max(channel))/semi_width_peak
else:
    return 0

if __name__ == "__main__":
    time_values = pd.Series([1, 2, 3, 4, 5, 6])
    channel_values = pd.Series([0, 1, 2, 3, 4, 5])

    result = semi_width(time_values, channel_values)
    print(result)

```

