TRABAJO FIN DE GRADO

Sistema para la extracción y análisis de construcciones sintácticas de código Python

Abel Busto Dopazo

DIRECTORES

FRANCISCO ORTIN SOLER

MIGUEL GARCÍA RODRÍGUEZ

Proyecto presentado en cumplimiento de los requisitos para el Grado en Ingeniería Informática del Software en el

Computational Reflection Research Group

Escuela de Ingeniería Informática

Universidad de Oviedo

Resumen

El objetivo principal de este proyecto de investigación es el diseño e implementación de un sistema de extracción de construcciones sintácticas de código Python para su posterior análisis y explotación. La representación sintáctica de programas de ordenador suele llevarse a cabo con estructuras comúnmente conocidas como Árboles de Sintaxis Abstracta (*ASTs*). La extracción de estas estructuras es un paso necesario para poder analizar cómo los programadores utilizan las distintas construcciones sintácticas del lenguaje de programación. Esta información puede ser utilizada en diferentes campos de aplicación, como la enseñanza de la programación (evitar patrones poco recomendables y enseñar lo de mayor calidad), la implementación de *IDE*s (proponer refactorizaciones de código) y la documentación de patrones, entre otros.

Para poder trabajar con grandes volúmenes de programas, es necesario almacenar esas estructuras en bases de datos. No obstante, la mayoría de los sistemas almacenan el código fuente como texto plano, no permitiendo así explotar de forma directa su estructura sintáctica. En cuanto a madurez, rendimiento y eficiencia, el modelo relacional es posiblemente el más utilizado en la actualidad. El principal problema es que la información sintáctica de los programas se representa jerárquicamente mediante estructuras de árbol o grafo. Por lo tanto, para almacenarlas en un sistema relacional, es necesario traducir los distintos patrones sintácticos (clases, métodos, sentencias, expresiones, etc.) en formato de tabla.

Para cada construcción sintáctica, se identifica la información relativa al tipo de nodo (heterogénea). También se incluye información de su contexto relativa a su nodo padre, de los nodos hijos y de sus niveles de profundidad y anchura. Así, se aúna información local y global de cada nodo.

El primer objetivo de este trabajo es la extracción del AST de cualquier programa Python. Para llevar a cabo este objetivo, el sistema propuesto modifica la salida del módulo AST de la Librería Estándar de Python (*Python Standard Library*) para enriquecer la información sintáctica proporcionada por su AST.

El segundo objetivo del presente proyecto es la transformación de esos ASTs en un modelo relacional que permita almacenar grandes volúmenes de código, pudiéndolos consultar de un modo eficiente. El sistema implementa un mecanismo para traducir los ASTs obtenidos en vectores n-dimensionales que puedan ser almacenados en forma de tabla. De este modo, será posible obtener de un modo eficiente las distintas construcciones sintácticas empleadas por los programadores de numerosos proyectos reales.

El tercer objetivo es mostrar un ejemplo de cómo las estructuras sintácticas obtenidas pueden emplearse para minar, analizar y documentar los patrones sintácticos recurrentes utilizados por los programadores. De este modo, se podrían detectar fragmentos de código altamente repetidos (*idioms*), asociar patrones al nivel de experiencia de los programadores, identificar patrones que pudiesen dar lugar a errores o reconocer los patrones de uso de una nueva característica añadida a una versión reciente de un lenguaje. Puesto que todas estas posibilidades son numerosas y sobrepasan el tiempo asignado al Trabajo Final de Grado, nos limitamos a realizar, a modo de ejemplo, análisis frecuencia de construcciones sintácticas, de detección de anomalías o valores atípicos y de correlación significativamente distinta entre construcciones sintácticas de programadores nóveles y expertos.

Palabras clave

Construcciones sintácticas, árboles de sintaxis abstractas, detección de anomalías, lenguajes de programación, Python

Abstract

The main objective of this research final degree project is the design and implementation of a system for extracting syntactic constructions from Python code for its subsequent analysis and exploitation. The syntactic representation of computer programs is usually carried out with structures commonly known as Abstract Syntax Trees (ASTs). Extracting these structures is a necessary step to analyze how programmers use the various syntactic constructions in a programming language. This information can be used in different application fields, such as programming education (avoiding undesirable patterns and teaching higher quality ones), IDE implementation (suggesting code refactorings), and documentation of patterns, among others.

To work with large volumes of programs, it is necessary to store these structures in databases. However, most systems store the source code as plain text, which does not allow the direct exploitation of its syntactic structure. In terms of maturity, performance, and efficiency, the relational model might possibly be the most used these days. The main problem is that the syntactic information of programs is represented hierarchically through tree or graph structures, not as tables. Therefore, to store them in a relational system, it is necessary to translate the various syntactic patterns (classes, methods, statements, expressions, etc.) into table format.

For each syntactic construction, information related to the type of node (heterogeneous) is identified. Moreover, information about its context, relative to its parent node, child nodes, and their depth and width levels, is also included. In this way, local and global information of each node is combined.

The first objective of this work is the extraction of the ASTs from any Python program. To achieve this goal, the proposed system modifies the output of the AST module from the Python Standard Library to enrich the syntactic information provided by its AST.

The second objective of this project is to transform these ASTs into the relational model to allow storing large volumes of code, making them efficiently queryable. The system implements a mechanism to translate the obtained ASTs into n-dimensional vectors that can be stored in table form. In this way, it will be possible to efficiently obtain the different syntactic constructions used by programmers in numerous real projects.

The third objective is to show an example of how the obtained syntactic structures can be used to mine, analyze, and document the recurring syntactic patterns used by programmers. This could detect highly repeated code fragments (idioms), associate patterns with the programmers' experience levels, identify patterns that could lead to errors, or recognize usage patterns of a new feature added to a recent version of the programming language. Since all these possibilities are numerous and exceed the time allocated for the Final Degree Project, we limit ourselves to providing an analysis example: we detail the syntactic construction anomalies or outliers found real Python programs.

Keywords

Syntactic constructs, abstract syntax trees, anomaly detection, programming languages, Python

Agradecimientos

En primer lugar, me gustaría agradecer encarecidamente la colaboración e implicación de mis dos tutores. En especial a Miguel, por su atención y plena disponibilidad para lograr que este trabajo se haya completado logrando que ambas partes hayamos quedado satisfechas. Además, por enseñarme y ayudarme a tomar decisiones sobre mi futuro profesional.

Por otro lado, agradecer a mis amigos y a mi pareja por su apoyo a pesar de no entender casi nada de lo que estaba realizando en este trabajo.

Por último, eternamente agradecido a mis padres y a mi hermano por brindarme un entorno cómodo y tranquilo en el que trabajar.

Este trabajo ha sido financiado con fondos de la Universidad de Oviedo a través de su apoyo a grupos de investigación oficiales (GR-2011-0040).

Índice de contenido

Lista de Figuras

Lista de Tablas

1 Introducción

En la era digital actual, los repositorios de código como *GitHub*, *GitLab* y *Bitbucket* han experimentado un crecimiento exponencial, convirtiéndose en pilares fundamentales para la colaboración y el desarrollo de software [1]. Estos repositorios no solo facilitan el trabajo en equipo, sino que también almacenan grandes cantidades de código, proporcionando una fuente inestimable de información para el análisis y la explotación de datos a gran escala. Esta línea de investigación que utiliza grandes volúmenes de código para realizar modelos predictivos a partir de grandes cantidades de código se denomina *Big Code* [2], como la unión de los términos *Big data* y *source Code*. Dos ejemplos famosos de la aplicación de técnicas de *big data* a estos repositorios son *GitHub Copilot* o ChatGP*T* [3], que asisten a los desarrolladores sugiriendo fragmentos de código, completando funciones y ayudando a resolver problemas de programación de manera más eficiente. No obstante, existen numerosos ejemplos adicionales que pueden consultare, por ejemplo, en [4].

En el contexto de la representación y análisis de programas de ordenador, los Árboles de Sintaxis Abstracta (ASTs) juegan un papel crucial. Los ASTs son estructuras jerárquicas heterogéneas que representan la sintaxis de un programa de manera abstracta, despojándolo de detalles innecesarios que se incluyen en las gramáticas concretas para poder procesar el código fuente sin ambigüedades sintácticas. Los ASTs, una vez realizado el análisis sintáctico, se centran en la estructura del programa, permitiendo hacer recorridos sobre el mismo para, por ejemplo, realizar análisis semántico o generación de código. Además, los ASTs permiten a los desarrolladores y analistas examinar y manipular el código de manera más eficiente, facilitando tareas como la refactorización, la optimización y la detección de errores.

Puesto que los ASTs representan la información sintáctica de los programas de entrada, resulta interesante utilizarlos para minar información y crear modelos predictivos. Sin embargo, la mayoría de los algoritmos de minería de datos están diseñados para trabajar con datos en formato tabular, en el que cada muestra, individuo, entrada o instancia (fila de la tabla) tiene asociado un número fijo de características o *features* (columnas). Por lo tanto, es necesario desarrollar métodos que permitan traducir la información contenida en un AST a una estructura tabular, evitando la pérdida de información.

El lenguaje de programación Python ha ganado una popularidad significativa en los últimos años debido a su simplicidad, versatilidad y amplia adopción en diversos campos, incluyendo el desarrollo web, la ciencia de datos, la inteligencia artificial y la automatización de tareas [5] [6]. Su sintaxis clara y legible, junto con una comunidad activa y una vasta colección de bibliotecas y *frameworks*, han consolidado a Python como una de las elecciones preferidas, tanto para programadores noveles como para expertos. Tal y como se puede apreciar en la Figura 1, el ranking de popularidad de lenguajes de programación Tiobe identifica a Python como el lenguaje más popular en junio de 2024, sacando más de un 5% de diferencia respecto al segundo (C++). Por este motivo, hemos elegido Python como lenguaje para la realización del presente trabajo.

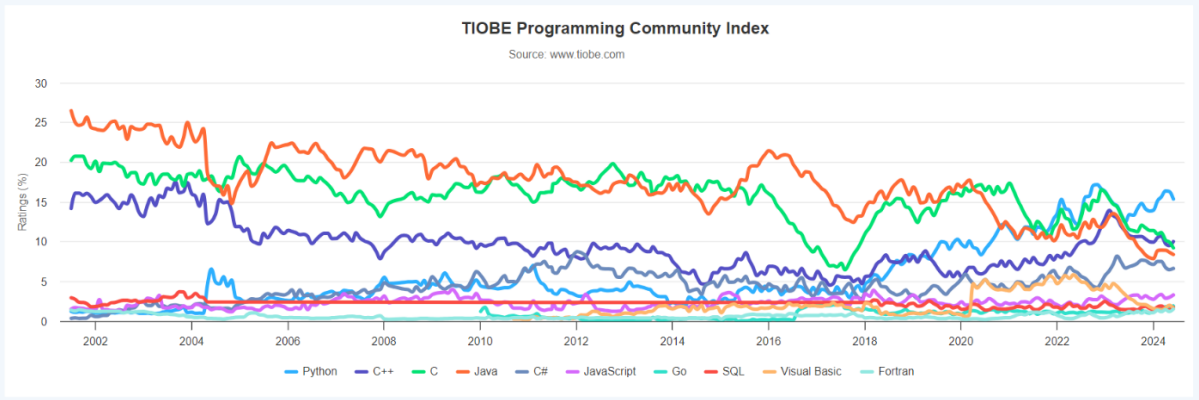

Figura 1: Índice Tiobe de popularidad de lenguajes de programación (junio 2024).

El trabajo propuesto busca aprovechar los ASTs para extraer y analizar construcciones sintácticas de código Python. La extracción de estas estructuras permitirá almacenar y gestionar grandes volúmenes de datos en bases de datos relacionales, transformando los ASTs en vectores n-dimensionales para que puedan ser consultados de manera eficiente. Esta decisión no solo facilita el almacenamiento y la consulta de datos sintácticos, sino que también abre un abanico de posibilidades para el análisis y la explotación de patrones de programación, debido a que facilita la confección de conjunto de datos (*datasets*) n-dimensionales utilizados en los algoritmos tradicionales de aprendizaje automático y minería de datos.

Una de las aplicaciones más prometedoras de este sistema es la documentación de construcciones recurrentes utilizadas por programadores con diferentes niveles de experiencia. Por ejemplo, los profesores de programación podrían identificar patrones de código comúnmente empleados por estudiantes no usados por los expertos y señalar aquellos que sean propensos a errores, ofreciendo alternativas más eficientes y robustas basadas en las prácticas de programadores expertos [7]. Asimismo, el análisis del código fuente podría predecir el nivel de experiencia de los programadores según las construcciones sintácticas que utilizan, permitiendo a los entornos de desarrollo integrado (*IDE*s) sugerir mejores prácticas y técnicas avanzadas adaptadas en función de la experiencia del programador. Otro ejemplo de aplicación podría ser el desarrollo de sistemas de tutoría inteligentes (*ITS*s) [8]. Estos sistemas se basan en personalizar el proceso de aprendizaje para cada estudiante, identificando sus debilidades y sugiriendo ejercicios específicos para mejorar sus habilidades en función de su nivel de experiencia. Los datos no solo podrían utilizarse para mejorar la calidad del código producido, sino que también podrían servir como herramientas pedagógicas y de mentoría, guiando a los programadores noveles hacia prácticas más avanzadas y eficaces. De esta manera, se optimizaría el proceso de enseñanza y se fomentaría un aprendizaje más efectivo y eficiente. Estas son posibles utilizaciones de la base de datos creada en este trabajo, no incluidas en el mismo, pero que sí probablemente formarán parte del trabajo futuro a realizar por el alumno.

Las principales contribuciones de este trabajo son:

1. La creación de un conjunto de datos formado por código fuente Python programado por desarrolladores expertos y programadores nóveles (alumnos de primer curso de Ingeniería Informática).
2. Un diseño e implementación de un AST con un nivel de detalle de sus entidades más preciso que el utilizado por el módulo AST de la Librería Estándar de Python (*Python Standard Library, PSL*) [9], para así poder sacar más provecho de la información más precisa en la explotación de los datos. Para ello, modificamos el análisis sintáctico que realiza la *PSL* para que cree estos nuevos árboles enriquecidos en lugar de los originales.

3. Diseño de una base de datos relacional que nos permita almacenar en tablas las estructuras sintácticas propias de los programas Python.
4. Un sistema de extracción de construcciones sintácticas de código Python que implemente técnicas de transformación de ASTs a tablas que nos permitan almacenar y analizar grandes volúmenes de datos de forma eficiente.
5. Un ejemplo de análisis de los datos extraídos. Más concretamente, realizaremos un análisis de frecuencia de las construcciones sintácticas, detección de anomalías o valores atípicos y un estudio de correlación con el nivel de experiencia de los programadores.

Este trabajo se encuentra organizado de la siguiente manera. El siguiente capítulo describe el trabajo relacionado. El Capítulo 3 presenta la descripción del sistema propuesto y el Capítulo 4 describe la metodología seguida para la evaluación del sistema. En el Capítulo 5 se explican los resultados de los diferentes experimentos, mientras que el Capítulo 6 detalla las conclusiones y el trabajo futuro. Finalmente, el Capítulo 7 detalla la planificación y presupuesto.

2 Trabajo relacionado

Peng y Zhang estudiaron las características comunes del lenguaje Python utilizadas en diversos proyectos [10]. Utilizando la herramienta PYSCAN, los autores realizaron un análisis automatizado para escanear y examinar las construcciones sintácticas específicas del lenguaje presentes en el código fuente. El trabajo abarcó 35 proyectos populares de Python de ocho dominios de aplicación diferentes, cubriendo más de 4.3 millones de líneas de código. Este estudio generó estadísticas detalladas y tendencias sobre la frecuencia y el uso de estas características en diferentes contextos de desarrollo de software. La investigación se centra en identificar patrones comunes y recurrentes en el código Python, proporcionando una visión empírica y cuantitativa del uso del lenguaje en proyectos reales. Los resultados obtenidos ofrecen conclusiones importantes para desarrolladores y equipos de software, facilitando la comprensión y optimización de prácticas de programación basadas en evidencia empírica.

Dakhel, Desmarais y Khomh analizaron cómo las distribuciones estadísticas de los patrones sintácticos pueden ser utilizadas para evaluar la experiencia de los desarrolladores [11]. Este trabajo se basa en la distribución de patrones sintácticos (*SP*s), utilizando específicamente la ley de *Zipf* para identificar la habilidad de los programadores. La ley de *Zipf*, que describe la frecuencia de ocurrencia de elementos en muchos tipos de datos, se emplea aquí para identificar patrones distintivos en el código fuente desarrollado por programadores con diferentes niveles de pericia. Este enfoque permite una evaluación cuantitativa y objetiva de las habilidades de los desarrolladores basada en la complejidad y la sofisticación de los patrones sintácticos que emplean. Los resultados del estudio sugieren que las variaciones en las distribuciones de patrones sintácticos pueden correlacionarse con niveles de experiencia y competencia de los programadores.

Robles *et al.* desarrollan un sistema para evaluar el nivel de competencia en Python mediante el análisis del código [12]. Este sistema clasifica a los programadores en seis niveles de competencia basándose en características específicas del código, como la complejidad ciclomática, el uso de construcciones avanzadas y patrones de estilo. Analizan grandes volúmenes de código fuente mediante técnicas de minería de datos para identificar patrones recurrentes, y los clasifican correlacionando estos patrones con diferentes niveles de habilidad. El estudio se llevó a cabo con 165 programas escritos por 33 estudiantes universitarios. Estos niveles no son predefinidos, sino que se determinan en función de los datos analizados, utilizando algoritmos que agrupan las características del código en niveles de competencia. Esto permite identificar fragmentos de código que pueden ser entendidos por desarrolladores con un cierto nivel de competencia y ayudar en el proceso de incorporación de nuevos desarrolladores en proyectos de software de código abierto.

Este trabajo continúa la línea de investigación iniciada en los trabajos de Losada *et al.* [8] y Ortin *et al.* [13]. A diferencia de los trabajos previos, que estaban centrados en el lenguaje Java, este estudio se enfrenta a un contexto diferente y un desafío mayor debido a las características distintivas de Python. Python se distingue por su amplia variedad de construcciones sintácticas y su flexibilidad como lenguaje interpretado de alto nivel. Esto implica que los patrones sintácticos en Python pueden abarcar desde estructuras simples hasta paradigmas de programación más complejos, como la programación orientada a objetos y la programación funcional. Por lo tanto, este nuevo trabajo no solo busca extraer y analizar patrones comunes, sino también capturar la diversidad y la riqueza de las prácticas de codificación en Python, ofreciendo resultados útiles para la mejora de herramientas de desarrollo y la comprensión de las tendencias de uso del lenguaje.

3 Descripción del sistema

La Figura 2 muestra el diagrama de la arquitectura de nuestro sistema. A continuación, describimos someramente sus elementos para posteriormente profundizar en los mismos. La entrada del sistema es un conjunto de programas Python obtenidos tanto de GitHub como de una asignatura de primer año de programación del Grado en Ingeniería Informática del Software de la Universidad de Oviedo (Capítulo 4.1). De esta forma, podremos crear una base de datos con código escrito por principiantes y expertos. La salida es un conjunto de datos con las construcciones sintácticas extraídas incluyendo su relación con el nivel de experiencia del programador (Base de Datos) y un conjunto de informes acerca de las construcciones sintácticas de los programas.

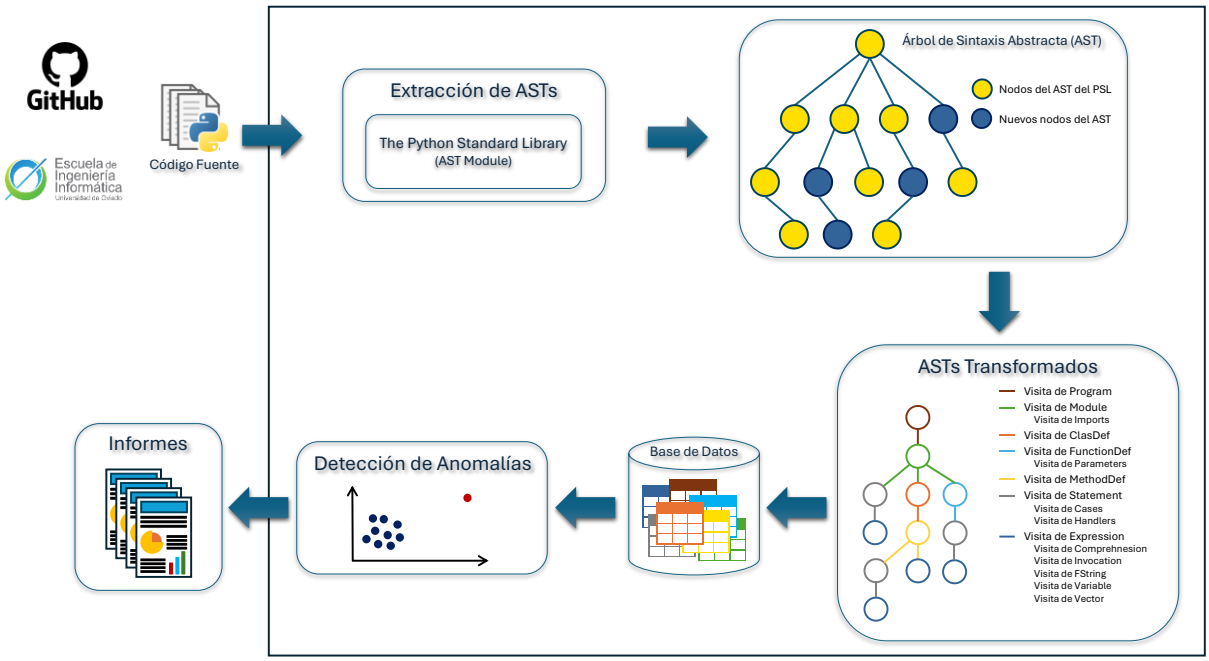

Figura 2: Arquitectura del sistema propuesto para la extracción de información sintáctica y generación de informes.

Lo primero que hacemos es modificar la salida del módulo AST de la *Python Standard Library* (*PSL*), aumentando la información sintáctica ofrecida por éste. Este se hace implementando el patrón de diseño *Visitor* [14] que, recorriendo el AST original, crea uno nuevo con información más específica. Para ello, rediseñamos el AST, añadiendo nuevos tipos de nodos, atributos y relaciones, proporcionando así una información más detallada que la ofrecida por el módulo original (Sección 3.1).

A continuación, se realiza una trasformación de los ASTs a tablas de un modelo relacional que diseñamos tras el análisis exhaustivo de las distintas construcciones sintácticas del lenguaje Python, analizando su gramática libre de contexto. El sistema identifica siete tipos diferentes de construcciones sintácticas principales, detalladas en la Sección 3.2: programa, modulo, definición de clase, de función y de método, sentencia y expresión. Además de estas construcciones, el sistema distingue casos concretos de sentencias como *cases* y *handlers*; y de expresiones como *comprehnesion*, invocación, cadena formateada (*f-string* en Python), variable y vector. Por último, también se identifican las construcciones utilizadas para importar módulos y paquetes, así como las utilizadas en la especificación de los parámetros en las distintas definiciones de Python, y los

argumentos en las llamadas a funciones y métodos. En total, el sistema propuesto identifica 16 tipos distintos de construcciones sintácticas. Para cada una de ellas, se genera una tabla en una base de datos relacional, que almacena la información de cada nodo del AST. Se ha llevado a cabo un proceso manual de extracción de características (*feature engineering*) para traducir las estructuras de árbol en tablas (Sección 3.2).

Posteriormente, los datos de las tablas son procesados y filtrados para proceder con los análisis a realizar (Sección 5). Si las anomalías son debidas a errores de medición (entradas que no debían haber sido consideradas) se eliminan; en caso contrario, se incluyen en un informe de anomalías.

3.1 Extracción de ASTs

Tal y como hemos mencionado anteriormente, la principal estructura de datos para representar sintácticamente un programa es el Árbol de Sintaxis Abstracta (*Abstract Syntax Tree*, *AST*). Cada nodo de un AST representa una construcción sintáctica de un programa como una definición de un tipo, una variable, la invocación a un método o una expresión aritmética. Python es un lenguaje de programación comúnmente interpretado, conocido por su simplicidad y legibilidad, lo que facilita la escritura y el mantenimiento de código [15]. El análisis de las construcciones sintácticas en Python es un reto debido al elevado número de construcciones sintácticas que ofrece el lenguaje. Para abordar este desafío, hemos utilizado el módulo AST de la *Python Standard Library* (*PSL*) [9]. Este módulo proporciona una representación abstracta y jerárquica del código fuente de Python, permitiendo el recorrido y manipulación de la estructura del código. Es posible descomponer los programas en sus componentes básicos, facilitando así el estudio de las distintas construcciones sintácticas empleadas por los programadores.

A continuación, vamos a explicar el proceso para la modificación del AST generado por la *PSL*. Por cada fichero Python, la *PSL* realiza un análisis sintáctico del mismo y genera un AST. Mediante el patrón de diseño *Visitor* [14] recorremos el AST original para añadir información más detallada sobre la estructura sintáctica del programa, incluyendo nuevas categorías sintácticas, relaciones o atributos.

A modo de ejemplo, todas las operaciones binarias en Python son representadas en el *PSL* como instancias de `BinOp`, dificultando el conocimiento del tipo concreto de operación que representa. Para dar más información acerca de la expresión, hemos añadido nodos que representan operaciones binarias concretas en función del operador: operaciones aritméticas (`Arithmetic`), potencias (`Pow`), desplazamiento de bits (`Shift`) u operaciones lógicas a nivel de bits (`BWLogical`), entre otras. Esta información extra nos permite identificar de un modo más sencillo que, por ejemplo, la primera suele ser más habitual entre los programadores noveles, pudiéndose detectar como patrón común a este tipo de programadores (la última es más comúnmente utilizada por programadores expertos). Lo mismo sucede cuando declaramos un valor o un literal. La *PSL* interpreta todos los literales como instancias de una misma entidad `Constant`. Sin embargo, nuestro sistema tiene en cuenta el tipo de literal y permite clasificar la expresión con una de las siguientes clases: `IntLiteral`, `FloatLiteral`, `ComplexLiteral`, `NoneLiteral`, `BoolLiteral`, `StringLiteral` o `EllipsisLiteral`.

Además de las nuevas clases, nuestro diseño también añade nuevas relaciones para almacenar información sintáctica de interés. Por ejemplo, los nodos de sentencias o expresiones del nuevo AST incorporan información relativa al rol que éstos juegan en su construcción sintáctica padre. Así, por ejemplo, una asignación puede jugar dos roles distintos si su padre es una sentencia `while`: podría ser la condición del bucle o formar parte de su cuerpo. La información que denota cuándo una

asignación se utiliza como condición es significativa, por ejemplo, para detectar programadores no principiantes (éstos rara vez usan operaciones aritméticas en las condiciones de `while`). Estas modificaciones nos facilitan ampliar el nivel de detalle en comparación con la versión original, donde solamente conoceríamos la clase que define a un nodo y las que definen a sus hijos.

La Figura 3 muestra la diferencia entre las 58 clases del AST de la *PSL* [9] y las 80 definidas en el nuevo diseño propuesto.

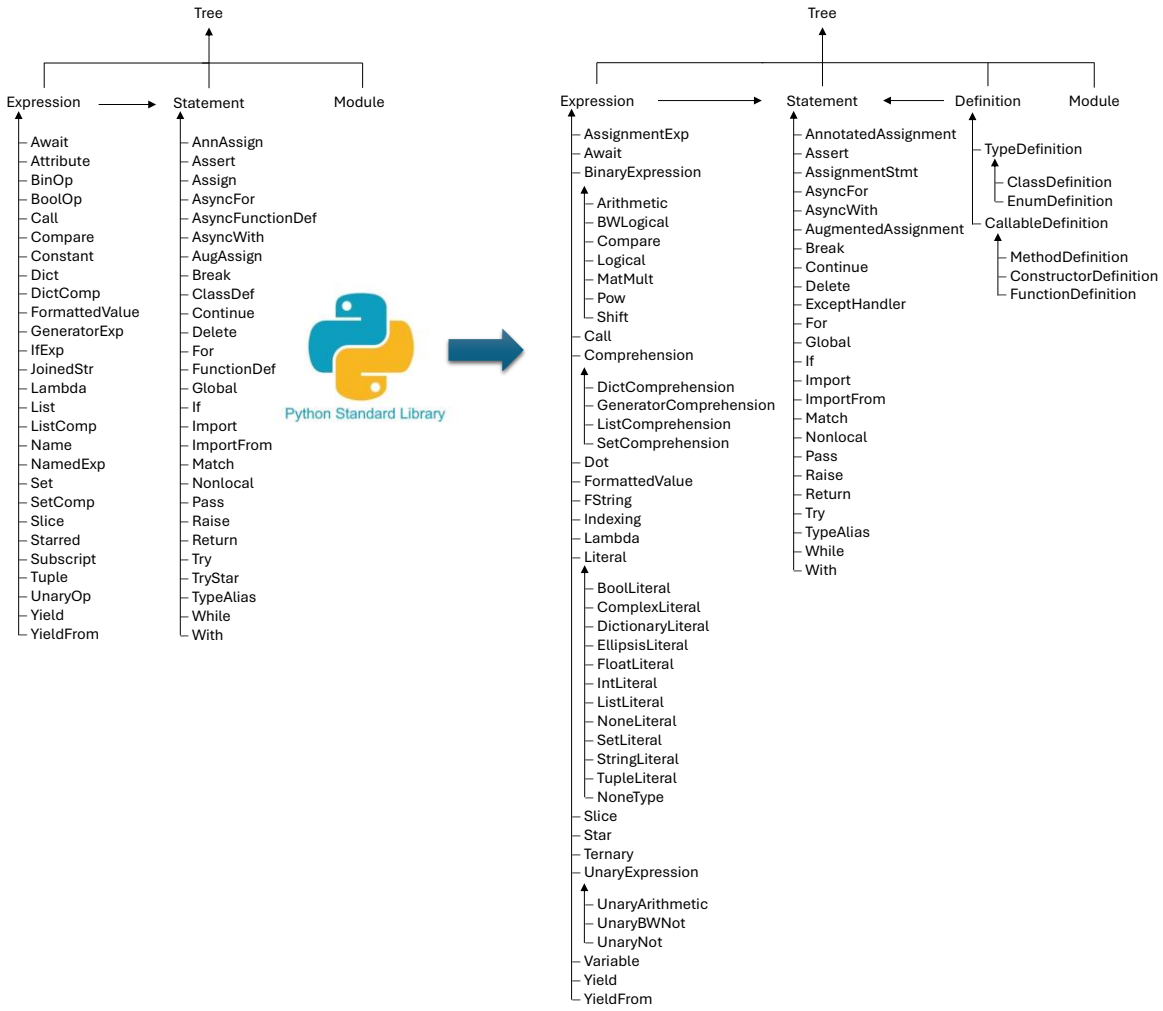

Figura 3: Relación de clases del AST del PSL y del sistema propuesto.

3.2 Generación de tablas

Tal y como mencionamos anteriormente, los algoritmos de minería de datos y aprendizaje automático pueden utilizarse para extraer información valiosa acerca de las construcciones sintácticas. Sin embargo, muchos de estos algoritmos están diseñados para trabajar con datos en formato tabular. Por esta razón, es necesario convertir los ASTs en tablas para poder aplicar estos algoritmos de manera efectiva. Lo primero es definir los distintos tipos de nodos del *AST* que tengan una estructura común (homogénea), para poder almacenarlos en la misma tabla. Identificamos los siguientes siete tipos de nodos: *program* (conjunto de directorios y ficheros Python que componen un proyecto

independiente), *module* (fichero.py), *class definition* (definiciones de clases), *function definition* (definición de funciones), *method definition* (definiciones de métodos), *statement* (sentencias) y *expression* (expresiones).

Además de estas entidades principales, usamos una tabla *imports* para guardar información resumida de los `imports` que usa un módulo. La información de los parámetros de las definiciones de funciones y métodos se guarda en *parameters*. También usamos tablas para representar información que solo está presente en el caso de las siguientes sentencias. Usamos *case*, para almacenar información exclusiva de las sentencias de tipo `Match` y *handlers*, para la información relativa a las sentencias `Try` y `TryStar`. Por último, almacenamos de forma independiente información específica de algunas expresiones, como *comprehensions* (generadores de listas, diccionarios, generadores, tuplas y sets), *callargs* (invocaciones a funciones), *fstring* (cadenas de texto formateadas), *variable* (variables) y *vector* (información relativa a los literales de tipo lista, diccionario, tuplas y sets). Por último, incluimos una tabla *nodes* que sirve para relacionar cada nodo con su nodo padre. Esta tabla contiene únicamente los identificadores del elemento, de su elemento padre y, además, el nombre de la tabla a la que pertenece el elemento padre para facilitar las consultas. Para cada una de estas entidades identificadas, se ha llevado a cabo un proceso manual de extracción y definición de características también conocido como *feature engineering*.

Para realizar la traducción, hacemos uso del patrón de diseño *Visitor* [14]. Recorremos el AST obteniendo y almacenando información de cada construcción sintáctica en una tabla correspondiente de la base de datos. Para la recolección de la información es necesario pasar información adicional de nodos padres a hijos y viceversa. Un nodo padre, por ejemplo, una expresión, trasmite información contextual hacia abajo relativa a sus características como su profundidad o su categoría sintáctica, para que sus nodos hijos puedan almacenar información relativa a su padre. Por otro lado, se trasmite información hacia arriba: un nodo de definición de método trasmite información, como por ejemplo su número de anotaciones de tipos o el tamaño de su cuerpo, hacia su nodo padre (típicamente una definición de clase) para que en este se pueda almacenar información relativa a las características de sus hijos.

Un ejemplo concreto de este funcionamiento se da entre los `FunctionDef` y los `Statements`. El nodo `FunctionDef` pasa a sus hijos su categoría sintáctica para que estos puedan rellenar la información relativa a su padre (atributo `parent`). A su vez, los hijos de tipo `Statement` pasan su categoría sintáctica al nodo `FunctionDef` para que este pueda llenar la información relativa a la distribución de categorías sintácticas en su cuerpo (atributos `expression_pct` y `body_count`).

Además, se ha realizado un sistema de claves internas para poder identificar la estructura de árbol concreta de cada programa. La Figura 4 muestra el modelo entidad relación con el diseño de la base de datos empleado en el sistema.

Las Tablas 1 a 16 mostradas más abajo detallan toda la información almacenada cada una de las construcciones sintácticas. A modo de ejemplo inicial, la Tabla 7 (*statements*) muestra la información almacenada para las sentencias. Además del nombre de la clase de la sentencia (su categoría sintáctica), guardamos la categoría sintáctica de su nodo padre. También, almacenamos el rol que la sentencia juega en el nodo padre. Asimismo, almacenamos la distancia desde el nodo raíz al actual (altura) y el número de arcos desde el actual al nodo hoja más distante (profundidad). Para posibilitar obtener información con respecto a los hijos de la sentencia se almacenan las categorías de sus tres primeros hijos.

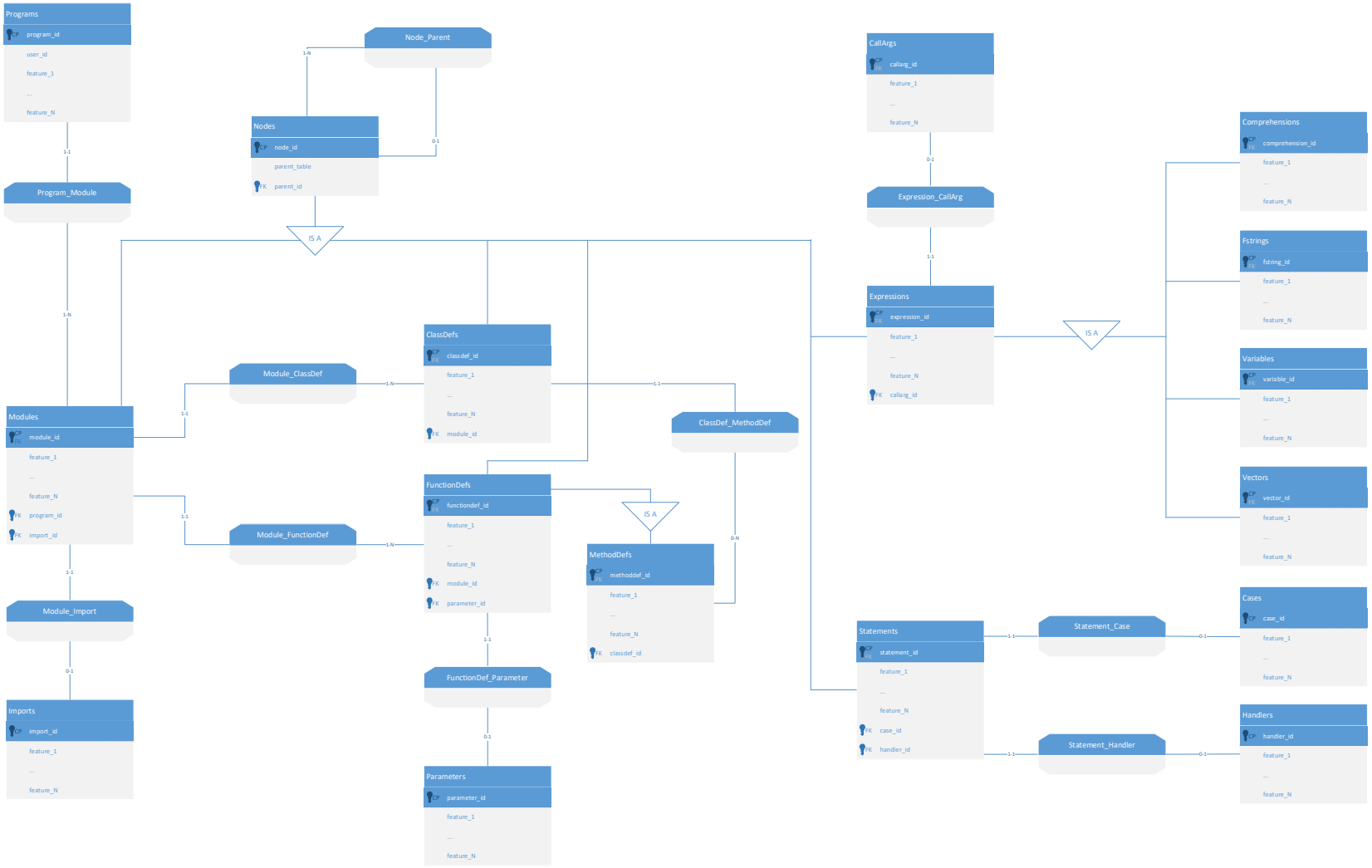

Figura 4: Diagrama entidad relación con las tablas de la base de datos

A continuación, explicamos las características extraídas para cada construcción sintáctica.

3.2.1 Programs

La Tabla 1 muestra la información almacenada para cada programa. Incluimos diferentes porcentajes definidos en ese programa (clases, enumerados, funciones…), todos ellos obtenidos de la sintaxis de sus hijos, e información sobre la implementación de las clases en paquetes.

Tabla 1: Características de programa.

Nombre	**Descripción**	**Dominio**
Name	Nombre del proyecto	String
Has subdirs with code	Si en el directorio base del proyecto hay algún subdirectorio con ficheros Python pero sin un fichero __init__.py en el	True or False.
Has packages	Si en el directorio base del proyecto hay algún subdirectorio con ficheros Python y un fichero __init__.py en el	True or False
Number of modules	Número de ficheros Python en el total del proyecto	Integer
Number of subdirs with code	Número de subdirectorios con ficheros Python en su interior	Integer
Number of packages	Número de subdirectorios con ficheros Python en su interior, pero sin un fichero __init__.py	Integer
Class defs pct	Proporción de las definiciones dentro del proyecto que son definiciones de clases	[0, 1]
Function defs pct	Proporción de las definiciones dentro del proyecto que son definiciones de funciones	[0, 1]
Enum defs pct	Proporción de las definiciones dentro del proyecto que son clases enumeradas	[0, 1]
Has code root package	Si el proyecto tiene ficheros Python en el directorio base	True or False
Average defs per module	Número medio de definiciones que hay en cada fichero Python del proyecto	Real
Expertise level	Si el usuario que escribió este proyecto es experto o	BEGINNER | EXPERT

Nombre	Descripción	Dominio
	principiante	
UserID	Identificador del usuario que escribió este proyecto	Unique ID (Integer)

3.2.2 Modules

En lo que respecta a los módulos (ficheros Python), calculamos distintas características. Se incluyen el nombre del fichero y la convención de nombrado que sigue, también se incluyen diferentes proporciones de definiciones (clases, funciones y enumerados). Además, se incluye información relativa a las proporciones de sentencias y expresiones presentes en el fichero, contadores de funciones y clases y, alguna información extra extraída de los hijos del módulo. La Tabla 2 muestra las características almacenadas para un módulo.

Tabla 2: Características para el módulo.

Nombre	Descripción	Dominio
Name	Nombre del fichero Python	String
Name convention	La convención utilizada para el nombre del fichero.	Lower | Upper | CamelLow | CamelUp | SnakeCase | Discard | NoNameConvention
Has doc string	Si el fichero tiene un comentario de módulo. Esto es una cadena como primer hijo del módulo	True or False.
Global statements pct	Proporción de los hijos del módulo que son sentencias (sin contar imports y definiciones)	[0, 1]
Global expressions pct	Proporción de los hijos del módulo que son expresiones	[0, 1]
Number of classes	Número de clases definidas en el módulo	Integer
Number of functions	Número de funciones definidas en el módulo	Integer
Class defs pct	Proporción de las definiciones de este módulo que son clases	[0, 1]
Function defs pct	Proporción de las definiciones de este módulo que son funciones	[0, 1]
Enum defs pct	Proporción de las definiciones de este módulo que son clases enumeradas	[0, 1]
Average statements function body	Número medio de sentencias en el cuerpo de las funciones de este módulo	Real

Nombre	Descripción	Dominio
Average statements method body	Número medio de sentencias en el cuerpo de los métodos de este módulo	Real
Type annotation pct	Proporción de anotaciones de tipo en los parámetros y en las funciones	[0, 1]
Has entry point	Si el fichero tiene el idiom "`if __name__ == '__main__'`"	True or False

3.2.3 Imports

Esta información es complementaria a la tabla de módulos ya que almacena la información de los *imports* (una entrada por cada módulo).

Tabla 3: Características para los imports.

Nombre	Descripción	Dominio
Number imports	Número de imports distintos en el módulo	Integer
Module imports pct	Proporción de imports simples (nodo Import)	[0, 1]
Average imported modules	Número medio de elementos importados por cada nodo Import	Real
From imports pct	Proporción de imports from (nodo ImportFrom)	[0, 1]
Average from imported modules	Número medio de elementos importados por cada nodo ImportFrom	Real
Average as in imported modules	Número medio de elementos importados con un alias con respecto a los nodos ImportFrom	Real
Local imports pct	Proporción de imports que no son definidos al comienzo del fichero	[0, 1]

3.2.4 Class Definitions

Respecto a la definición de clases, almacenamos la convención que sigue y la longitud del nombre de la clase. Contabilizamos además los decoradores de la clase, los métodos, las clases base que utiliza, el número de sentencias que forma la clase y el número de *keywords* (distintas de "`meta=`") que utiliza la clase. Además, guardamos si la clase es un enumerado y si la clase tiene un comentario de clase (su primer hijo es una cadena), si la clase tiene una anotación de tipo genérica (por ejemplo, "`class list[T]`") y si la clase usa una meta clase (define el *keyword* "`meta=`" en la cláusula de herencia).

Para identificar los tipos de métodos que se definen en la clase hemos identificado 7 tipos de métodos especiales: métodos privados, métodos mágicos, métodos asíncronos, métodos de clase, métodos estáticos, métodos abstractos y métodos de propiedad. Estos tipos se identifican en función de su nombre (por ejemplo, un método privado empieza por _) o por las anotaciones que tiene el método (por ejemplo, `@classmethod` para un método de clase). De cada uno de estos tipos almacenamos la proporción de estos métodos que hay definidos en la clase.

Por último, guardamos las proporciones de anotaciones de tipos en los parámetros y tipos de retorno de los métodos de la clase, la proporción de expresiones que se definen en el cuerpo de la clase, proporción de asignaciones que se definen en el cuerpo de la clase, el número medio de sentencias en los cuerpos de los métodos y el código fuente de la propia clase como cadena de texto.

Tabla 4: Características para la definición de clases.

Nombre	**Descripción**	**Dominio**
Name convention	La convención utilizada para el nombre de la clase.	Lower | Upper | CamelLow | CamelUp | SnakeCase | Discard | NoNameConvention
Is enum class	Si la clase es una clase enumerada (hereda de la clase Enum)	True or False
Number of characters	Número de caracteres del nombre de la clase	Integer
Number of decorators	Número de decoradores de la clase	Integer
Number of methods	Número de métodos definidos en la clase	Integer
Number of base classes	Número de clases base que se definen en la clase	Integer
Has generic type annotations	Si la clase tiene una anotación de tipo genérico	True or False.
Has doc string	Si la clase tiene un comentario de clase. Esto es si su primer hijo es una cadena	True or False.
Body count	Número de sentencias en el cuerpo de la clase	Integer
Assignments pct	Proporción de las sentencias en el cuerpo de la clase que son asignaciones	[0, 1]
Expressions pct	Proporción de las sentencias en el cuerpo de la clase que son expresiones	[0, 1]
Uses metaclass	Si la clase usa una meta class. Esto es si define una keyword "`meta=`" en la	True or False.

Nombre	Descripción	Dominio
	cláusula de herencia	
Number of keywords	Número de keywords diferentes de metaclass que tiene la clase	Integer
Height	Distancia (número de nodos) desde la definición de la clase hasta el nodo del módulo en el que está	Integer
Average stmts method body	Número medio de sentencias en el cuerpo de los métodos	Real
Type annotations pct	Proporción de los métodos y de los parámetros de los métodos con anotación de tipos	[0, 1]
Private methods pct	Proporción de los métodos que son privados	[0, 1]
Magic methods pct	Proporción de los métodos que son magic	[0, 1]
Async methods pct	Proporción de los métodos que son asíncronos	[0, 1]
Class methods pct	Proporción de los métodos que son de clase	[0, 1]
Static methods pct	Proporción de los métodos que son estáticos	[0, 1]
Abstract methods pct	Proporción de los métodos que son abstractos	[0, 1]
Property methods pct	Proporción de los métodos que son de propiedad	[0, 1]

3.2.5 Function Definitions

Tabla 5: Características para definiciones de funciones.

Nombre	Descripción	Dominio
Name convention	La convención utilizada para el nombre de la función.	Lower | Upper | CamelLow | CamelUp | SnakeCase | Discard | NoNameConvention
Number of characters	Número de caracteres del nombre de la función	Integer
Is private	Si la función es privada o no	True or False
Is magic	Si la función es mágica o no	True or False
Is async	Si la función es asíncrona o no	True or False

Nombre	Descripción	Dominio
Height	Distancia (número de nodos) desde la definición de función hasta la raíz del módulo	Integer
Body count	Número de sentencias en el cuerpo de la función	Integer
Expressions pct	Proporción de las sentencias en el cuerpo de la función que son expresiones	[0, 1]
Number of decorators	Número de decoradores que tiene la función	Integer
Has return type annotation	Si la función tiene anotación del tipo que devuelve	True or False
Has doc string	Si la función tiene un comentario de función. Esto es si su primer hijo es una cadena	True or False
Type annotations pct	Proporción de los parámetros que tiene anotación de tipo. Incluido el tipo de retorno	[0, 1]

La Tabla 5 muestra la información almacenada para cada una de las definiciones de funciones. Guardamos la convención y el número de caracteres del nombre de la función. Muchas de las características que guardamos en esta tabla coinciden con las almacenadas para las definiciones de clases, por ejemplo, la proporción de expresiones en el cuerpo de la función, el número de decoradores, la proporción de anotaciones de tipos y el código fuente.

Para las funciones hemos identificado tres tipos: funciones privadas, funciones asíncronas y funciones mágicas.

3.2.6 Method Definitions

Tabla 6: Características de definiciones de métodos.

Nombre	Descripción	Dominio
Is class method	Si el método es de clase. Esto es si tiene el decorador `@classmethod`	True or False
Is static method	Si el método es estático. Esto es si tiene el decorador `@staticmethod`	True or False
Is constructor method	Si el método es un constructor. Esto es si se llama __init__	True or False
Is abstract method	Si el método es abstracto. Esto es si tiene el decorador `@abstract`	True or False

Nombre	Descripción	Dominio
Is property	Si el método es de propiedad. Esto es si tiene el decorador `@property`	True or False
Is cached	Si el método es cacheado. Esto es si tiene el decorador `@cache`	True or False
Is wrapper	Si el método es un *wrapper*. Esto es si tiene el decorador `@wraps`	True or False

La Tabla 6 muestra la información adicional que complementa la tabla de definiciones de funciones para aquellas funciones que han sido definidas dentro de una clase (métodos). En esta tabla la única información que vamos a almacenar es la respectiva al tipo de método. Tendremos en cuenta los tipos de métodos ya descritos en el apartado de las definiciones de clases (estático, abstracto, de propiedad y de clase) y, añadiremos tres tipos más: constructor (si el nombre del método es __init__), cacheado (si tiene el decorador `@cache`) y *wrapper* (si tiene el decorador `@wraps`).

3.2.7 Statements

En esta tabla vamos a agrupar todas las sentencias (teniendo un campo para distinguir los diferentes tipos) exceptuando las definiciones de funciones que hemos mostrado en los apartados anteriores y las expresiones que mostraremos en apartados posteriores.

Las sentencias van a tener tres campos de especial importancia: la categoría, que representa la categoría sintáctica de la sentencia; el rol de sentencia, que representa el rol que cumple la sentencia en su padre; y, la categoría del padre. Para estos tres campos definimos el dominio de posibles valores que pueden tomar (Anexo 9.1).

Además de esos tres campos, vamos a guardar su profundidad y su altura, las categorías sintácticas de sus tres primeros hijos y, solo para algunos tipos de sentencia (`If`, `While`, `With`, …), el número de sentencias de su cuerpo y si tienen una cláusula `else`.

Tabla 7: Características de sentencias.

Nombre	Descripción	Dominio
Category	Categoría sintáctica de la sentencia	StatementCategory*
Parent	Categoría sintáctica del nodo padre	Module | ClassDef | FunctionDef | MethodDef | StatementCategory*
Statement role	Rol que cumple la sentencia en el nodo padre	StatementRole*
Depth	Distancia máxima desde la sentencia hasta un nodo hoja	Integer
Height	Distancia desde el nodo actual hasta el nodo raíz.	Integer
Has or else	Si la sentencia tiene una cláusula	True, False, N/A

Nombre	Descripción	Dominio
	`else`. Solo aplicable para `Try`, `TryStar`, `If`, `For`, `AsyncFor` y `While`. N/A en otro caso.	
Body size	Número de sentencias en el cuerpo de la sentencia. Solo aplicable para `While`, `If`, `For`, `AsyncFor`, `Try`, `TryStar`, `With` y `AsyncWith`. N/A en otro caso.	True, False, N/A
First, second and third child	Categoría sintáctica del primer, segundo y tercer hijo de la sentencia respectivamente	Parameter| ExpressionCategory*

* Dominio definido en el Anexo 9.1.

3.2.8 Cases

Tabla 8: Características de los cases.

Nombre	Descripción	Dominio
Number of cases	Número de cases en la sentencia `Match`	Integer
Guards	Proporción de *guards* en función del número de cases.	[0, 1]
Average body count	Número medio de sentencias en el cuerpo de los *cases*	Real
Average match value	Número medio de cláusulas `MatchValue` dentro de los *cases* de la sentencia `Match`	Real
Average match singleton	Número medio de cláusulas `MatchSingleton` dentro de los *cases* de la sentencia `Match`	Real
Average match sequence	Número medio de cláusulas `MatchSequence` dentro de los *cases* de la sentencia `Match`	Real
Average match mapping	Número medio de cláusulas `MatchMapping` dentro de los *cases* de la sentencia `Match`	Real
Average match class	Número medio de cláusulas `MatchClass` dentro de los *cases* de la sentencia `Match`	Real
Average match star	Número medio de cláusulas `MatchStar` dentro de los *cases* de la sentencia `Match`	Real
Average match as	Número medio de cláusulas	Real

Nombre	Descripción	Dominio
	`MatchAs` dentro de los *cases* de la sentencia `Match`	
Average match or	Número medio de cláusulas `MatchOr` dentro de los *cases* de la sentencia `Match`	Real

En esta tabla almacenaremos información adicional a la tabla de sentencias para las sentencias `Match`. En este caso vamos a guardar el número medio de cada uno de los 8 tipos de cláusulas (`MatchValue`, `MatchSingleton`, `MatchSequence`, `MatchMapping`, `MatchClass`, `MatchStar`, `MatchAs` y `MatchOr`), además del número medio de sentencias en los cuerpos de las cláusulas *case*, el número de *cases* total y la proporción de *guards* en relación con total de *cases*.

3.2.9 Handlers

Tabla 9: Características de los handlers.

Nombre	Descripción	Dominio
Number of handlers	Número de cláusulas `except`	Integer
Has finally	Si la sentencia `Try` tiene una cláusula `Finally`	True or False
Has catch all	Si la sentencia `Try` tiene una cláusula `except` que capture todas las excepciones (`type==None`)	True or False
Average body count	Número medio de sentencias en el cuerpo de las cláusulas `except`.	Real
Has star	Si incluye una cláusula `except` con estrella (`TryStar`)	True or False

En esta tabla vamos a almacenar información adicional a la tabla de sentencias para las sentencias `Try` y `TryStar`, concretamente en relación con el contenido y forma de las cláusulas `except`. Guardaremos: el número de *handlers* (cláusulas `except`); si el *try* contiene la cláusula `finally`; si el *try* contiene un *catch all* (`except` que admite cualquier tipo de excepción); el número medio de sentencias en el cuerpo de los *except*s y, si algún *handler* incluye el operador estrella (si la sentencia es un `TryStar`, contiene un `except**`).

3.2.10 Expressions

Tabla 10: Características de expression.

Nombre	Descripción	Dominio
Category	Categoría sintáctica del nodo.	ExpressionCategory*

Nombre	Descripción	Dominio
First, second, third and fourth child	Categoría sintáctica del hijo correspondiente.	ExpressionCategory*
Parent	Categoría sintáctica del nodo padre.	Module | ClassDef | FuncionDef | MethodDef | StatementCategory* | ExpressionCategory*
Expression role	Rol del nodo actual en el nodo padre.	ExpressionRole*
Height	Distancia en arcos desde el nodo actual hasta el nodo raíz.	Integer
Depth	Distancia en arcos desde el nodo actual hasta el nodo hoja más distante.	Integer

* Dominio definido en el Anexo 9.1.

La Tabla 10 muestra la información almacenada para las expresiones. Almacenamos una información similar a la que almacenamos para las sentencias. En este caso, además de la categoría sintáctica de la expresión y de su nodo padre, almacenamos las categorías sintácticas de sus cuatro primeros hijos. También guardamos información sobre el papel que desempeña en su padre (rol), junto con los valores de la altura y la profundidad.

En los apartados siguientes vamos a comentar las diferentes tablas adicionales utilizadas para completar la información específica de algunas expresiones.

3.2.11 Comprehensions

Tabla 11: Características de comprehensiones.

Nombre	Descripción	Dominio
Category	Categoría sintáctica de la *comprehension*	ListComp | SetComp | DictComp | GenComp
Number of ifs	Número de condiciones de la *comprehension* (`Ifs`)	Integer
Number of generators	Número de generadores en la *comprehension*	Integer
Is async	Si la *comprehension* es asíncrona	True or False

Esta es una tabla específica de expresiones, contiene la información relativa a las *comprehensions*. Esto son las expresiones de las siguientes categorías sintácticas: ListComp, SetComp, DictComp y GenComp. En esta tabla, a parte del tipo de *comprehension* que es, almacenaremos el número de generadores y el número de condiciones (`Ifs`) de la *comprehension*. Además, guardaremos si la *comprehension* es asíncrona.

3.2.12 CallArgs

Tabla 12: Características de las invocaciones a funciones.

Nombre	Descripción	Dominio
Number args	Número de argumentos en la invocación a la función	Integer
Named args pct	Proporción de argumentos pasados por referencia	[0, 1]
Double star args pct	Proporción de argumentos con la sintaxis **args	[0, 1]

Esta tabla almacena la información extra necesaria para las expresiones con categoría sintáctica `Call` (invocación a función). Esta información extra está formada por: número de argumentos; proporción de argumento parados por referencia (`arg_name=arg_value`) y, proporción de argumentos con la sintaxis "`**arg`".

3.2.13 FStrings

La información relativa a las cadenas de texto formateadas (expresiones de la categoría `JoinedStr`) la guardaremos en esta tabla. Esta información será el número de elementos de la cadena formateada (constantes más valores formateados), la proporción de constantes en los elementos y la proporción de expresiones en los elementos.

Tabla 13: Características de las cadenas formateadas.

Nombre	Descripción	Dominio
Number of elements	Número de elementos en la cadena formateada. Tanto constantes como valores formateados.	Integer
Constants pct	Proporción de los elementos anteriormente mencionados que son constantes.	[0, 1]
Expressions pct	Proporción de los elementos anteriormente mencionados que son expresiones (`FormattedValues`)	[0, 1]

3.2.14 Variables

En este caso, esta tabla contiene la información relativa a las variables. La información almacenada es: la convención de nombrado que sigue la variable; el número de caracteres del nombre; si la variable es privada, en función de su nombrado y, si la variable es mágica, en función también de su nombrado.

Tabla 14: Características de las variables.

Nombre	Descripción	Dominio
Name convention	Convención de nombrado que sigue la variable.	Lower | Upper | CamelLow | CamelUp | SnakeCase | Discard | NoNameConvention
Number of characters	Número de caracteres del nombre de la variable	Integer
Is private	Si la variable es privada. Esto es si su nombre comienza por _	True or False
Is magic	Si la variable es mágica. Esto es si su nombre es de la forma __name__	True or False

3.2.15 Vectors

Esta es la última tabla del conjunto de tablas de expresiones. En esta recogeremos la información necesaria para los vectores. Los vectores son todas las expresiones que pertenezcan a una de las siguientes categorías sintácticas: `ListLiteral`, `SetLiteral`, `DictLiteral` y `GeneratorLiteral`. La información almacenada será: la categoría del vector, para diferenciar entre las 4 posibles; el número de elementos que componen el vector y, si el vector es homogéneo, esto es si todos los elementos que lo componen son del mismo tipo.

Tabla 15: Características de los vectores.

Nombre	Descripción	Dominio
Category	Categoría sintáctica del vector	ListLiteral | SetLiteral | DictLiteral | GeneratorLiteral
Number of elements	Número de elementos que componen el vector	Integer
Homogeneous	Si todos los elementos del vector son del mismo tipo	True or False

3.2.16 Parameters

En esta tabla, almacenaremos la información relativa a los parámetros declarados tanto en la definición de funciones, métodos y constructores, como en las lambda expresiones. Para distinguir entre los parámetros que provienen de cada una de las posibilidades vamos a almacenar su rol. Además, vamos a almacenar la proporción de los parámetros que son de cada posible tipo: parámetros posicionales; parámetros variables; parámetros únicamente con *keyword* y parámetros con valor por defecto. Para algunos de estos tipos incluiremos un valor booleano para saber si tienen al menos un parámetro de ese tipo. Por último, incluiremos el convenio de nombrado más habitual entre los nombres de los parámetros y la proporción de parámetros con anotación de tipo.

Tabla 16: Características de los parámetros.

Nombre	Descripción	Dominio
Parameters role	Indica el nodo en el que se definieron los parámetros	FunctionParameters | LambdaParameters
Number of params	Número de parámetros	Integer
Por only param pct	Proporción de parámetros posicionales.	[0, 1]
Var param pct	Proporción de parámetros como variable	[0, 1]
Has var param	Si hay algún parámetro del tipo variable	True or False
Type annotation pct	Proporción de los parámetros que tienen una anotación de tipo	[0, 1]
Kw only param pct	Proporción de los parámetros que solo se pueden pasar por referencia	[0, 1]
Default value pct	Proporción de los parámetros que definen un valor por defecto	[0, 1]
Has kw param	Si hay algún parámetro del tipo keyword	True or False
Name convention	Convención de nombrado más seguida por los nombres de los parámetros	Lower | Upper | CamelLow | CamelUp | SnakeCase | Discard | NoNameConvention

Las 16 tablas creadas contienen información sobre construcciones sintácticas homogéneas, permitiéndonos obtener patrones comunes de definiciones, sentencias o expresiones. Sin embargo, un programa se compone de distintas construcciones, representado mediante información heterogénea (p. ej., un programa tiene un módulo, que tiene una clase, que define un método, donde se usa una sentencia que incluye a su vez varias expresiones distintas). Por ello, además de utilizar la información homogénea, debemos ofrecer la posibilidad de obtener información heterogénea y así poder trabajar con patrones sintácticos más expresivos y heterogéneos, formados por características de varias construcciones sintácticas. La creación de conjuntos de datos heterogéneos se puede realizar mediante operaciones de disgregación de datos (*drill down*) que combinan la información de dos o más tablas [16]. Así, conseguimos relacionar características de diferentes construcciones sintácticas como programas, definiciones, sentencias y expresiones. Como hemos comentado anteriormente, la tabla *nodes* contiene los identificadores de todos los nodos y sus correspondientes padres. De este modo, mediante la utilización de las sentencias SQL `where` y `join` es posible unir los datos de las diferentes tablas y obtener conjuntos de datos heterogéneos, tal y como se realizó en trabajos previos para el lenguaje Java [17].

En resumen, nuestro sistema crea 16 conjuntos de datos homogéneos para el análisis y explotación de construcciones sintácticas, pero también ofrece la posibilidad de crear conjuntos de datos heterogéneos para su futuro análisis y explotación (ver Trabajo Futuro 6.2).

3.3 Generación de informes

Como se ha indicado, la información generada puede utilizarse para diversos escenarios de minería de datos y aprendizaje automático. A modo de ejemplo, en este trabajo nos limitaremos a hacer los siguientes informes.

El primer análisis es un estudio de valores anómalos o *outliers*. Realizamos análisis univariable de cada una de las características de los 16 conjuntos de datos, así como análisis multivariante de cada una de las entidades homogéneas (no los realizamos para información heterogénea). El detalle de la metodología utilizada se describe en la Sección 4.2. Así mismo, para cada característica de las entidades homogéneas realizamos un análisis de frecuencia (Sección 4.3) y de correlación con el nivel de experiencia del programador (Sección 4.4), para ver si existen diferencias significativas en función de su veteranía.

Los análisis se reportan mediante *Jupyter Notebooks* [18]. Para cada una de las 16 tablas se desarrollaron tres análisis: uno con todos los datos, otro con los datos de los alumnos y, finalmente, otro con los datos de expertos. En cada uno de ellos se analizaron todas las variables de la tabla correspondiente, descritas en la Sección 3.2, siguiendo los análisis descritos en el párrafo anterior. De este modo, se creó una colección de 48 *notebooks* con todos los análisis llevados a cabo. El Anexo 9.2 contiene todos los resultados obtenidos durante el análisis, y en el Capítulo 5 se resaltan los resultados más interesantes. Todos los *notebooks* implementados están disponibles para su consulta en [19] e incluidos como Anexo 9.4.

4 Metodología

4.1 Conjunto de Datos

El conjunto de datos utilizado incluye programas escritos tanto por programadores principiantes como por expertos, sumando un total de 13.94 millones de nodos de ASTs (Tabla 17). Los programas de principiantes han sido obtenidos de estudiantes de primer año del Grado en Ingeniería de Software de la Universidad de Oviedo, durante el periodo 2020-2024, de la asignatura de Fundamentos de la Informática. Cada programa representa un examen o una práctica realizada por un alumno o por un grupo de alumnos.

El código fuente de los programas escritos por expertos fue obtenido utilizando la *API* de *GitHub* [20], que permite acceder a los repositorios públicos con licencia *open source* más relevantes. Aunque los detalles exactos del algoritmo utilizado por la *API* de GitHub para devolver los repositorios no están públicamente documentados, se sabe que tiene en cuenta factores como la fecha de última actividad, la relevancia del título y la descripción, así como el número de estrellas, contribuidores y *forks* para determinar la relevancia de los repositorios. Dado que en este proyecto nos enfocamos exclusivamente en proyectos escritos en Python, utilizamos la *API* de *GitHub* para filtrar y obtener únicamente repositorios que estuvieran etiquetados como proyectos escritos en este lenguaje de programación. Una vez obtenida la lista de resultados (en el Anexo 9.3 está disponible la lista completa), utilizamos *Git* para clonar estos repositorios y obtener así el código fuente completo. Este proceso nos ha permitido analizar grandes volúmenes de código Python provenientes de proyectos reales y diversos.

Finalmente, el conjunto de datos utilizado en este trabajo está formado por 1.609 programas con 18.226 ficheros Python.

Tabla 17: Número de Nodos de los ASTs

	PRINCIPIANTE	EXPERTO	TOTAL
PROGRAMS	1.591	18	1.609
MODULES	7.124	11.102	18.226
IMPORTS	7.124	11.102	18.226
CLASSDEFS	7.755	14.569	22.324
FUNCTIONDEFS	59.879	88.089	147.968
METHODDEFS	35.666	56.754	92.420
PARAMETERS	61.752	92.146	153.898
EXPRESSIONS	3.490.521	4.792.486	8.283.007
CALLARGS	338.727	521.668	860.395
COMPREHENSIONS	4.982	13.277	18.259
FSTRINGS	4.977	17.053	22.030
VARIABLES	1.167.967	1.694.338	2.862.305
VECTORS	121.764	229.900	351.664
STATEMENTS	457.483	617.277	1.074.760
CASES	0	29	29
HANDLERS	8.953	6.803	15.756
TOTAL			**13.942.876**

En la Tabla 17, podemos observar que la distribución de los datos para cada clase de programador se encuentra relativamente equilibrada, excepto en el caso de los programas. Esto se debe a que los proyectos expertos son de mayor entidad y contienen un mayor número de archivos. El habrer añadido más proyectos de desarrolladores expertos hubiese desequilibrado el resto de las tablas. En la construcción del conjunto de datos, se buscaba lograr un mayor equilibrio en las demás construcciones sintácticas, independientemente del número de programas, al que daremos menos importancia en los informes realizados a la hora de considerar el nivel de experiencia del programador.

A partir del código fuente, se generan los ASTs y se recorren éstos para para rellenar las 16 tablas definidas en la Sección 3.2. La Tabla 17 muestra el número de nodos AST obtenidos para cada una de las tablas, observándose cómo la base de datos tiene un total de casi 14 millones de entradas.

4.2 Análisis de anomalías

Los datos utilizados pueden contener instancias con valores atípicos, representando construcciones sintácticas significativamente distintas al resto de la población. La identificación y análisis de anomalías (*outliers*) es importante para ofrecer una información valiosa y precisa del conjunto de datos obtenido en este trabajo. Adicionalmente, determinadas anomalías pueden deberse a errores en los datos, que deben ser eliminadas para evitar conclusiones erróneas [21].

Nuestros conjuntos de datos poseen información tanto numérica como categórica. Para detectar valores anómalos univariables en los datos numéricos, empleamos el test de Tukey [22]. Este test se basa en comparaciones relativas al rango intercuartil (IQR), definido como la diferencia entre el tercer y primer cuartil de una distribución ($IQR = Q_3 - Q_1$), donde Q_n representa el cuartil n.

De este modo, se define un valor atípico como aquél fuera de alguno de los dos siguientes intervalos:

$$[Q_1 - 1{,}5 \times IQR,\ Q_3 + 1{,}5 \times IQR] \quad (1)$$
$$[Q_1 - 3 \times IQR,\ Q_3 + 3 \times IQR] \quad (2)$$

El primer rango permite identificar un valor atípico leve y el segundo uno extremo. En base a la distribución de los valores de la variable, consideraremos (1) o (2) como rangos para la detección de anomalías, eligiendo (2) siempre que haya valores atípicos extremos y (1) en caso contrario.

Los límites de Tukey se basan en los cuartiles de los datos y son sensibles a la presencia de sesgo en la distribución. Cuando hay asimetría en los datos, los límites de Tukey pueden no ser tan efectivos para identificar *outliers* de manera equitativa en ambos extremos de la distribución. Los métodos *Adjusted Box Plots* están diseñados para describir distribuciones sesgadas y se basan en medidas de asimetría [23]. El Coeficiente de Medcouple (*Medcouple Coefficient*, *MC*) [24] es útil para identificar la asimetría en los datos, especialmente en presencia de valores atípicos o sesgados. Es una medida robusta porque no se ve tan afectada por valores extremos como la media y la desviación típica. El *MC* puede proporcionar información adicional sobre la asimetría de la distribución, lo que te permite ajustar los límites de Tukey de manera más apropiada para una distribución de datos específica. En [24] proponen el siguiente método para ajustar los límites de Tukey en función del *MC*:

$$[Q_1 - 1{,}5 \times e^{-4\times MC} \times IQR,\ Q_3 + 1{,}5 \times e^{3{,}5\times MC} \times IQR] \quad (3)$$
$$[Q_1 - 1{,}5 \times e^{-3{,}5\times MC} \times IQR,\ Q_3 + 1{,}5 \times e^{4\times MC} \times IQR] \quad (4)$$

En este caso, utilizamos diferentes rangos según la dirección de la asimetría en los datos. Si el coeficiente de asimetría *MC* es mayor que 0, lo que indica una asimetría hacia la derecha, utilizamos el primer rango (3) para identificar valores atípicos. Si la asimetría es hacia la izquierda (MC menor que 0), empleamos el segundo rango (4). Cuando aplicamos el test de Tukey y detectamos valores atípicos extremos en el rango (2), utilizamos el rango (3) o (4), según la dirección de la asimetría, para verificar si el número de valores atípicos obtenido con este rango es menor. Si este es el caso, dicho rango será el utilizado para considerar los valores como anómalos.

Por ejemplo, la característica *Number of modules* de *Program* (ver Tabla 1) toma valores en el rango [1, 3.294], con un valor medio de 11,3 módulos por programa. Al aplicar las fórmulas de cálculo de intervalos (1) y (2), se obtienen los rangos [-2, 6] para valores atípicos leves y [-5, 9] para valores extremos. De esta forma, se identifican 64 (3,97%) y 33 (2,05%) instancias anómalas, respectivamente, por estar fuera de los rangos obtenidos.

No obstante, el coeficiente de asimetría MC de los valores de esta variable es 0,97. Este valor indica una fuerte asimetría positiva, lo que significa que la distribución presenta una cola larga en su extremo derecho. En este caso, al tratarse de una asimetría hacia la derecha, aplicamos la fórmula (3) y obtenemos el nuevo rango [0, 143], con el cual se reducen a 17 (1,05%) las instancias identificadas como anómalas.

Para la detección de anomalías multivariante, se utilizó el algoritmo de *Isolation Forest*. Este algoritmo identifica los valores atípicos teniendo en cuenta lo lejos que está un punto de datos (instancia) del resto de los datos [25]. El hiperparámetro de contaminación especifica la proporción de valores atípicos en el conjunto de datos. En nuestro caso encontramos que el valor 0,01 (1%) es el factor de contaminación que mejor identifica los valores atípicos en nuestros conjuntos de datos.

Para el caso de las variables categóricas, empleamos un valor umbral del análisis de frecuencia en el que consideramos que un valor categórico es atípico cuando su número de ocurrencias es menor que 0,2% / *número posibles valores*. Por ejemplo, una variable booleana tendrá un valor anómalo cuando éste ocurra menos del 0,1% del total.

4.3 Análisis de frecuencia

Se realiza un análisis de frecuencia para cada una de las variables de las 16 entidades. Una con todos los datos del conjunto de datos, otra para los programadores expertos y una tercera para los novatos. Si las variables son categóricas, nos limitamos a emplear la frecuencia de cada valor. Para las variables numéricas creamos tantos intervalos (*bins*) como el valor menor obtenido con los métodos de Sturges (Ecuación 5) y el de Freedman-Diaconis (Ecuación 6)—funcionamiento utilizado por la función `hist` de `matplotlib`.

$$\text{Número de intervalos} = \lceil \log_2(n)+1 \rceil \quad (5)$$

$$\text{Ancho del intervalo} = 2\ \frac{IQR}{n^{1/3}} \quad (6)$$

La Ecuación (5) identifica un número de intervalos, denotando ⌈ ⌉ el redondeo hacia arriba del número entero más cercano (techo o *ceiling*). El resultado devuelve el número de intervalos a calcular, que son todos del mismo tamaño.

Para Freedman-Diaconis (Ecuación 6), IQR denota el rango intercuartil, definido como la diferencia entre el tercer y primer cuartil de una distribución. *n* representa el número de observaciones del conjunto de datos. El resultado es el valor que tiene cada intervalo, teniendo todos el mismo.

4.4 Correlación con el nivel de experiencia

Para las variables numéricas de cada una de las construcciones sintácticas, se realiza un análisis de correlación con el nivel de experiencia de usuario. En el caso de obtener diferencias significativas, documentamos las diferencias que dicha variable obtiene en función del nivel del programador.

Para las variables categóricas, estudiamos si las frecuencias obtenidas para el total de las muestras distan de aquellas producidas para los programadores noveles y expertos, discutiendo las diferencias.

4.5 Entorno de ejecución

El sistema presentado en este trabajo se ha desarrollado usando Python 3.12.3 y la correspondiente versión del módulo *Abstract Syntax Trees* – ast de la *Python Standard Library* [9]. Para la detección y análisis de *outliers* se utilizaron los paquetes Pandas 2.2.2, NumPy 1.26.4, statsmodels 0.14.2 y Jupyter 1.0.0 entre otros. Todo el código fuente desarrollado está disponible en *GitHub* para su consulta [19]. Para almacenar los *datasets* utilizamos el sistema de bases de datos de código abierto PostgreSQL 14.11. Todo el código fue ejecutado en un servidor Dell PowerEdge R540 con dos procesadores Intel Xeon Silver 4210R 2.4GHz (40 núcleos) con 160GB DDR4 3.200 MHz de memoria RAM, ejecutando un sistema operativo Ubuntu Server 22.04.1 de 64 bits.

El espacio utilizado en disco para almacenar todo el código fuente es de 5,18 Gb. El tiempo empleado por el sistema para procesar y crear el conjunto de datos es de aproximadamente 2 horas. La base de datos creada en PostgreSQL ocupa 3,27 Gb de espacio en disco.

5 Evaluación

Nuestro conjunto de datos consta de un total de 13.942.876 nodos extraídos de los ASTs generados a partir del conjunto de programas seleccionados. Todos estos nodos pertenecen a una de las 16 tablas definidas (en la Tabla 17 se puede ver el desglose de los datos).

Siguiendo el método descrito en la Sección 4.2, hemos realizado el análisis de valores anómalos, frecuencias y correlación. En cada una de las siguientes subsecciones resumimos los resultados más destacables. Los análisis más detallados se puede consultar el Anexo 9.2. y en los Anexos 9.4.1 - 9.4.16.

5.1 Programs

5.1.1 Número de módulos (number of modules)

Los programas con más de 143 módulos han sido detectados como anómalos. En este análisis, identificamos dos programas con más de 3.000 módulos: *ray* y *llama_index*, ambos con más de 900 contribuidores (Anexo 9.3). Además, entre los programas de alumnos, detectamos seis programas que tienen más de 500 módulos. Esto se debe a que en la entrega incluyeron librerías completas de Python, y en todos estos casos, estos datos no fueron incluidos en el conjunto de datos final.

5.1.2 Número de paquetes (number of packages)

En el caso de los principiantes, cualquier programa que tenga al menos un paquete se considera anómalo (el 0% tiene solo uno y el 0,37% más de uno). Sin embargo, para los expertos, solo hemos detectado como anómalos los programas con más de 172 paquetes.

5.1.3 Porcentaje de definiciones de clases (class defs pct)

En el caso concreto de las definiciones de clases, mientras que los programas de expertos siempre tienen alguna definición de clase, entre los programas de principiantes solo el 2% tiene al menos una. De este modo, hemos detectado como anómalo cualquier programa de alumno que contenga alguna definición de clase. En el caso de los expertos, solo identificamos como anómalos aquellos programas donde al menos el 82% de las definiciones sean de clases.

5.1.4 Porcentaje de definiciones de funciones (function defs pct)

Respecto a las definiciones de funciones, hemos identificado como anómalos los programas escritos por expertos donde las funciones representen menos del 42% de las definiciones. En este caso, no hemos detectado valores anómalos en los programas de principiantes.

5.1.5 Porcentaje de definiciones de enumerados (enum defs pct)

Hemos detectaron como anómalos los programas que tienen una o más enumeraciones, debido a que tan solo el 0,7% de los programas las utilizan. Además, todos los programas con enumeraciones fueron escritos por expertos. En lugar de utilizar enumeraciones, los programadores suelen utilizar constantes para definir valores fijos, principalmente debido a su simplicidad.

5.1.6 Análisis multivariante

El análisis multivariante también identificó como anómalos los programas *ray* y *llama_index*, con medias de 3.286 módulos, 153 subdirectorios con código y 839 paquetes.

5.1.7 Otros

Hemos apreciado cómo 544 programas (cerca del 34%) de alumnos no tienen definiciones (clases, funciones o enumeraciones) en comparación con el 0% de los expertos. Esto se debe a que, al inicio

de la asignatura, el enfoque está orientado a enseñar a los alumnos el manejo de bucles, sentencias condicionales y expresiones, dejando las definiciones de funciones y clases para más adelante. Por esta razón, cualquier programa que contenga alguna definición de clase o enumeración se ha detectado como anómalo en los estudiantes.

En la siguiente figura, se puede observar la estrecha correlación entre los programadores expertos y presencia de paquetes, la definición de enumerados y el número de paquetes. Así mismo, se puede observar una relación fuerte entre el nivel de experiencia principiante y la existencia de código en el directorio raíz.

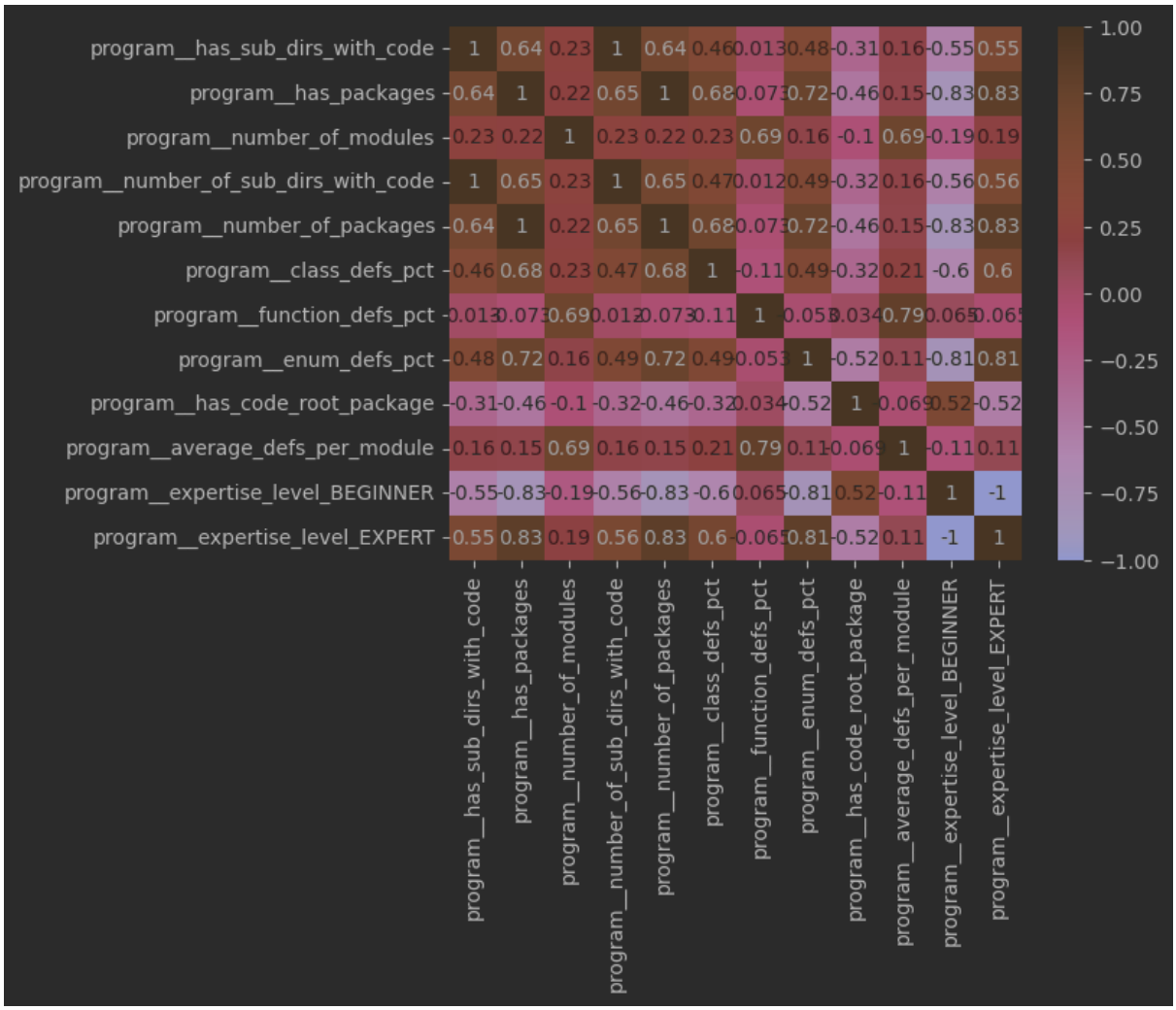

Figura 5: Matriz de correlación de Spearman para la tabla programs

5.2 Modules

Casi el 22% (3.974) de los módulos no tienen definiciones (clases, funciones o enumerados), de los cuales más de la mitad (2.199) son de expertos. Tal y como mencionamos en el apartado anterior, en el caso de los alumnos, esto se debe a que no usan clases o funciones hasta bien avanzado el curso. En el caso de los expertos, la mayoría de estos módulos se utilizan como archivos de configuración, scripts ejecutables y de automatización de tareas.

5.2.1 Número de clases (number of classes)

Hemos detectado como anómalos los módulos que tienen más de 76 clases, identificándose únicamente tres módulos que cumplen esta condición. Cabe destacar qué entre los módulos de expertos, cerca del 50% no tienen ninguna definición de clase. En el caso de los principiantes, este porcentaje es del 70%.

5.2.2 Número de funciones (number of functions)

En lo que a funciones se refiere, hemos identificado como anómalos cuando definen más de 185 funciones. Solamente un módulo, de experto, supera este límite con 285 funciones definidas.

5.2.3 Convenio de nombrado (name convention)

Ninguno de los convenios de nombrado de módulos fue identificado como anómalo. Entre los expertos, el convenio de nombrado más habitual es el `SnakeCase`, con cerca de un 65% de uso. Por el contrario, entre los principiantes el `SnakeCase` solo se usa un 20% de los casos, y es el `Lower` el más usado con un 45% de los módulos.

En la siguiente figura se puede apreciar la distribución de todos los valores en función del convenio de nombrado y el nivel de experiencia:

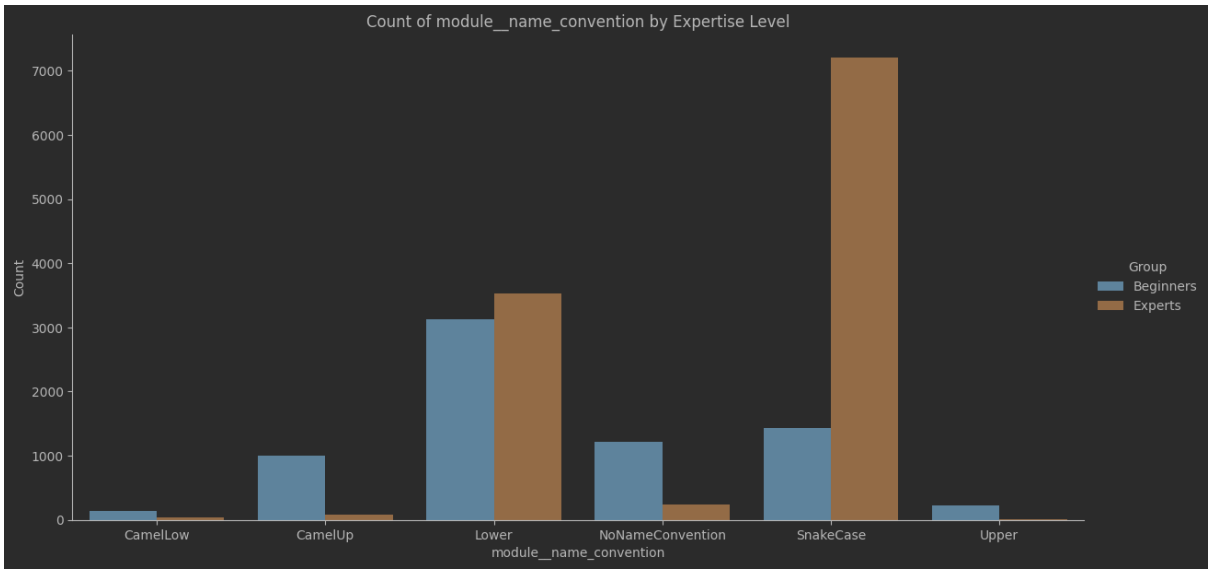

Figura 6: Distribución de valores de la variable name_convention en Modules

5.2.4 Porcentaje de anotaciones de tipo (type annotations pct)

Hemos detectado como anómalo cualquier módulo con un porcentaje de anotaciones de tipo superior al 96%. Teniendo en cuenta solo a los expertos, no hemos identificado anomalías, mientras que, entre los principiantes, una sola anotación ya hace que el módulo sea considerado anómalo. Cabe destacar que, entre los expertos, el 41% de los módulos tienen al menos una anotación de tipo, mientras que entre los principiantes este porcentaje es de apenas el 12%. Además, entre los principiantes no existe ningún módulo con un 100% de anotaciones de tipo.

5.2.5 Análisis multivariante

El análisis multivariante identificó 22 módulos anómalos que tenían una media de 49,4 clases y 33,9 funciones definidas. Además, todos ellos cuentan con un punto de entrada (`has_entry_point`) y no tienen definiciones de enumeraciones. En contraste, los módulos no anómalos tenían una media de 0,98 clases y 2,48 funciones, con y sin puntos de entrada y definiciones de enumeraciones.

5.3 Imports

5.3.1 Número de imports (number of imports)

Hemos identificado como anómalo cuando el número de *imports* es superior a 30 en el caso de los expertos y 20 en el caso de los principiantes. Detectamos 3 módulos de expertos con más de 123 *imports*. Cabe destacar que el 18% de los módulos no tienen ningún *import*.

5.3.2 Media de imports de tipo as (average as in imported modules)

Solo el 6% de los módulos tiene *imports* del tipo `as`, de modo que todos ellos se consideran anómalos.

5.4 Class Definitions

5.4.1 Anotaciones de tipo genéricas (generic type annotations)

Ninguna definición de clase tiene anotaciones de tipo genérica. Esto puede deberse a que dicha funcionalidad fue añadida a Python en la versión 3.12, la más reciente al momento de realizar este trabajo, por lo que aún no era una característica muy conocida entre los programadores.

Las anotaciones de tipo genéricas son útiles para declarar funcionamientos independientemente del objeto que vaya a implementarlo.

```
from collections import queue

class Queue[T]:
   def __init__(self) -> None:
         self.elements: deque[T] = deque()

    def push(self, element: T) -> None:
         self.elements.append(element)

    def pop(self) -> T:
         return self.elements.popleft()
```

En este ejemplo, la clase implementa una funcionalidad de tipo cola para objetos del tipo que se parametrice al instanciar dicha clase.

5.4.2 Número de decoradores

Cualquier definición de clase que tenga un decorador es considerada anómala. Solo el 13% de las definiciones de clase tienen decoradores. En el caso de los principiantes, este porcentaje se reduce hasta apenas un 0,1%, mientras que para los expertos ronda el 20%. En la siguiente figura se pueden apreciar estos datos:

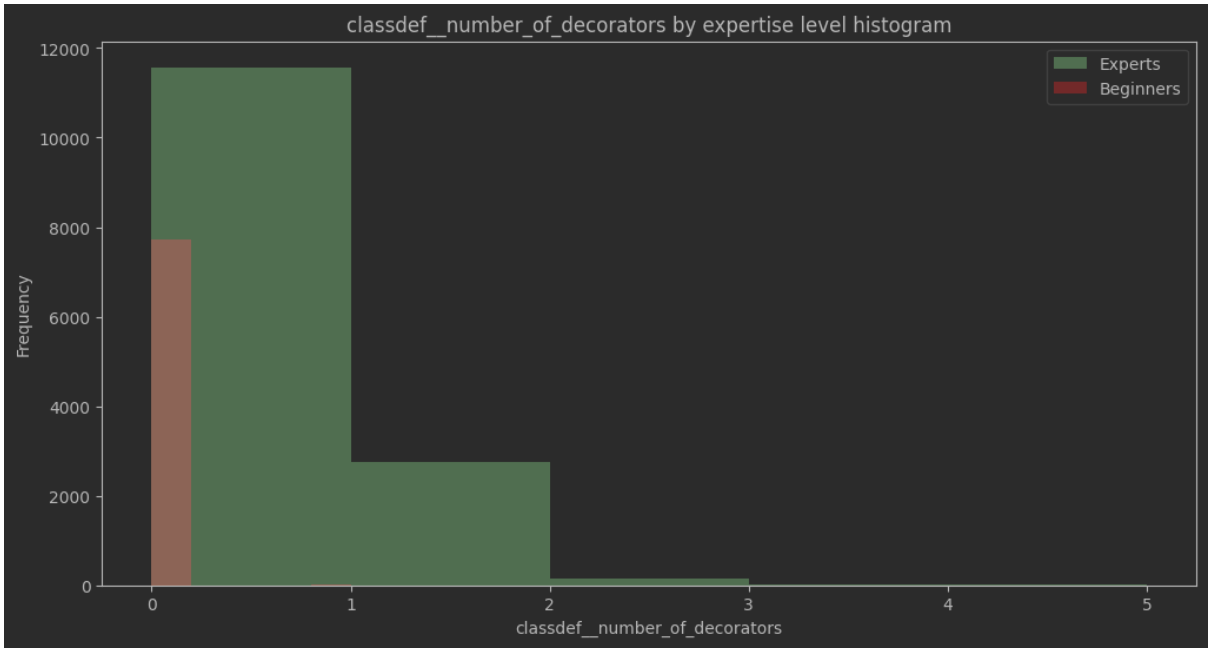

Figura 7: Distribución de valores de la variable number of decorators en Classdefs

5.4.3 Número de métodos (number of methods)

Hemos identificado como anómalas las clases con más de 17 métodos. Cabe destacar que hemos detectado una clase escrita por expertos que contiene más de 200 definiciones de métodos. También se identificaron aproximadamente 4.000 clases (20% del total) que carecen de métodos. Estas clases generalmente consisten en un comentario de clase y una sentencia `Pass`. Por último, es importante destacar que el número medio de métodos por clase es de 4,17.

5.4.4 Número de clases base (number of base classes)

Hemos identificado como anómala cualquier clase que tenga más de una clase base. Esto se debe a que solo el 4,2% de las clases tienen más de una clase base. Entre las clases con más de una clase base, el 70% pertenece a expertos y hay una clase con un total de 45 clases base. Esta última clase implementa parte de una interfaz gráfica y hereda funcionalidad de múltiples clases de *frameworks*.

5.4.5 Porcentaje de anotaciones de tipo (type annotations pct)

Hemos detectado como anómalas las clases cuya proporción de tipos anotados en métodos y funciones es superior al 80%. En el caso de los alumnos, cualquier clase que contenga alguna de estas anotaciones ya es detectada como anómala.

La siguiente figura representa la distribución de los valores en función del nivel de experiencia:

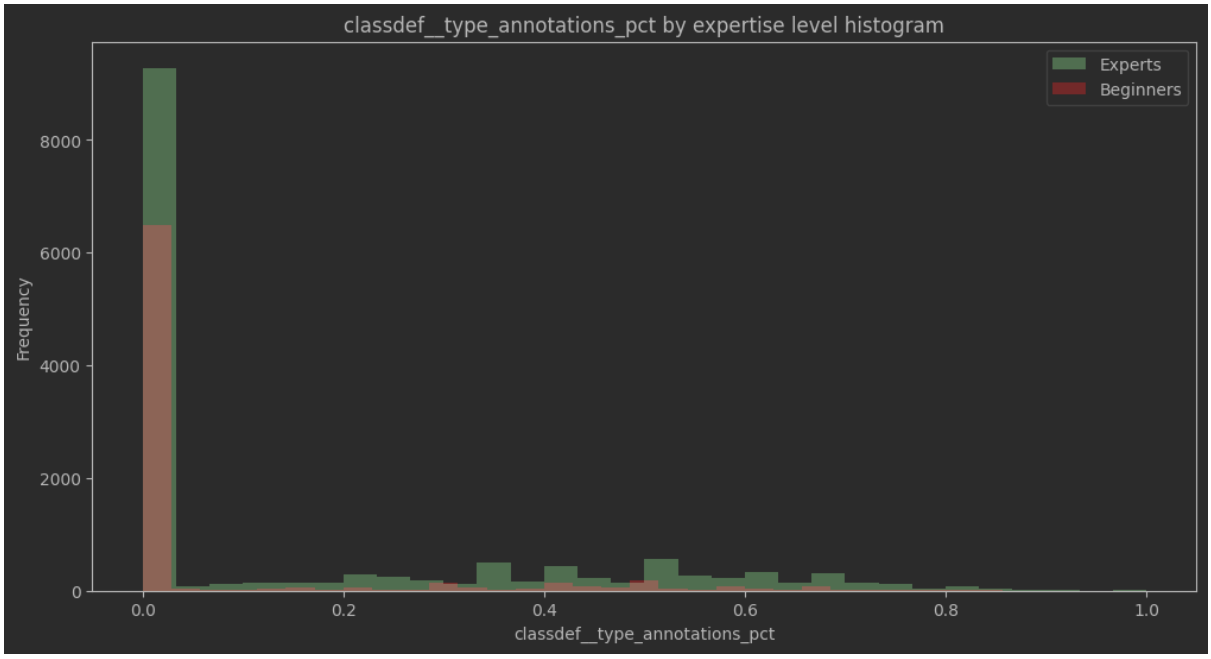

Figura 8: Distribución de valores de la variable type annotations pct en Classdefs

5.4.6 Convenio de nombrado (name convention)

Ningún convenio de nombrado de clases fue identificado como anómalo; en todos los casos, el convenio de nombre más utilizado es `CamelUp`, utilizado por casi el 90% de las clases. Es notable que los principiantes no hacen uso de los convenios de nombrado `CamelLow` o `Discard` cómo se puede apreciar en la siguiente figura:

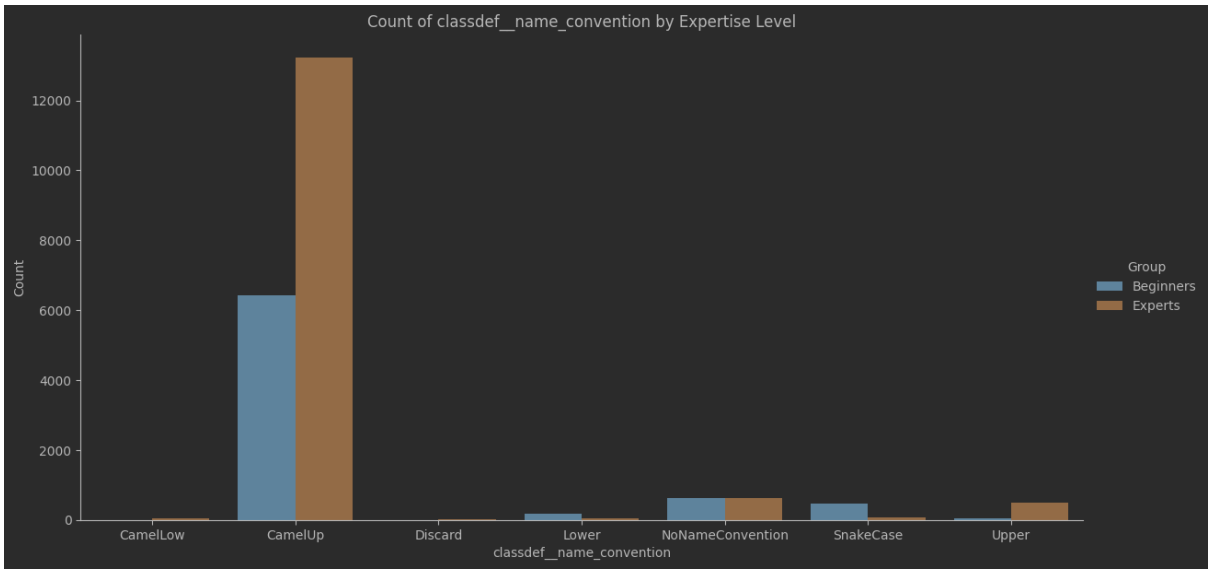

Figura 9: Distribución de valores de la variable name convention en Classdefs

5.4.7 Análisis multivariante

El análisis multivariante ha identificado como anómalas 19 clases con una media de 52 sentencias en su cuerpo, un porcentaje de métodos estáticos cercano al 95% y convenio de nombrado `SnakeCase`.

5.4.8 Otros

Hemos identificado como anómalas aquellas clases que poseen algún método especial, ya sea privado, asíncrono, de clase, estático, abstracto o propiedad, excluyendo los métodos mágicos.

5.5 Function Definitions

5.5.1 Número de decoradores (number of decorators)

Las definiciones de funciones que tienen al menos un decorador hemos detectado como anómalas. Sin embargo, en el conjunto de datos de expertos, solo detectamos como anómalas aquellas definiciones de métodos que tienen más de cuatro decoradores. Hemos detectado una función con un total de 45 decoradores.

5.5.2 Porcentaje de anotaciones de tipo (type annotation pct)

El 73% de las funciones no tienen ninguna anotación de tipo, mientras que cerca del 22% tienen todas las anotaciones de tipo posibles.

Al desglosar los datos por nivel de experiencia, observamos que el 84% de las funciones escritas por principiantes carecen de anotaciones de tipo, y solo el 15% tienen todas las anotaciones de tipo posibles. En contraste, entre los expertos, estos porcentajes son del 64% y 27% respectivamente. Basándonos en estos datos, no encontramos anomalías en las funciones de los expertos, pero hemos detectado como anómala cualquier definición de función escrita por principiantes que incluya anotaciones de tipos en los parámetros o en el retorno. La siguiente figura muestra los datos utilizados para el análisis:

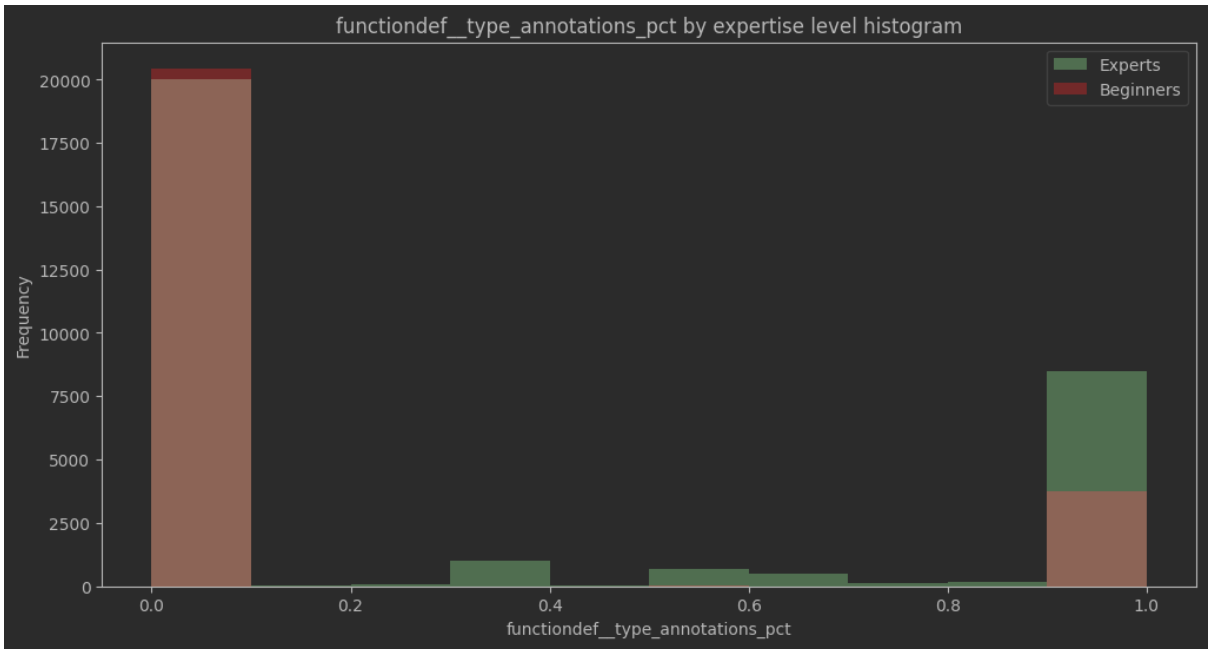

Figura 10: Distribución de valores de la variable type annotations pct en Functiondefs

5.5.3 Altura (height)

Respecto a la altura de las definiciones de funciones, el 83% tienen una altura de 1 (funciones en el ámbito global del módulo) y solo un 7% tienen una altura superior a 2. Por lo tanto, cualquier función con una altura superior a 1 se ha detectado como anómala.

5.5.4 Convenio de nombrado (name convention)

El convenio de nombre más utilizado en todos los casos es `SnakeCase`. En el caso concreto de los principiantes hemos detectado como anómalos los nombrados `Discard` y `Upper`.

5.5.5 Análisis multivariante

El análisis multivariante de los datos de expertos identificó como anómalas 38 funciones con nombres cortos (3,8 caracteres de media frente a 17,7 de media en las demás definiciones), en minúsculas (convención `Lower`), con un alto porcentaje de tipos anotados (97% de media frente a 31%) y cuerpos largos (12,3 sentencias de media frente a 4,9).

5.6 Method Definitions

5.6.1 Contiene anotaciones de tipo de retorno (has return type annotation)

Detectamos como anómalo el uso de decoradores en las definiciones de métodos.

5.6.2 Porcentaje de anotaciones de tipo (type annotations pct)

Respecto a las anotaciones de tipos en los métodos, al igual que en el caso de las funciones, el uso de anotaciones de tipo por parte de los principiantes se detecta como anómalo.

5.6.3 Convenio de nombrado (name convention)

El convenio de nombrado más usado es `SnakeCase`, mientras que `Discard` se ha identificado como un valor anómalo. En el caso de los principiantes, ni siquiera se utiliza este convenio de nombrado.

5.6.4 Métodos asíncronos (is async method)

Hemos detectado como anómalo la definición de métodos asíncronos por parte de los principiantes, no así en el caso de los expertos.

5.6.5 Otros

No hemos encontrado ningún método `wrapper` ni `cached`, ni entre los principiantes ni entre los expertos.

5.6.6 Análisis multivariante

El análisis multivariante ha identificado 111 definiciones de métodos como anómalas. Todas estas definiciones son de métodos abstractos, con el tipo de retorno anotado y con el 100% del cuerpo compuesto por expresiones.

5.7 Statements

5.7.1 Tamaño del cuerpo (body size)

Hemos identificado 52 sentencias como anómalas debido a que contienen más de 49 sentencias o expresiones en su cuerpo. En particular, una de estas sentencias destaca al contener un total de 276 elementos.

5.7.2 Categoría sintáctica (category)

La sentencia más utilizada es `AssignmentStmt`, mientras que las de tipo `Match` y `Nonlocal` se han identificado como anómalas, utilizadas únicamente en el 0,1% y 0,2% de los casos respectivamente. Además, las sentencias de tipo `ExceptHandler` y `TypeAlias` no se usan nunca. Un `TypeAlias` es una estructura Python definida en la versión 3.12 del lenguaje, de ahí que no se utilice. Con esta estructura se pueden definir tipos genéricos.

```
Vector = list[float] # Using the type alias in function annotations
def scale_vector(vector: Vector, scalar: float) -> Vector:
```

Un `ExceptHandler` es la cláusula *except* de una sentencia de tipo **`Try`** o **TryStar**.

En el caso concreto de los principiantes, además de no utilizar las dos anteriores, tampoco hacen uso de `AsyncWith` y `AsyncFor`.

En la siguiente figura se muestran los datos utilizados para el análisis anterior:

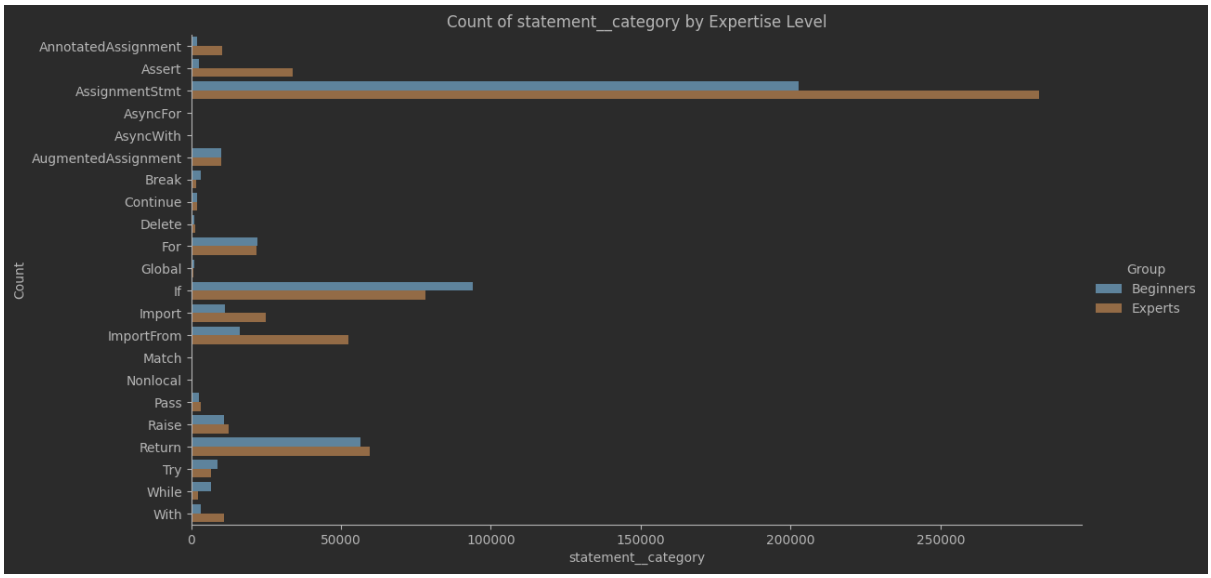

Figura 11: Distribución de valores de la variable category en Statements

5.7.3 Rol de sentencia (statement role)

Ninguna sentencia cumple el rol de `TryHandler`, `AsyncForElseBody` o `TryHandlerStar`, y el rol `WhileElseBody` fue identificado como anómalo. Considerando solo los datos de expertos, el rol `ForElseBody` también hemos detectado como anómalo. Los principiantes tampoco tienen sentencias que cumplan los roles de `AsyncForBody`, `CaseBody` o `AsyncWithBody`, y el rol `AsyncMethodDefBody` es el único identificado como anómalo. La siguiente figura muestra los datos referidos:

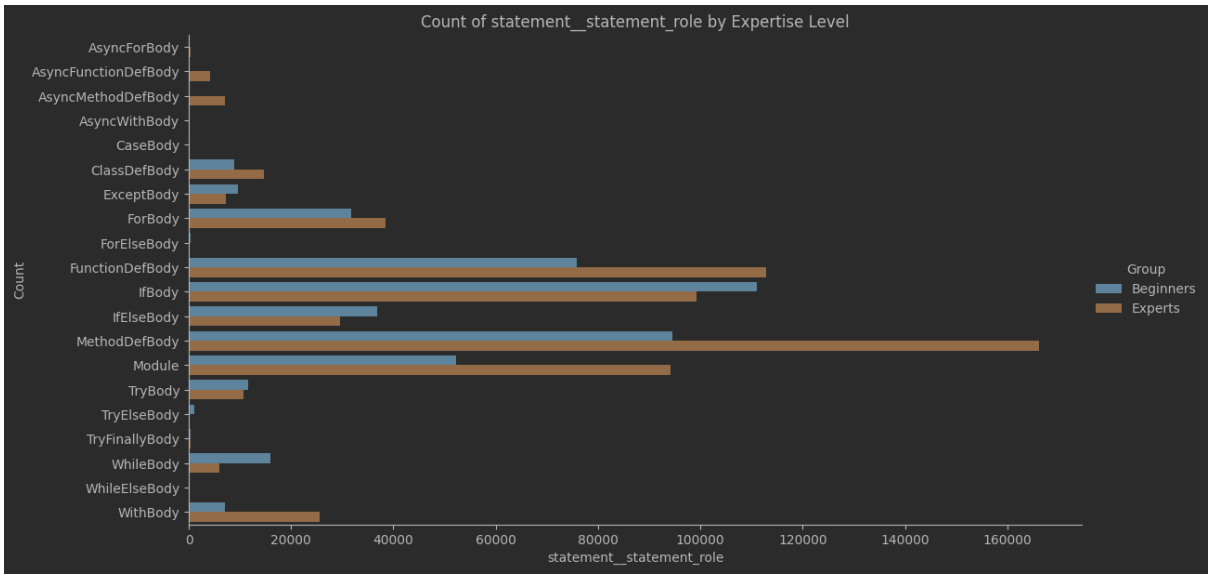

Figura 12: Distribución de valores de la variable statement role en Statements

5.7.4 Categoría sintáctica del padre (parent)

La categoría más usada como padre difiere según la experiencia del programador. Lo principiantes utilizan **`If`** mientras que los expertos utilizan **`MethodDef`**. En ningún caso hemos detectado valores anómalos. En general, nunca se usan las categorías **`AsyncWith, Nonlocal, Continue, Return, Match, ImportFrom, AnnotatedAssignment, Global, Import, Raise, Assert,`**

`Break`, `TypeAlias`, `AugmentedAssignment`, `Pass`, `AssignmentStmt`, `AsyncFor` o **`Delete`.** Estos datos se pueden observar en la siguiente figura:

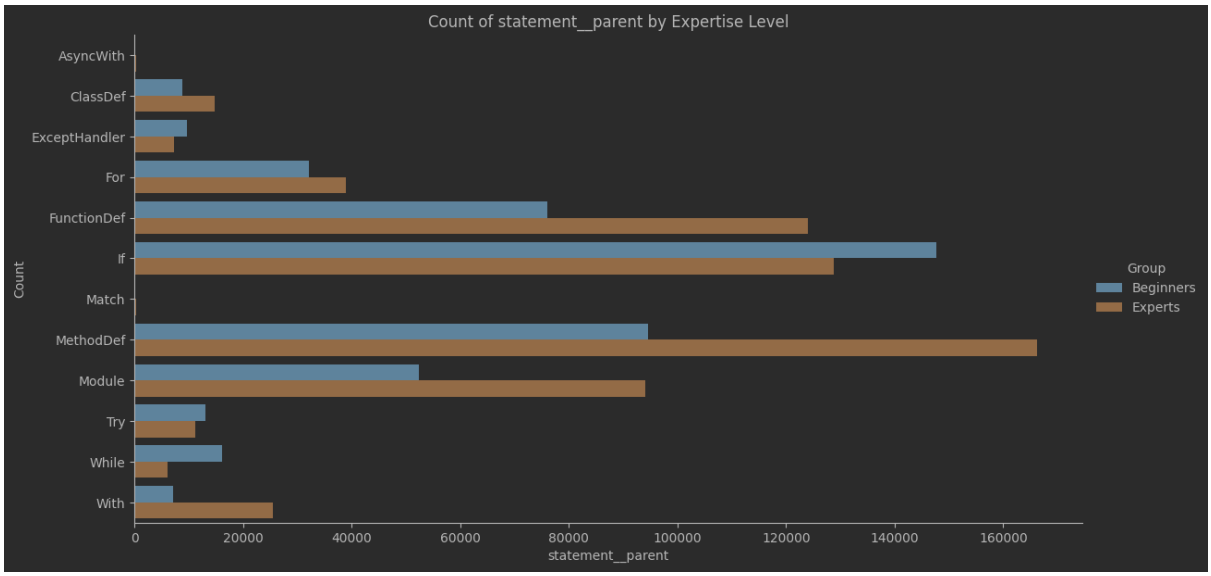

Figura 13: Distribución de valores de la variable parent en Statements

5.7.5 Categoría sintáctica del primer hijo (first child category)

La categoría más usada como primer hijo de sentencia es `Variable` y los valores `Shift`, `SetLiteral`, `SetComprehension`, `Pow`, `UnaryBWNot`, `MatMult` y `AssignmentExp` fueron identificados como anómalos. Por otra parte, las categorías `FormattedValue`, `Star`, `YieldFrom`, `EllipsisLiteral`, `Yield`, `Parameter` y `Slice` no se usan nunca. La siguiente figura muestra los datos utilizados para el análisis:

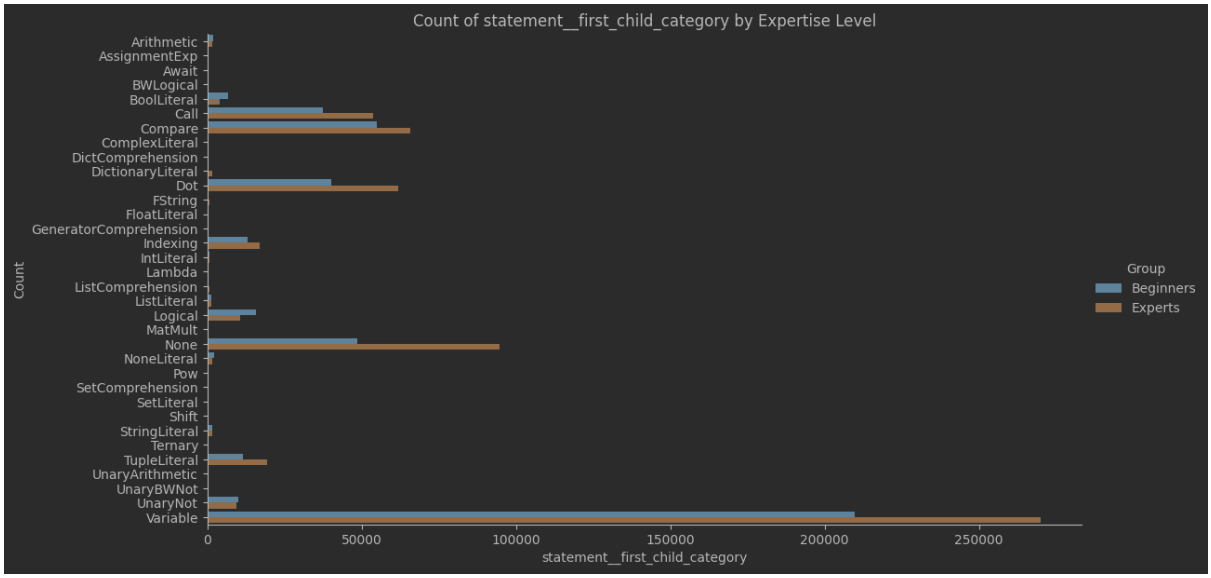

Figura 14: Distribución de valores de la variable first child category en Statements

5.8 Cases

5.8.1 Guards

No hemos detectado ningún *case* con un `Guard`.

5.8.2 Número de cases (number of cases)

Hemos detectado como anómala cualquier sentencia `Match` con más de 10 cases. Tan solo 2 sentencias `Match` han sido detectadas como anómalas.

5.8.3 Otros

No hemos detectado ningún *case* de los tipos `Singleton`, `Sequence`, `Or`, `Mapping` o `Star`.

Los dos tipos de **`Match`** que hemos detectado son **`As`** y **`Value`**. En el caso del **`As`** permite asignar un nombre al elemento comparado en el case. De esta forma, se puede tratar dentro del case bajo el nombre especificado.

```
match x:
    case [x] as y:
         ...
    case 5:
         ...
```

El **`Value`** es el más estándar, comparando con un valor simple.

Entre los principiantes no hay ninguna sentencia del tipo `Match`.

5.9 Handlers

5.9.1 Tiene star (has star)

No hay ninguna sentencia `Try` del tipo `TryStar`.

5.9.2 Otros

Hemos detectado como anómalo cuando el número de *handlers* o el número medio de sentencias del cuerpo es distinto de 1. Cabe destacar el caso anómalo de un `Try` que solo cuenta con una cláusula `finally` sin ningún `except`, esto hace que tenga una media de sentencias en cada except de 0. Las siguientes figuras muestran las gráficas que demuestran ambas conclusiones:

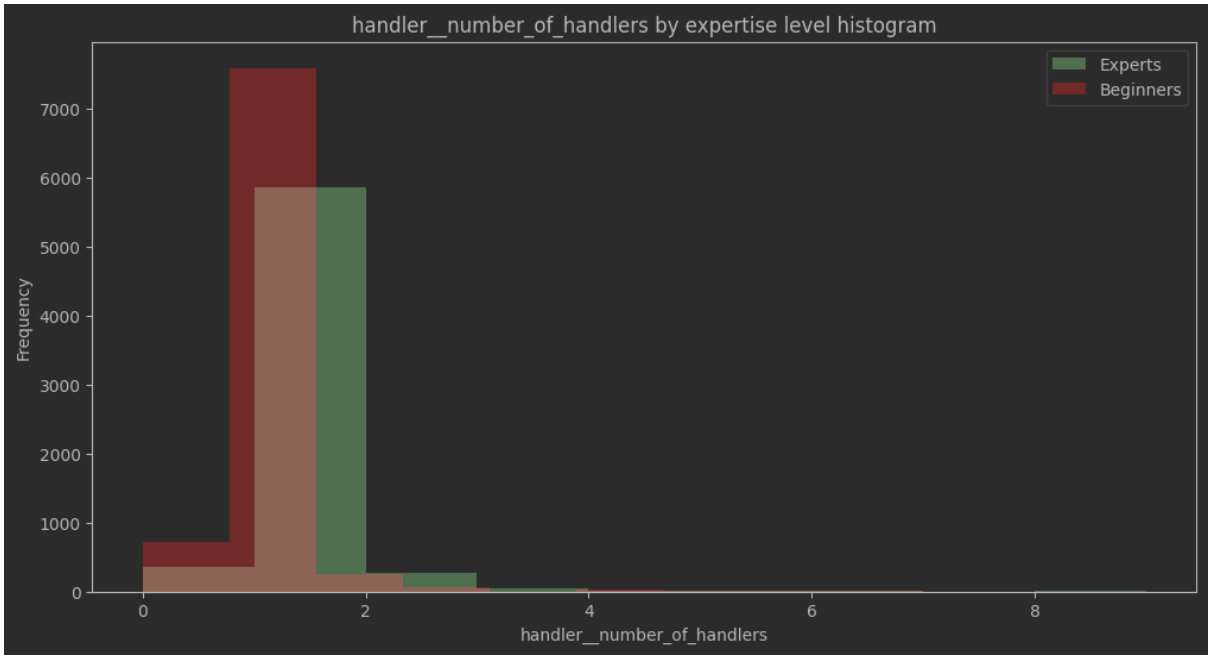

Figura 15: Distribución de valores de la variable number of handlers en Handlers

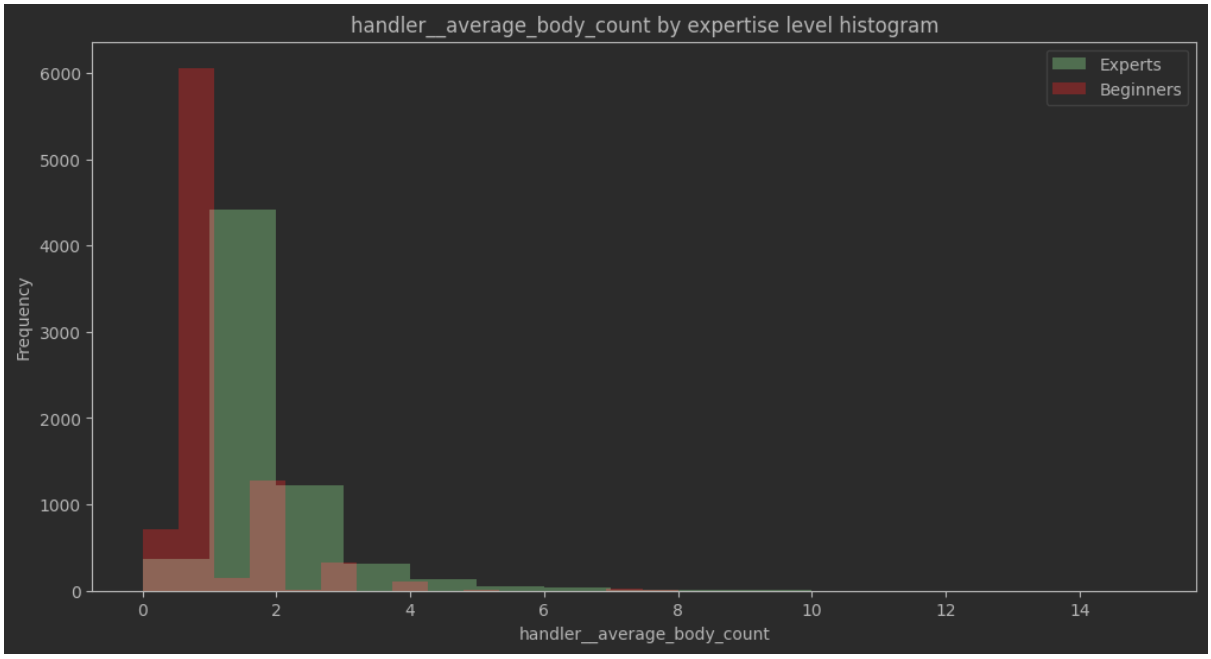

Figura 16: Distribución de valores de la variable average body count en Handlers

5.10 Expressions

5.10.1 Categoría sintáctica (category)

La categoría de expresión más utilizada es la de Variable. Por otro lado, la categoría sintáctica `NoneType` no se usa nunca. En el caso de los principiantes, tampoco hacen uso de expresiones `MatMult`. Finalmente, detectamos como anómalos los valores `AssignmentExp`, `SetComprehension`, `MatMult`, `YieldFrom` y `UnaryBWNot`. La siguiente gráfica muestra las categorías sintácticas utilizadas en función de la experiencia del programador:

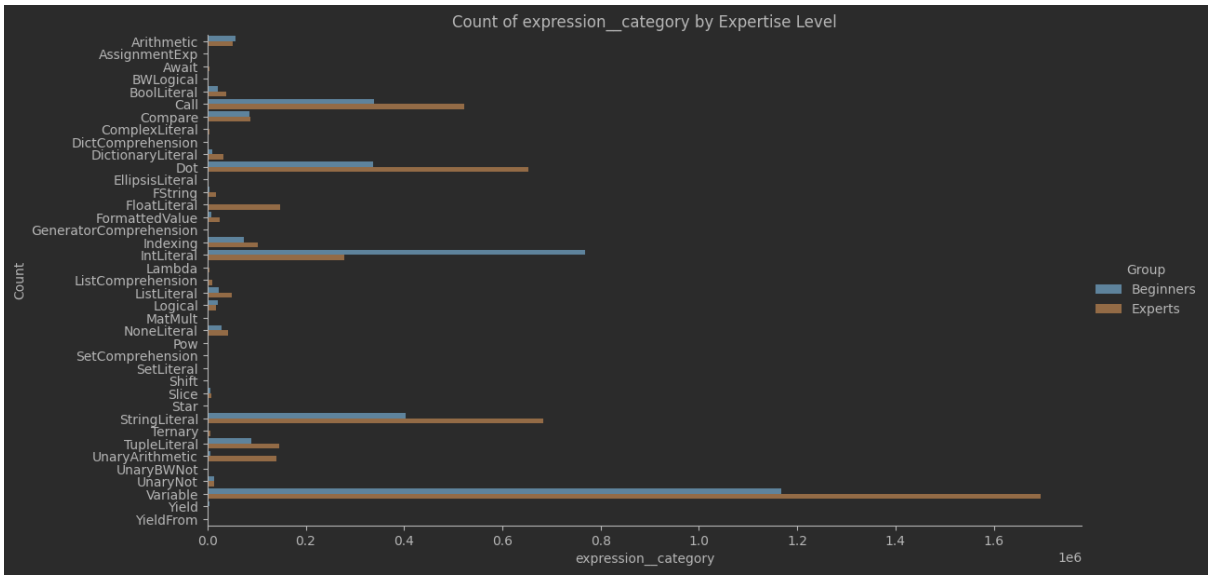

Figura 17: Distribución de valores de la variable category en Expressions

5.10.2 Categoría sintáctica del padre (parent)

En cuanto a la categoría del padre, el valor predominante es `Call`. Los valores `AsyncWith`, `AssignmentExp`, `Match`, `UnaryBWNot` y `MatMult` fueron identificados como anómalos. La siguiente gráfica muestra los datos utilizados para este análisis:

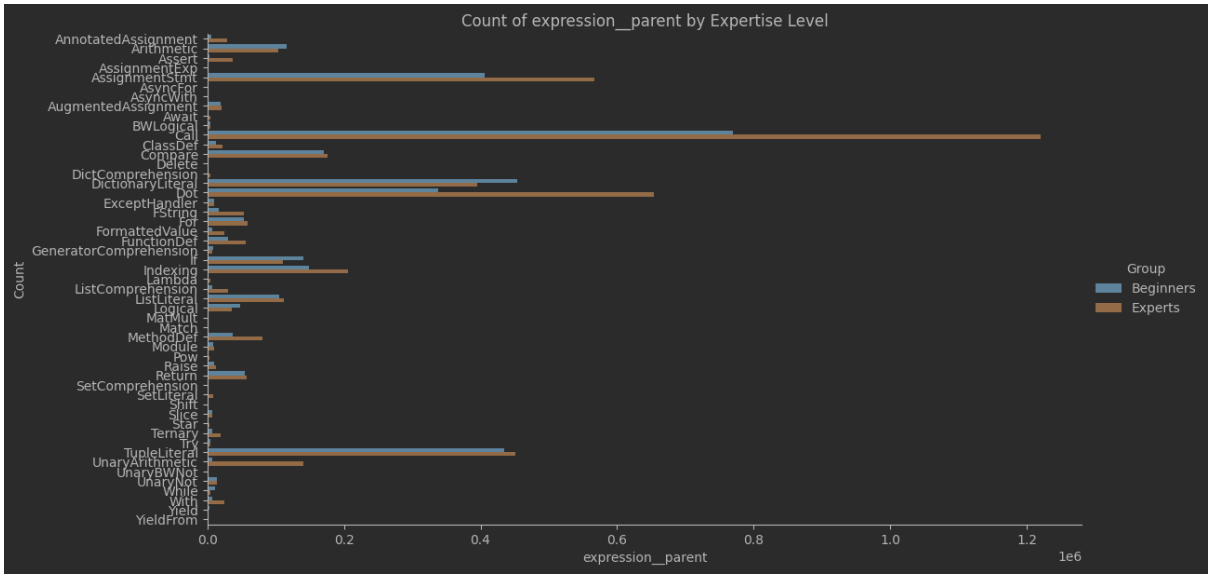

Figura 18: Distribución de valores de la variable parent en Expressions

5.10.3 Categoría sintáctica del primer padre (first child category)

Detectamos como anómalos los primeros hijos de las expresiones con las categorías `AssignmentExp`, `SetComprehension`, `DictComprehension`, `NoneType`, `Shift`, `Lambda`, `GeneratorComprehension`, `MatMult`, `Await`, `UnaryBWNot`, `Star` o `EllipsisLiteral`. Además, las categorías `YieldFrom`, `Yield` y `Parameter` nunca se usan como primer hijo. Todo esto se puede apreciar en la siguiente gráfica:

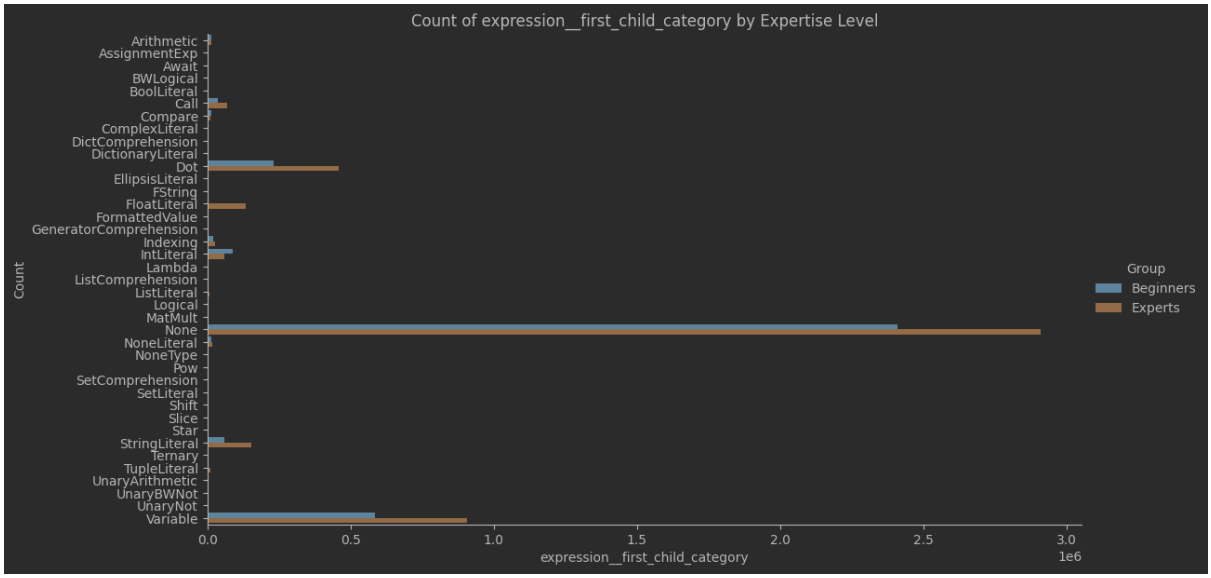

Figura 19: Distribución de valores de la variable first child category en Expressions

5.10.4 Rol de la expresión (expression role)

Respecto a el rol de la expresión, el valor más frecuente es `CallArg`. Sin embargo, en el caso de los principiantes, es `TupleLiteral`. En este análisis, 13 de las 79 categorías posibles fueron identificadas como anómalas (véase 9.2.10). En la siguiente figura se puede apreciar el uso de dichas categorías:

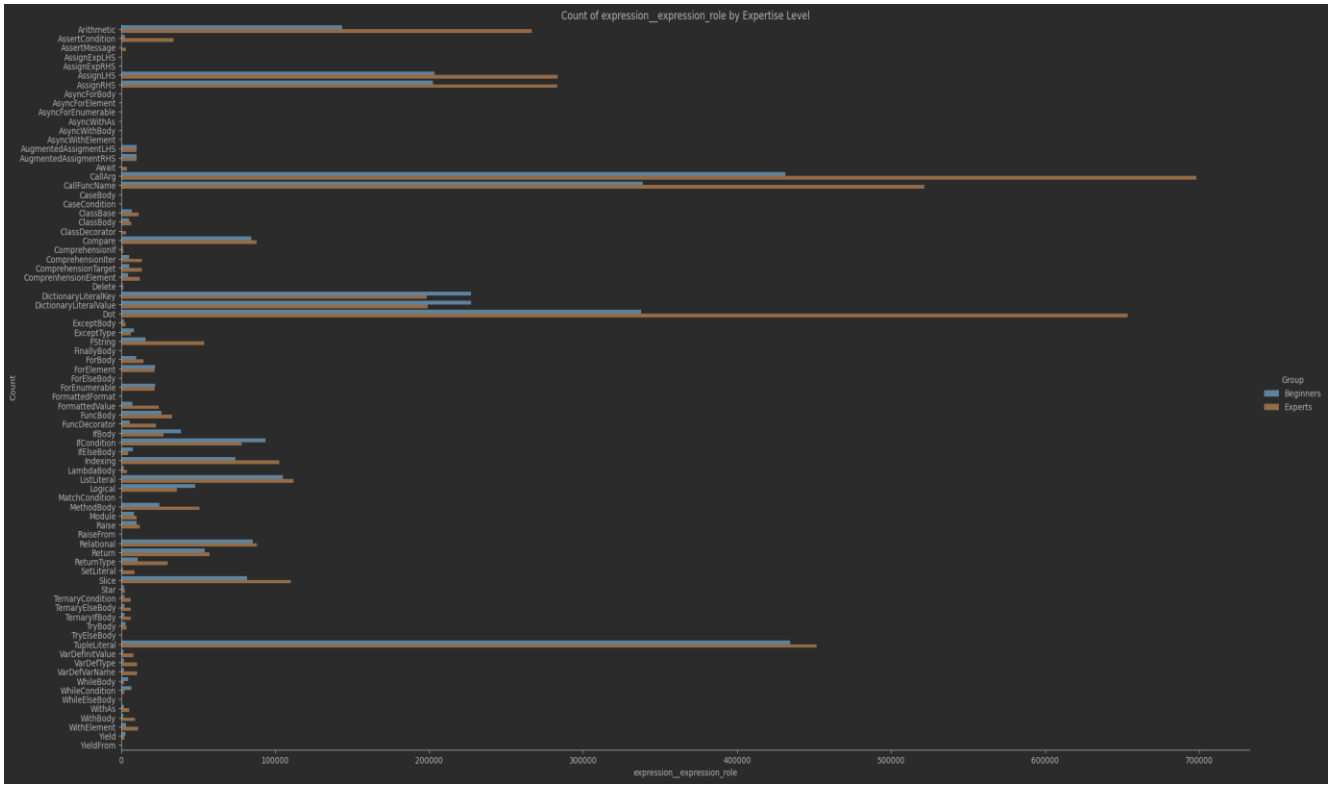

Figura 20: Distribución de valores de la variable expressions role en Expressions

5.11 Comprehensions

Hemos detectado como anómalas las *comprehensions* asíncronas. En el caso de los principiantes, no hemos detectado ninguna de este tipo.

Las comprehensions son estructuras de generación de listas, tuplas o sets. Normalmente, toman listas o valores como parámetro, aplican operaciones o condiciones, y generan otra lista como retorno.

```
numbers2 = {x: x**2 for x in numbers}
```

Este ejemplo, toma una lista llamada numbers y genera otra lista con los valores cuadrados de la lista original.

5.12 CallArgs

Detectamos como anómalas las invocaciones que utilicen argumentos pasados por nombre o *double star*.

5.13 FStrings

Hemos identificado como anómala cualquier cadena formateada que tenga más de 10 elementos. En el análisis, detectamos varias cadenas formateadas con un número de elementos muy superior a la media. En el caso de los principiantes, encontramos una cadena con 43 elementos, mientras que la media es de 3,2. Entre los expertos, identificamos tres cadenas con alrededor de 40 elementos, siendo la media de 3,16. Los datos utilizados para este análisis se pueden apreciar en la siguiente gráfica:

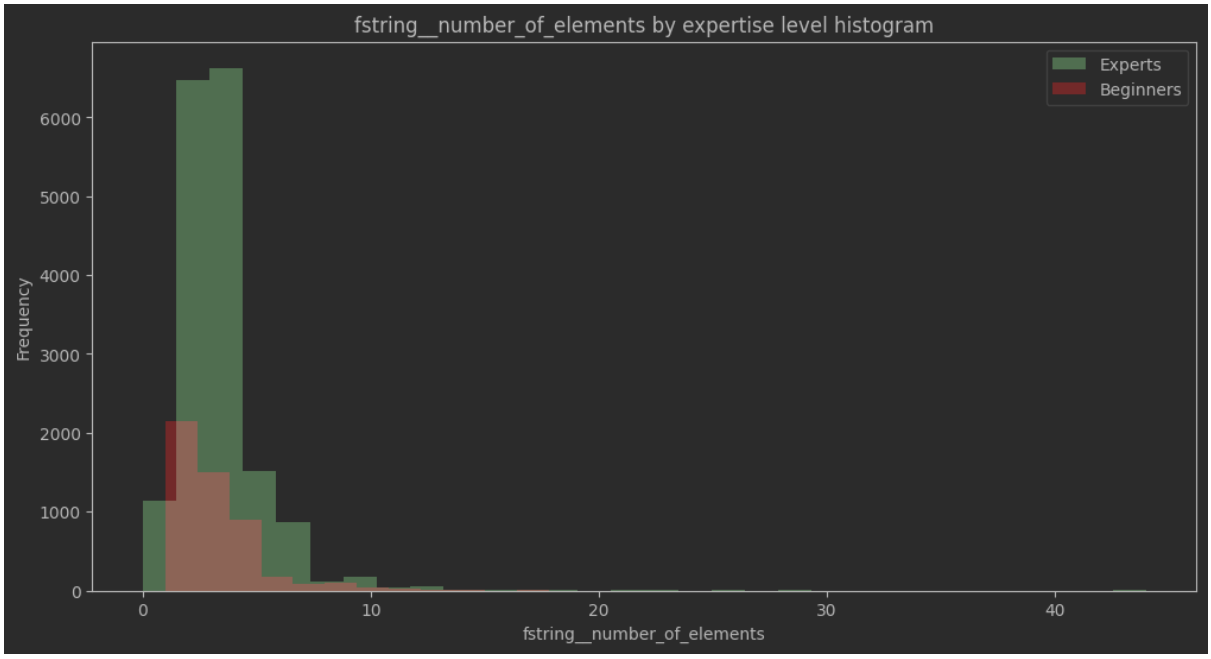

Figura 21: Distribución de valores de la variable number of elements en Fstrings

5.14 Variables

5.14.1 Convenio de nombrado (name convention)

El convenio de nombre más utilizado en todos los casos es `Lower`. No hemos encontrado valores anómalos. La siguiente figura muestra el conjunto de datos utilizado para este análisis:

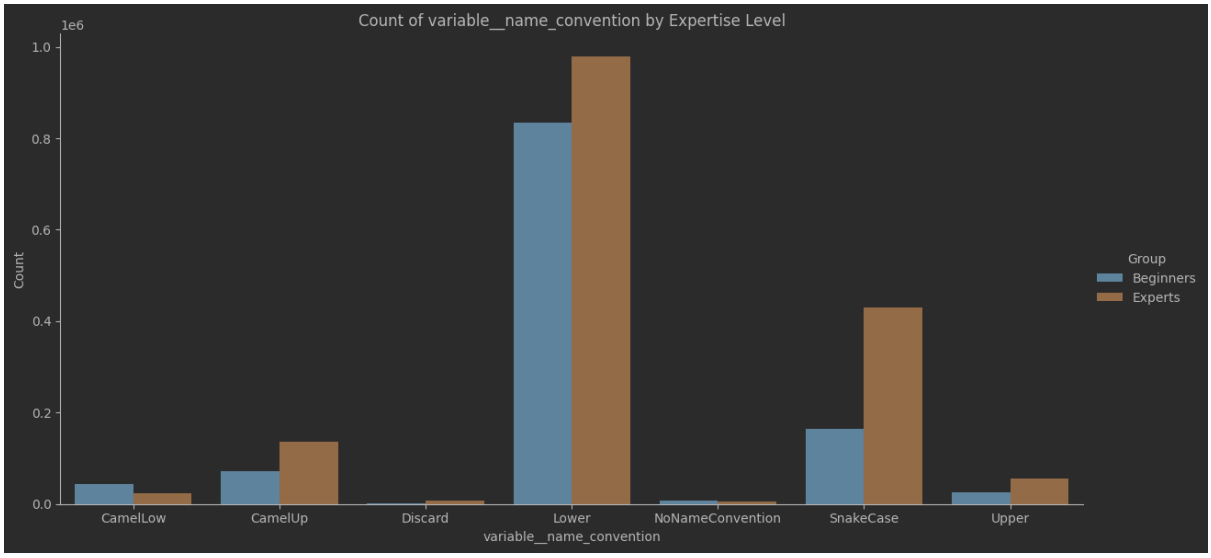

Figura 22: Distribución de valores de la variable name convention en Variables

5.14.2 Análisis multivariante

El análisis multivariante identificó un conjunto de 3.389 variables privadas con nombres largos (24 caracteres de media frente a los 7 de las demás).

5.15 Vectors

5.15.1 Número de elementos (number of elements)

Hemos identificado como anómalos los vectores con un número de elementos superior a 81 o inferior a 2. Cabe destacar que hemos detectado un vector con más de 14.500 elementos, escrito por un experto.

5.15.2 Categoría sintáctica (category)

La categoría de vector más usada es `TupleLiteral`. No hemos encontrado valores anómalos. En la siguiente figura se muestra el uso de cada categoría en función de la experiencia:

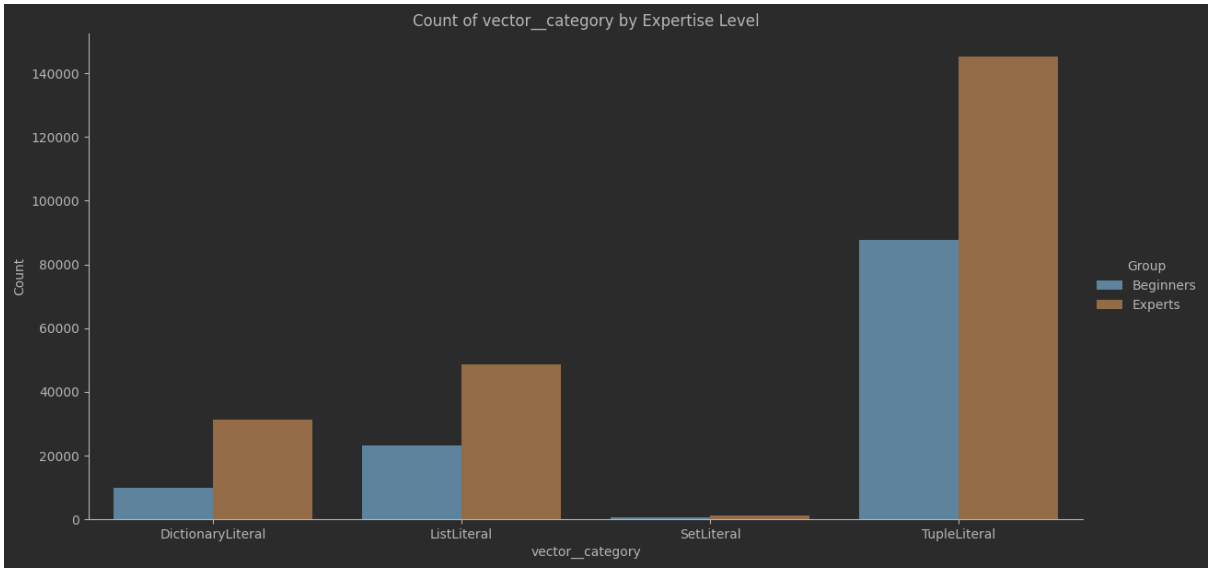

Figura 23: Distribución de valores de la variable category en Vector

5.16 Parameters

5.16.1 Número de parámetros (number of params)

Hemos detectado como anómalo un número de parámetros inferior a 1 y, en el caso de los principiantes, también cuando es superior a 11.

5.16.2 Otros

Hemos detectado como anómalo el uso de argumentos posicionales, con *keyword* o con valor por defecto.

6 Conclusiones y Trabajo Futuro

6.1 Conclusiones

En este trabajo, proponemos un sistema para la extracción y análisis de construcciones sintácticas de código Python, con el objetivo de ampliar el conocimiento sobre cómo se utilizan estas construcciones en la práctica. A través de la identificación y modificación de Árboles de Sintaxis Abstracta (ASTs), hemos logrado representar de manera detallada 16 construcciones sintácticas distintas.

Diseñamos una versión modificada de la salida del módulo AST de la Librería Estándar de Python (*Python Standard Library*). Esta modificación nos permitió ampliar la información sintáctica proporcionada, facilitando una representación más rica y contextualizada de los nodos del árbol. Posteriormente, desarrollamos un método eficiente para almacenar estos ASTs en un modelo relacional, convirtiendo las estructuras jerárquicas en tablas que incluyen información tanto local como global del contexto de los nodos.

Trabajamos con un conjunto de datos extenso, que comprende más de 13 millones de instancias extraídas de más de mil programas de distintas fuentes, escritos por programadores de dos niveles de experiencia: expertos y principiantes. Este amplio y variado conjunto de datos nos permitió generar información empírica valiosa sobre las construcciones sintácticas anómalas, proporcionando una visión detallada de cómo se utilizan las características sintácticas del lenguaje Python en diferentes contextos. A través del análisis de valores anómalos, identificamos no solo casos extremos en el uso de construcciones sintácticas, sino también valores erróneos que fueron excluidos del conjunto de datos final. El informe de anomalías realizado proporciona un conjunto de valores límite y rangos que sirven para identificar construcciones atípicas en programas Python en general, así como para cada uno de los niveles de experiencia analizados.

El sistema propuesto no solo permite identificar las construcciones sintácticas más utilizadas por programadores principiantes y expertos, sino que también destaca las diferencias en su uso. Esto permite una diferenciación clara entre los patrones de programación de ambos grupos, ofreciendo la posibilidad de adaptar estrategias educativas para mejorar el aprendizaje de los principiantes. Por ejemplo, los profesores pueden utilizar esta información para ayudar a los estudiantes a aprender los usos del lenguaje que son más comunes entre los desarrolladores profesionales, promoviendo así mejores prácticas de programación.

6.2 Trabajo Futuro

Como parte del trabajo futuro, planeamos aplicar diversas técnicas de minería de datos, utilizando algoritmos interpretables de aprendizaje automático supervisados y no supervisados sobre distintos conjuntos de datos homogéneos y heterogéneos. Con estas técnicas, esperamos obtener reglas de asociación, agrupamiento (*clustering*) y reglas de clasificación, entre otros.

Además, queremos aumentar la información extraída del compilador, añadiendo información semántica relacionada con el flujo de ejecución, la dependencia de datos y las invocaciones entre métodos [26]. Esto permitirá un análisis más profundo y completo de las construcciones sintácticas, ofreciendo una visión más detallada de cómo se interrelacionan y se utilizan en diferentes contextos de programación.

Asimismo, tras enriquecer los conjuntos de datos con información semántica, planeamos evaluar modelos de redes neuronales de grafos para la clasificación de la experiencia de los programadores [27]. Incluso se podrían desarrollar modelos cuya salida, en lugar de una clase, fuese una puntuación que refleje el nivel de experiencia del programador.

Todos los datos utilizados en nuestro trabajo, el informe de valores anómalos, el nuevo diseño del AST y todo el código fuente utilizado para implementar nuestro sistema están disponibles para su descarga en https://github.com/ComputationalReflection/PythonSourceCodeAnalysis.

7 Planificación y Presupuesto

7.1 Planificación del proyecto

En la siguiente tabla se representan las fechas y las horas de trabajo del proyecto. Las horas de trabajo diarias han variado en función de la carga de trabajo, la dificultad de este y la previsión de plazos restantes. Durante las fases 1, 3 y 4 hemos trabajado 2 horas diarias. En la fase 2 han sido 3 horas. Por último, en la fase de documentación, 4 horas diarias.

Tabla 18: Planificación del Proyecto

Tarea	Duración (h)	Comienzo	Fin
Fase 1 (Diseño y análisis de la solución)	***30***	***15/10/2023***	***02/11/2023***
Decisión del alcance	10	15/10/2023	19/10/2023
Elección de tecnologías	10	20/10/2023	26/10/2023
Diseño de la arquitectura	10	27/10/2023	02/11/2023
Fase 2 (Implementación de la solución)	***294***	***03/11/2023***	***19/03/2024***
Desarrollo del programa	150	03/11/2023	11/01/2024
Desarrollo de la interfaz a la BD	39	12/01/2024	30/01/2024
Creación y configuración del entorno	9	31/01/2024	02/02/2024
Integración de las partes	21	05/02/2024	13/02/2024
Testing y corrección de errores I	75	14/02/2024	19/03/2024
Fase 3 (Generación del dataset)	***70***	***20/03/2024***	***07/05/2024***
Obtención de programas	10	20/03/2024	26/03/2024
Ejecución del programa	10	27/03/2024	02/04/2024
Testing y corrección de errores II	50	03/04/2024	07/05/2024
Fase 4 (Análisis del dataset)	***60***	***08/05/2024***	***18/06/2024***
Análisis del *dataset*	30	08/05/2024	28/05/2024
Testing y corrección de errores III	30	29/05/2024	18/06/2024
Documentación del proyecto	***60***	***19/06/2024***	***09/07/2024***
Total	*514*	*15/10/2023*	*09/07/2024*

7.2 Planificación del proyecto

7.2.1 Precios por hora

Todas las tareas completadas en las diferentes fases del proyecto han sido realizadas de forma secuencial. Una misma persona, tomando el rol de investigador y programador alternativamente ha desarrollado el proyecto en su conjunto.

En las tablas que siguen se representan los salarios por hora que hemos tenido en cuenta para cada uno de los roles. Estos salarios se han calculado teniendo en cuenta la situación actual del mercado laboral en España. En estos salarios hemos tenido en cuenta únicamente el coste directo, los costes indirectos del trabajo se pueden visualizar en el apartado 7.2.3 Presupuesto Total.

Tabla 19: Precio por hora investigador

Unidad	Descripción	Precio/hora (€)	Horas	Subtotal (€)
Hora	Investigador	50,00 €	1	50,00 €
			Precio total / hora	50,00 €

Tabla 20: Precio por hora programador

Unidad	Descripción	Precio/hora (€)	Horas	Subtotal (€)
Hora	Programador	40,00 €	1	40,00 €
			Precio total / hora	40,00 €

7.2.2 Precio por unidad de trabajo

El precio asociado a cada una de las tareas que componen el proyecto se detalla a continuación, considerando el rol que ha realizado dicha tarea y el número de horas dedicadas a su realización.

Como se puede observar, el rol de programador está asociado a las tareas que incluyen la implementación de código, el manejo de sistemas informáticos y el *testing*. Por otro lado, el investigador se encarga del resto de tareas que, en resumen, están relacionadas con los aspectos más teóricos y de toma de decisiones del proyecto.

Tabla 21: Precios por unidad de trabajo. Parte 1: Trabajo Relacionado

Trabajo Relacionado			
Cantidad (horas)	**Descripción**	**Precio (€)**	**Subtotal (€)**
10	Investigador	50,00 €	500,00 €
		Precio total	500,00 €

Tabla 22: Precios por unidad de trabajo. Parte 2: Elección de tecnologías

Elección de tecnologías			
Cantidad (horas)	**Descripción**	**Precio (€)**	**Subtotal (€)**
10	Investigador	50,00 €	500,00 €
		Precio total	500,00 €

Tabla 23: Precios por unidad de trabajo. Parte 3: Diseño de la arquitectura

Diseño de la arquitectura			
Cantidad (horas)	**Descripción**	**Precio (€)**	**Subtotal (€)**
10	Investigador	50,00 €	500,00 €
		Precio total	500,00 €

Tabla 24: Precios por unidad de trabajo. Parte 4: Desarrollo del programa

Desarrollo del programa			
Cantidad (horas)	**Descripción**	**Precio (€)**	**Subtotal (€)**
150	Programador	40,00 €	6.000,00 €
		Precio total	6.000,00 €

Tabla 25: Precios por unidad de trabajo. Parte 5: Desarrollo de la interfaz a la BD

Desarrollo de la interfaz a la BD			
Cantidad (horas)	**Descripción**	**Precio (€)**	**Subtotal (€)**
39	Programador	40,00 €	1.560,00 €
		Precio total	1.560,00 €

Tabla 26: Precios por unidad de trabajo. Parte 6: Creación y configuración del entorno

Creación y configuración del entorno			
Cantidad (horas)	**Descripción**	**Precio (€)**	**Subtotal (€)**
9	Programador	40,00 €	360,00 €
		Precio total	360,00 €

Tabla 27: Precios por unidad de trabajo. Parte 7: Integración de las partes

Integración de las partes			
Cantidad (horas)	**Descripción**	**Precio (€)**	**Subtotal (€)**
21	Programador	40,00 €	840,00 €
		Precio total	840,00 €

Tabla 28: Precios por unidad de trabajo. Parte 8: Testing y corección de errores I

***Testing* y corrección de errores I**			
Cantidad (horas)	**Descripción**	**Precio (€)**	**Subtotal (€)**
75	Programador	40,00 €	3.000,00 €
		Precio total	3.000,00 €

Tabla 29: Precios por unidad de trabajo. Parte 9: Obtención de programas

Obtención de programas

Cantidad (horas)	Descripción	Precio (€)	Subtotal (€)
10	Investigador	50,00 €	500,00 €
		Precio total	500,00 €

Tabla 30: Precios por unidad de trabajo. Parte 10: Ejecución del programa

Ejecución del programa

Cantidad (horas)	Descripción	Precio (€)	Subtotal (€)
10	Investigador	50,00 €	500,00 €
		Precio total	500,00 €

Tabla 31: Precios por unidad de trabajo. Parte 11: Testing y corrección de errores II

***Testing* y corrección de errores II**

Cantidad (horas)	Descripción	Precio (€)	Subtotal (€)
50	Programador	40,00 €	2.000,00 €
		Precio total	2.000,00 €

Tabla 32: Precios por unidad de trabajo. Parte 12: Análisis del dataset

Análisis del *dataset*

Cantidad (horas)	Descripción	Precio (€)	Subtotal (€)
30	Investigador	50,00 €	1.500,00 €
		Precio total	1.500,00 €

Tabla 33: Precios por unidad de trabajo. Parte 13: Testing y corrección de errores III

***Testing* y corrección de errores III**

Cantidad (horas)	Descripción	Precio (€)	Subtotal (€)
30	Programador	40,00 €	1.200,00 €
		Precio total	1.200,00 €

Tabla 34: Precios por unidad de trabajo. Parte 14: Documentación del proyecto

Documentación del proyecto

Cantidad (horas)	Descripción	Precio (€)	Subtotal (€)
60	Investigador	50,00 €	3.000,00 €
		Precio total	3.000,00 €

7.2.3 Presupuesto total

La Tabla 35 muestra el presupuesto final del proyecto. En este presupuesto se incluyen los precios de cada una de las tareas listadas en el apartado anterior, además de los costes indirectos (un 10% de los costes directos). El listado de conceptos incluidos dentro de los costes indirectos puede verse listados en la Tabla 36.

Tabla 35: Presupuesto total

Unidades	Descripción	Precio (€)
1	Precio 1: Decisión del alcance	500,00 €
1	Precio 2: Elección de tecnologías	500,00 €
1	Precio 3: Diseño de la arquitectura	500,00 €
1	Precio 4: Desarrollo del programa	6.000,00 €
1	Precio 5: Desarrollo de la interfaz a la BD	1.560,00 €
1	Precio 6: Creación y configuración del entorno	360,00 €
1	Precio 7: Integración de las partes	840,00 €
1	Precio 8: *Testing* y corrección de errores I	3.000,00 €
1	Precio 9: Obtención de programas	500,00 €
1	Precio 10: Ejecución del programa	500,00 €
1	Precio 11: *Testing* y corrección de errores II	2.000,00 €
1	Precio 12: Análisis de *dataset*	1.500,00 €
1	Precio 13: *Testing* y corrección de errores III	1.200,00 €
1	Precio 14: Documentación del proyecto	3.000,00 €
	Precio total	21.960,00 €
	Costes indirectos (10%)	2.196,00 €
	Presupuesto total (directos + indirectos)	**24.156,00 €**

Tabla 36: Conceptos de los costes indirectos

Conceptos
Electricidad
Internet
Comunicaciones
Dietas
Transporte
Limpieza y mantenimiento

8 Referencias

[1] Allamanis, M. and Sutton, C., «Mining source code repositories at massive scale using language modeling,» de *10th Working Conference on Mining Software Repositories (MSR)*, 2013.

[2] Ortin, F., Escalada J., and Rodriguez-Prieto, O., «Big code: New opportunities for improving software construction,» *Journal of Software,* vol. 11, nº 11, pp. 1083-1088, 2016.

[3] Bird, C., Ford, D., Zimmermann, T., Forsgren, N., Kalliamvakou, E., Lowdermilk, T., and Gazit, I., «Taking Flight with Copilot: Early insights and opportunities of AI-powered pair-programming tools,» *ACM Queue,* vol. 20, nº 6, p. 35–57, 2023.

[4] Allamanis, M., Barr, E.T., Devanbu, P. and Sutton, C., «A survey of machine learning for big code and naturalness,» *ACM Computing Surveys (CSUR),* vol. 51, nº 4, pp. 1-37, 2018.

[5] IEEE Spectrum, «Top Programming Languages 2023,» [En línea]. Available: https://spectrum.ieee.org/top-programming-languages-2023. [Último acceso: Junio 2024].

[6] KDnuggets , «Data Science Tools, 2019-2020: Trends and Analysis,» [En línea]. Available: https://www.kdnuggets.com/2020/06/data-science-tools-popularity-animated.html. [Último acceso: junio 2024].

[7] Iyer, V., and Zilles, C., «Pattern census: A characterization of pattern usage in early programming courses,» de *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education (SIGCSE '21)*, New York, NY, USA, 2021.

[8] Losada, A., Facundo, G., Garcia, M., and Ortin, F., «Mining common syntactic patterns used by Java programmers,» *IEEE Latin America Transactions,* vol. 20, nº 5, pp. 753-762, 2022.

[9] Python 3.12.3, «The Python Standard Library, ast - Abstract Syntax Trees,» [En línea]. Available: https://docs.python.org/3/library/ast.html. [Último acceso: Junio 2024].

[10] Peng, Y., Zhang, Y., and Hu, M., «An Empirical Study for Common Language Features Used in Python Projects,» de *IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, Honolulu, HI, USA, 2021.

[11] Dakhel, A. M., Desmarais, M. C., and Khomh, F., «Assessing Developer Expertise from the Statistical Distribution of Programming Syntax Patterns,» de *Proceedings of the 25th Evaluation and Assessment in Software Engineering Conference (EASE 2021)*, 2021.

[12] Robles, G., Kula, R. G., Ragkhitwetsagul, C., Sakulniwat, T., Matsumoto, K., and Gonzalez-Barahona, J. M, «Pycefr: Python competency level through code analysis,» de *Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension (ICPC '22)*, New York, NY, USA, 2022.

[13] Ortin, F., Facundo, G., and Garcia, M., «Analyzing syntactic constructs of Java programs with machine learning,» *Elsevier Expert Systems with Applications,* vol. 215, pp. 119398-119414, 2023.

[14] Gamma, E., Helm, R., Johnson, R., and Vlissides, J., «Design Patterns: Abstraction and Reuse of Object-Oriented Design,» de *European Conf. on Object-Oriented Programming*, 1993.

[15] Van Rossum, G. and Drake Jr, F.L., «Python tutorial,» de *Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands*, 1995.

[16] Joglekar, M., Garcia-Molina, H. and Parameswaran, A, «Interactive data exploration with smart drill-down,» *IEEE Transactions on Knowledge and Data Engineering,* vol. 31, nº 1, pp. 46-60, 2017.

[17] Ortin, F., Rodriguez-Prieto, O., Pascual, N. and Garcia, M., «Heterogeneous tree structure classification to label Java programmers according to their expertise level,» *Future Generation Computer Systems,* vol. 105, pp. 380-394, 2020.

[18] Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederic, J., and Willing, C., «Jupyter Notebooks – a publishing format for reproducible computational workflows,» *Positioning and Power in Academic Publishing: Players, Agents and Agendas,* pp. 87-90, 2016.

[19] Computational Reflection, «Python Source Code Analysis,» [En línea]. Available: https://github.com/ComputationalReflection/PythonSourceCodeAnalysis. [Último acceso: junio 2024].

[20] GitHub, «REST API documentation,» [En línea]. Available: https://docs.github.com/es/rest. [Último acceso: junio 2024].

[21] Adams, J., Hayunga, D., Mansi, S., Reeb, D. and Verardi, V., «Identifying and treating outliers in finance,» *Financial Management,* vol. 48, nº 2, pp. 345-384, 2019.

[22] Tukey, J.W., Exploratory data analysis. Vol. 2., Reading, MA: Addison-wesley, 1977.

[23] Rousseeuw, P. J., and Ruts, I., «Algorithm AS 307: Bivariate Location Depth,» *Journal of the Royal Statistical Society. Series C (Applied Statistics),* vol. 45, nº 5, pp. 516-526, 1996.

[24] Hubert, M. and Vandervieren, E., «An adjusted boxplot for skewed distributions,» *Computational statistics & data analysis,* vol. 52, nº 12, pp. 5186-5201, 2008.

[25] Liu, F.T., Ting, K.M. and Zhou, Z.H., «Isolation forest,» *2008 eighth ieee international conference on data mining,* pp. 413-422, 2008.

[26] Rodriguez-Prieto, O., Mycroft, A., and Ortin, F., «An efficient and scalable platform for Java source code analysis using overlaid graph representations,» *IEEE Access,* vol. 8, p. 72239–72260, 2020.

[27] Wu, L., Cui, P., Pei, J., Zhao, L., and Song, L., «Graph neural networks,» *Graph Neural Networks: Foundations, Frontiers, and Applications,* pp. 27-37, 2022.

[28] Alfred, V. Aho, S. Lam Monica, and D. Ullman Jeffrey, Compilers Principles, Techniques and Tools, Pearson Education, 2007.

9 Anexos

9.1 Dominio de las características

9.1.1 Statement Category

Return, Delete, AssignmentStmt, TypeAlias, AugmentedAssignment, AnnotatedAssignment, For, AsyncFor, If, While, With, AsyncWith, Match, Raise, Try, Assert, Global, NonLocal, Pass, Break, Continue, ExceptHandler, Import, ImportFrom.

9.1.2 Statement Role

Module, IfBody, IfElseBody, FunctionDefBody, AsyncFunctionDefBody, MethodDefBody, AsyncMethodDefBody, ClassDefBody, ForBody, ForElseBody, AsyncForBody, AsyncForElseBody, WithBody, WhileBody, WhileElseBody, ExceptBody, AsyncWithBody, TryBody, TryElseBody, TryFinallyBody, TryHandler, TryHandlerStar, CaseBody.

9.1.3 Expression Category

Logical, AssignmentExp, Arithmetic, Pow, Shift, BWLogical, MatMult, UnaryArithmetic, UnaryNot, UnaryBWNot, Lambda, Ternary, SetLiteral, ListLiteral, TupleLiteral, DictionaryLiteral, ListComprehension, SetComprehension, DictComprehension, GeneratorComprehension, Await, Yield, YieldFrom, Compare, Call, FString, FormattedValue, IntLiteral, FloatLiteral, ComplexLiteral, NoneLiteral, BoolLiteral, StringLiteral, EllipsisLiteral, Dot, Variable, Slice, Indexing, Star, NoneType.

9.1.4 Expression Role

Module, FuncDecorator, FuncBody, ReturnType, ClassBase, ClassDecorator, MethodBody, ClassBody, Return, Delete, AssignLHS, AssignRHS, TypeAliasLHS, TypeAliasRHS, AugmentedAssignmentLHS, AugmentedAssignmentRHS, VarDefVarName, VarDefType, VarDefInitValue, ForElement, ForEnumerable, ForBody, ForElseBody, AsyncForElement, AsyncForEnumerable, AsyncForBody, AsyncForElseBody, WhileCondition, WhileBody, WhileElseBody, IfCondition, IfBody, IfElseBody, WithElement, WithAs, WithBody, AsyncWithElement, AsyncWithAs, AsyncWithBody, MatchCondition, CaseCondition, CaseGuard, CaseBody, Raise, RaiseFrom, TryBody, ExceptType, ExceptBody, TryElse, FinallyBody, AssertCondition, AssertMessage, Logical, AssignExpLHS, AssignExpRHS, Arithmetic, Pow, Shift, BWLogical, MatMult, LambdaBody, TernaryCondition, TernaryIfBody, TernaryElseBody, SetLiteral, ListLiteral, TupleLiteral, DictionaryLiteralKey, DictionaryLiteralValue, ComprehensionElement, ComprehensionTarget, ComprehensionIter, ComprehensionIf, Await, Yield, YieldFrom, Relational, Is, In, CallFuncName, CallArg, FString, Dot, Slice, Indexing, Star, TypeAnnotation,

DefaultParamValue, TypeVar, FormattedFormat, AugmentedAssigmentLHS, Compare, FormattedValue, AugmentedAssigmentRHS, TryElseBody, ComprenhensionElement.

9.2 Resultados Detección de Anomalías

En la Sección 5 describimos las instancias catalogadas como anómalas más relevantes para nuestro estudio. A continuación, enumeramos, por conjunto de datos estudiado, los valores de cada una de las características que hacen que sean detectados como anómalos. En cada caso, se muestran los valores teniendo en cuenta todos los datos, solo con nivel de experiencia *Beginner* y solo con nivel de experiencia *Expert*.

9.2.1 Programs

- Contiene subdirectorios con código (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
    - Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
    - *Beginner*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
    - *Expert*: El valor predominante es true. No se detectaron valores anómalos para esta característica.
- Contiene paquetes (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
    - Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
    - *Beginner*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
    - *Expert*: El valor predominante es true. No se detectaron valores anómalos para esta característica.
- Contiene código en el directorio raíz (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
    - Todos: El valor predominante es true. No se detectaron valores anómalos para esta característica.
    - *Beginner*: El valor predominante es true. No se detectaron valores anómalos para esta característica.
    - *Expert*: El valor predominante es true. No se detectaron valores anómalos para esta característica.
- Número de módulos (Numérica):
    - Todos: Se detecta como anómalo cuando es superior a 143,7.
    - *Beginner*: Se detecta como anómalo cuando es superior a 148,39.
    - *Expert*: Se detecta como anómalo cuando es superior a 1764,39.
- Número de subdirectorios con código (Numérica):
    - Todos: Se detecta como anómalo cuando es superior a 0.
    - *Beginner*: Se detecta como anómalo cuando es superior a 0.
    - *Expert*: Se detecta como anómalo cuando es superior a 145.
- Número de paquetes (Numérica):
    - Todos: Se detecta como anómalo cuando es superior a 0.

  - *Beginner*: Se detecta como anómalo cuando es superior a 0.
  - *Expert*: Se detecta como anómalo cuando es superior a 172,5.
- Media de definiciones por módulo (Numérica):
  - Todos: Se detecta como anómalo cuando es superior a 8,57.
  - *Beginner*: Se detecta como anómalo cuando es superior a 8.
  - *Expert*: Se detecta como anómalo cuando es superior a 8,72.
- Proporción de definiciones de clases (Numérica):
  - Todos: Se detecta como anómalo cuando es superior a 0.
  - *Beginner*: Se detecta como anómalo cuando es superior a 0.
  - *Expert*: Se detecta como anómalo cuando es superior a 0,82.
- Proporción de definiciones de funciones (Numérica):
  - Todos: No se detectaron valores anómalos para esta característica.
  - *Beginner*: No se detectaron valores anómalos para esta característica.
  - *Expert*: Se detecta como anómalo cuando es inferior a 0,43.
- Proporción de definiciones de enumeraciones (Numérica):
  - Todos: Se detecta como anómalo cuando es superior a 0.
  - *Beginner*: Se detecta como anómalo cuando es superior a 0**.**
  - *Expert*: Se detecta como anómalo cuando es superior a 0,017.
- Análisis Multivariante
  - Todos: Se detectaron 2 instancias anómalas.
  - *Beginner*: Se detectó 1 instancia anómala.
  - *Expert*: Se detectó 1 instancia anómala.

9.2.2 Modules

- Contiene comentario de módulo (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
  - Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  - *Beginner*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  - *Expert*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
- Contiene punto de entrada (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
  - Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  - *Beginner*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  - *Expert*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
- Número de clases (Numérica):
  - Todos: Se detecta como anómalo cuando es superior a 76.
  - *Beginner*: Se detecta como anómalo cuando es superior a 72,7.
  - *Expert*: Se detecta como anómalo cuando es superior a 73,08.
- Número de funciones (Numérica):
  - Todos: Se detecta como anómalo cuando es superior a 185.
  - *Beginner*: Se detecta como anómalo cuando es superior a 185,2.
  - *Expert*: Se detecta como anómalo cuando es superior a 143.

- Media de sentencias en el cuerpo de las funciones (Numérica):
  - Todos: Se detecta como anómalo cuando es superior a 17,6.
  - *Beginner*: Se detecta como anómalo cuando es superior a 19,4.
  - *Expert*: Se detecta como anómalo cuando es superior a 41,9.
- Media de sentencias en el cuerpo de los métodos (Numérica):
  - Todos: Se detecta como anómalo cuando es superior a 37,3.
  - *Beginner*: Se detecta como anómalo cuando es superior a 5,4.
  - *Expert*: Se detecta como anómalo cuando es superior a 59.
- Convención de nombrado (Nominal): En todos los casos esta variable toma 6 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,033%.
  - Todos: El valor predominante que toma esta variable es el de `SnakeCase`. No se detectaron valores anómalos para esta característica.
  - *Beginner*: El valor predominante que toma esta variable es el de `Lower`. No se detectaron valores anómalos para esta característica.
  - *Expert*: El valor predominante que toma esta variable es el de `SnakeCase`. No se detectaron valores anómalos para esta característica.
- Proporción de sentencias globales (Numérica):
  - Todos: Se detecta como anómalo cuando es superior a 0,83.
  - *Beginner*: No se detectaron valores anómalos para esta característica.
  - *Expert*: Se detecta como anómalo cuando es superior a 0,625.
- Proporción de expresiones globales (Numérica):
  - Todos: Se detecta como anómalo cuando es superior a 0,5.
  - *Beginner*: Se detecta como anómalo cuando es superior a 0,8.
  - *Expert*: Se detecta como anómalo cuando es superior a 0,36.
- Proporción de definiciones de clases (Numérica):
  - Todos: No se detectaron valores anómalos para esta característica.
  - *Beginner*: No se detectaron valores anómalos para esta característica.
  - *Expert*: No se detectaron valores anómalos para esta característica.
- Proporción de definiciones de funciones (Numérica):
  - Todos: No se detectaron valores anómalos para esta característica.
  - *Beginner*: No se detectaron valores anómalos para esta característica.
  - *Expert*: No se detectaron valores anómalos para esta característica.
- Proporción de definiciones de enumeraciones (Numérica):
  - Todos: Se detecta como anómalo cuando es superior a 0.
  - *Beginner*: Se detecta como anómalo cuando es superior a 0.
  - *Expert*: Se detecta como anómalo cuando es superior a 0.
- Proporción de anotaciones de tipos (Numérica):
  - Todos: Se detecta como anómalo cuando es superior a 0,96.
  - *Beginner*: Se detecta como anómalo cuando es superior a 0.
  - *Expert*: No se detectaron valores anómalos para esta característica.
- Análisis Multivariante
  - Todos: Se detectaron 22 instancias anómalas.
  - *Beginner*: Se detectaron 6 instancias anómalas.
  - *Expert*: Se detectaron 14 instancias anómalas.

9.2.3 Imports

- Número de imports (Numérica):
  - Todos: Se detecta como anómalo cuando es superior a 123,8.

  - *Beginner*: Se detecta como anómalo cuando es superior a 20.
  - *Expert*: Se detecta como anómalo cuando es superior a 30.
- Media de módulos importados (Numérica):
  - Todos: Se detecta como anómalo cuando es superior a 4.
  - *Beginner*: Se detecta como anómalo cuando es superior a 4.
  - *Expert*: Se detecta como anómalo cuando es superior a 4.
- Media de módulos importados con un From (Numérica):
  - Todos: Se detecta como anómalo cuando es superior a 6,28.
  - *Beginner*: Se detecta como anómalo cuando es superior a 4,57.
  - *Expert*: Se detecta como anómalo cuando es superior a 19,7.
- Media de módulos importados con un As (Numérica):
  - Todos: Se detecta como anómalo cuando es superior a 0.
  - *Beginner*: Se detecta como anómalo cuando es superior a 0.
  - *Expert*: Se detecta como anómalo cuando es superior a 0.
- Proporción de módulos importados con un Import simple (Numérica):
  - Todos: No se detectaron valores anómalos para esta característica.
  - *Beginner*: No se detectaron valores anómalos para esta característica.
  - *Expert*: No se detectaron valores anómalos para esta característica.
- Proporción de módulos importados con un Import From (Numérica):
  - Todos: No se detectaron valores anómalos para esta característica.
  - *Beginner*: No se detectaron valores anómalos para esta característica.
  - *Expert*: No se detectaron valores anómalos para esta característica.
- Proporción de imports locales (Numérica):
  - Todos: No se detectaron valores anómalos para esta característica.
  - *Beginner*: No se detectaron valores anómalos para esta característica.
  - *Expert*: No se detectaron valores anómalos para esta característica.
- Análisis Multivariante
  - Todos: Se detectaron 17 instancias anómalas.
  - *Beginner*: Se detectaron 9 instancias anómalas.
  - *Expert*: Se detectaron 13 instancias anómalas.

9.2.4 Class Definitions

- Si es una definición de enumerado (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
  - Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  - *Beginner*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  - *Expert*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
- Contiene anotación de tipo genérica (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
  - Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  - *Beginner*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  - *Expert*: El valor predominante es false. No se detectaron valores anómalos para esta característica.

- Contiene comentario de clase (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
    - Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
    - *Beginner*: El valor predominante es true. No se detectaron valores anómalos para esta característica.
    - *Expert*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
- Contiene una meta clase (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
    - Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
    - *Beginner*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
    - *Expert*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
- Número de caracteres del nombre (Numérica):
    - Todos: Se detecta como anómalo cuando es superior a 45.
    - *Beginner*: Se detecta como anómalo cuando es superior a 29.
    - *Expert*: Se detecta como anómalo cuando es superior a 49.
- Número de decoradores (Numérica):
    - Todos: Se detecta como anómalo cuando es superior a 0.
    - *Beginner*: Se detecta como anómalo cuando es superior a 0.
    - *Expert*: Se detecta como anómalo cuando es superior a 0.
- Número de métodos (Numérica):
    - Todos: Se detecta como anómalo cuando es superior a 17.
    - *Beginner*: Se detecta como anómalo cuando es superior a 21.
    - *Expert*: Se detecta como anómalo cuando es superior a 17.
- Número de clases base (Numérica):
    - Todos: Se detecta como anómalo cuando es superior a 1.
    - *Beginner*: Se detecta como anómalo cuando es superior a 1.
    - *Expert*: Se detecta como anómalo cuando es superior a 4.
- Media de sentencias en el cuerpo de los métodos (Numérica):
    - Todos: Se detecta como anómalo cuando es superior a 13.
    - *Beginner*: Se detecta como anómalo cuando es superior a 10,7.
    - *Expert*: Se detecta como anómalo cuando es superior a 14.
- Número de sentencias en el cuerpo de la clase (Numérica):
    - Todos: Se detecta como anómalo cuando es superior a 22.
    - *Beginner*: Se detecta como anómalo cuando es superior a 26.
    - *Expert*: Se detecta como anómalo cuando es superior a 18.
- Número de keywords (Numérica):
    - Todos: Se detecta como anómalo cuando es superior a 0.
    - *Beginner*: Se detecta como anómalo cuando es superior a 0.
    - *Expert*: Se detecta como anómalo cuando es superior a 0.
- Altura (Numérica):
    - Todos: Se detecta como anómalo cuando es superior a 1.
    - *Beginner*: Se detecta como anómalo cuando es superior a 1.

    - *Expert*: Se detecta como anómalo cuando es superior a 1.
- Proporción de asignaciones en el cuerpo de la clase (Numérica):
    - Todos: Se detecta como anómalo cuando es superior a 0,72.
    - *Beginner*: Se detecta como anómalo cuando es superior a 0,57.
    - *Expert*: Se detecta como anómalo cuando es superior a 0,625.
- Proporción de expresiones en el cuerpo de la clase (Numérica):
    - Todos: Se detecta como anómalo cuando es superior a 0,625.
    - *Beginner*: Se detecta como anómalo cuando es superior a 0,83.
    - *Expert*: Se detecta como anómalo cuando es superior a 0,66.
- Proporción de anotaciones de tipo (Numérica):
    - Todos: Se detecta como anómalo cuando es superior a 0,8.
    - *Beginner*: Se detecta como anómalo cuando es superior a 0.
    - *Expert*: Se detecta como anómalo cuando es superior a 0,83.
- Proporción de métodos privados (Numérica):
    - Todos: Se detecta como anómalo cuando es superior a 0.
    - *Beginner*: Se detecta como anómalo cuando es superior a 0.
    - *Expert*: Se detecta como anómalo cuando es superior a 0.
- Proporción de métodos mágicos (Numérica):
    - Todos: No se detectaron valores anómalos para esta característica.
    - *Beginner*: No se detectaron valores anómalos para esta característica.
    - *Expert*: No se detectaron valores anómalos para esta característica.
- Proporción de métodos asíncronos (Numérica):
    - Todos: Se detecta como anómalo cuando es superior a 0.
    - *Beginner*: Se detecta como anómalo cuando es superior a 0.
    - *Expert*: Se detecta como anómalo cuando es superior a 0.
- Proporción de métodos de clase (Numérica):
    - Todos: Se detecta como anómalo cuando es superior a 0.
    - *Beginner*: Se detecta como anómalo cuando es superior a 0.
    - *Expert*: Se detecta como anómalo cuando es superior a 0.
- Proporción de métodos estáticos (Numérica):
    - Todos: Se detecta como anómalo cuando es superior a 0.
    - *Beginner*: Se detecta como anómalo cuando es superior a 0.
    - *Expert*: Se detecta como anómalo cuando es superior a 0.
- Proporción de métodos abstractos (Numérica):
    - Todos: Se detecta como anómalo cuando es superior a 0.
    - *Beginner*: Se detecta como anómalo cuando es superior a 0.
    - *Expert*: Se detecta como anómalo cuando es superior a 0.
- Proporción de métodos de propiedad (Numérica):
    - Todos: Se detecta como anómalo cuando es superior a 0.
    - *Beginner*: Se detecta como anómalo cuando es superior a 0.
    - *Expert*: Se detecta como anómalo cuando es superior a 0.
- Convención de nombrado (Nominal):
    - Todos: En este caso la variable toma 7 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,028%. El valor predominante que toma esta variable es el de `CamelUp`. No se detectaron valores anómalos para esta característica.

  - *Beginner*: En este caso la variable toma 5 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,04%. El valor predominante que toma esta variable es el de `CamelUp`. No se detectaron valores anómalos para esta característica. Esta variable nunca toma los posibles valores `CamelLow` y `Discard`.
  - *Expert*: En este caso la variable toma 7 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,028%. El valor predominante que toma esta variable es el de `CamelUp`. No se detectaron valores anómalos para esta característica.
- Análisis Multivariante
  - Todos: Se detectaron 19 instancias anómalas.
  - *Beginner*: Se detectaron 3 instancias anómalas.
  - *Expert*: Se detectaron 18 instancias anómalas.

9.2.5 Function Definitions

- Si es una función privada (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
  - Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  - *Beginner*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  - *Expert*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
- Si es una función mágica (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
  - Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  - *Beginner*: El valor predominante es false. El valor true es anómalo.
  - *Expert*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
- Si es una función asíncrona (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
  - Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  - *Beginner*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  - *Expert*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
- Contiene anotación de tipo de retorno (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
  - Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  - *Beginner*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  - *Expert*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
- Contiene comentario de función (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.

  - Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  - *Beginner*: El valor predominante es true. No se detectaron valores anómalos para esta característica.
  - *Expert*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
- Número de caracteres del nombre (Numérica):
  - Todos: Se detecta como anómalo cuando es superior a 50.
  - *Beginner*: Se detecta como anómalo cuando es superior a 41.
  - *Expert*: Se detecta como anómalo cuando es superior a 62.
- Número de sentencias en el cuerpo de la función (Numérica):
  - Todos: Se detecta como anómalo cuando es superior a 18.
  - *Beginner*: Se detecta como anómalo cuando es superior a 14.
  - *Expert*: Se detecta como anómalo cuando es superior a 18.
- Número de decoradores (Numérica):
  - Todos: Se detecta como anómalo cuando es superior a 0.
  - *Beginner*: Se detecta como anómalo cuando es superior a 0.
  - *Expert*: Se detecta como anómalo cuando es superior a 4.
- Altura (Numérica):
  - Todos: Se detecta como anómalo cuando es superior a 1.
  - *Beginner*: Se detecta como anómalo cuando es superior a 1.
  - *Expert*: Se detecta como anómalo cuando es superior a 1.
- Proporción de sentencias en el cuerpo de la función que son expresiones (Numérica):
  - Todos: Se detecta como anómalo cuando es superior a 0,83.
  - *Beginner*: Se detecta como anómalo cuando es superior a 0,89.
  - *Expert*: Se detecta como anómalo cuando es superior a 0,83.
- Proporción de anotaciones de tipos (Numérica):
  - Todos: Se detecta como anómalo cuando es superior a 0,83.
  - *Beginner*: Se detecta como anómalo cuando es superior a 0.
  - *Expert*: No se detectaron valores anómalos para esta característica.
- Convención de nombrado (Nominal): En todos los casos esta variable toma 7 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,028%.
  - Todos: El valor predominante que toma esta variable es el de `SnakeCase`. No se detectaron valores anómalos para esta característica.
  - *Beginner*: El valor predominante que toma esta variable es el de `SnakeCase`. Los valores `Discard` y `Upper` son anómalos.
  - *Expert*: El valor predominante que toma esta variable es el de `SnakeCase`. No se detectaron valores anómalos para esta característica.
- Análisis Multivariante
  - Todos: Se detectaron 67 instancias anómalas.
  - *Beginner*: Se detectaron 20 instancias anómalas.
  - *Expert*: Se detectaron 38 instancias anómalas.

9.2.6 Method Definitions

- Si es un método de clase (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
  - Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.

    - *Beginner*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
    - *Expert*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
- Si es un método estático (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
    - Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
    - *Beginner*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
    - *Expert*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
- Si es un método constructor (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
    - Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
    - *Beginner*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
    - *Expert*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
- Si es un método abstracto (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
    - Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
    - *Beginner*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
    - *Expert*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
- Si es un método de propiedad (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
    - Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
    - *Beginner*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
    - *Expert*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
- Si es un método wrapper (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
    - Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
    - *Beginner*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
    - *Expert*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
- Si es un método cacheado (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
    - Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.

    - *Beginner*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
    - *Expert*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
- Si es un método privado (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
    - Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
    - *Beginner*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
    - *Expert*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
- Si es un método mágico (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
    - Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
    - *Beginner*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
    - *Expert*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
- Si es un método asíncrono (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
    - Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
    - *Beginner*: El valor predominante es false. El valor true es anómalo.
    - *Expert*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
- Contiene anotación de tipo de retorno (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
    - Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
    - *Beginner*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
    - *Expert*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
- Contiene un comentario de función (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
    - Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
    - *Beginner*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
    - *Expert*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
- Número de caracteres del nombre (Numérica):
    - Todos: Se detecta como anómalo cuando es superior a 40.
    - *Beginner*: Se detecta como anómalo cuando es superior a 32.
    - *Expert*: Se detecta como anómalo cuando es superior a 44.
- Número de sentencias en el cuerpo (Numérica):

  - Todos: Se detecta como anómalo cuando es superior a 76.
  - *Beginner*: Se detecta como anómalo cuando es superior a 35,1.
  - *Expert*: Se detecta como anómalo cuando es superior a 51,7.
- Número de decoradores (Numérica):
  - Todos: Se detecta como anómalo cuando es superior a 0.
  - *Beginner*: Se detecta como anómalo cuando es superior a 0.
  - *Expert*: Se detecta como anómalo cuando es superior a 0.
- Altura (Numérica):
  - Todos: Se detecta como anómalo cuando es superior a 2.
  - *Beginner*: Se detecta como anómalo cuando es superior a 2.
  - *Expert*: Se detecta como anómalo cuando es superior a 2.
- Proporción de sentencias en el cuerpo que son expresiones (Numérica):
  - Todos: No se detectaron valores anómalos para esta característica.
  - *Beginner*: Se detecta como anómalo cuando es superior a 0,83.
  - *Expert*: No se detectaron valores anómalos para esta característica.
- Proporción de anotaciones de tipos (Numérica):
  - Todos: No se detectaron valores anómalos para esta característica.
  - *Beginner*: Se detecta como anómalo cuando es superior a 0.
  - *Expert*: No se detectaron valores anómalos para esta característica.
- Convención de nombrado (Nominal):
  - Todos: En este caso la variable toma 7 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,028%. El valor predominante que toma esta variable es el de `SnakeCase`. El valor `Discard` es anómalo.
  - *Beginner*: En este caso la variable toma 6 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,033%. El valor predominante que toma esta variable es el de `SnakeCase`. No se detectaron valores anómalos para esta característica. Esta variable nunca toma el posible valor `Discard`.
  - *Expert*: En este caso la variable toma 7 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,028%. El valor predominante que toma esta variable es el de `SnakeCase`. El valor `Discard` es anómalo.
- Análisis Multivariante
  - Todos: Se detectaron 111 instancias anómalas.
  - *Beginner*: Se detectaron 39 instancias anómalas.
  - *Expert*: Se detectaron 69 instancias anómalas.

9.2.7 Statements

- Siendo una sentencia de la categoría correspondiente, si tiene una cláusula else (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
  - Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  - *Beginner*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  - *Expert*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
- Altura (Numérica):
  - Todos: Se detecta como anómalo cuando es superior a 10.
  - *Beginner*: Se detecta como anómalo cuando es superior a 10.

    - *Expert*: Se detecta como anómalo cuando es superior a 10.
- Profundidad (Numérica):
    - Todos: Se detecta como anómalo cuando es superior a 13.
    - *Beginner*: Se detecta como anómalo cuando es superior a 13.
    - *Expert*: Se detecta como anómalo cuando es superior a 13.
- Número de sentencias en el cuerpo de la sentencia, de haberlo. (Numérica):
    - Todos: Se detecta como anómalo cuando es superior a 49.
    - *Beginner*: Se detecta como anómalo cuando es superior a 26,5.
    - *Expert*: Se detecta como anómalo cuando es superior a 72,4.
- Categoría sintáctica (Nominal):
    - Todos: En este caso la variable toma 22 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,0090%. El valor predominante que toma esta variable es el de `AssingmentStmt`. Los valores `Match` y `Nonlocal` son anómalos. Esta variable nunca toma los posibles valores `ExceptHandler` y `TypeAlias`.
    - *Beginner*: En este caso la variable toma 18 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,01%. El valor predominante que toma esta variable es el de `AssingmentStmt`. No se detectaron valores anómalos para esta característica. Esta variable nunca toma los posibles valores `AsyncWith`, `AsyncFor`, `Match`, `TypeAlias`, `ExceptHandler` y `Nonlocal`.
    - *Expert*: En este caso la variable toma 22 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,0090%. El valor predominante que toma esta variable es el de `AssingmentStmt`. El valor `Match` es anómalo. Esta variable nunca toma los posibles valores `ExceptHandler` y `TypeAlias`.
- Categoría sintáctica del padre (Nominal):
    - Todos: En este caso la variable toma 12 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,016%. El valor predominante que toma esta variable es el de `FunctionDef`. No se detectaron valores anómalos para esta característica. Esta variable nunca toma los posibles valores `Raise`, `Return`, `Import`, `ImportFrom`, `Global`, `Nonlocal`, `AnnotatedAssignment`, `Pass`, `AssignmentStmt`, `Break`, `Delete`, `Continue`, `Assert`, `AugmentedAssignment`, `AsyncFor` y `TypeAlias`.
    - *Beginner*: En este caso la variable toma 10 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,02%. El valor predominante que toma esta variable es el de `FunctionDef`. No se detectaron valores anómalos para esta característica. Esta variable nunca toma los posibles valores `Raise`, `Return`, `Import`, `ImportFrom`, `Global`, `Nonlocal`, `AnnotatedAssignment`, `Pass`, `AssignmentStmt`, `Break`, `Delete`, `Continue`, `Assert`, `AugmentedAssignment`, `TypeAlias`, `AsyncWith`, `AsyncFor` y `Match`.
    - *Expert*: En este caso la variable toma 12 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,016%. El valor predominante que toma esta variable es el de `FunctionDef`. No se detectaron valores anómalos para esta característica. Esta variable nunca toma los posibles valores `Raise`, `Return`, `Import`, `ImportFrom`, `Global`, `Nonlocal`, `AnnotatedAssignment`, `Pass`, `AssignmentStmt`, `Break`, `Delete`, `Continue`, `Assert`, `AugmentedAssignment`, `AsyncFor` y `TypeAlias`.
- Rol de la sentencia (Nominal):

  - Todos: En este caso la variable toma 20 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,01%. El valor predominante que toma esta variable es el de `MethodDefBody`. El valor `WhileElseBody` es anómalo. Esta variable nunca toma los posibles valores `TryHandler`, `AsyncForElseBody` y `TryHandlerStar`.
  - *Beginner*: En este caso la variable toma 17 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,011%. El valor predominante que toma esta variable es el de `IfBody`. El valor AsyncMethodDefBody es anómalo. Esta variable nunca toma los posibles valores `TryHandler`, `AsyncForElseBody`, `TryHandlerStar`, `AsyncForBody`, `CaseBody` y `AsyncWithBody`.
  - *Expert*: En este caso la variable toma 20 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,01%. El valor predominante que toma esta variable es el de `MethodDefBody`. Los valores `WhileElseBody` y `ForElseBody` son anómalos. Esta variable nunca toma los posibles valores `TryHandler`, `AsyncForElseBody` y `TryHandlerStar`.
- Categoría sintáctica del primer hijo (Nominal):
  - Todos: En este caso la variable toma 34 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,0058%. El valor predominante que toma esta variable es el de `Variable`. Los valores `Shift`, `SetLiteral`, `SetComprehension`, `Pow`, `UnaryBWNot`, `MatMult` y `AssignmentExp` son anómalos. Esta variable nunca toma los posibles valores `FormattedValue`, `Star`, `YieldFrom`, `EllipsisLiteral`, `Yield`, `Parameter` y `Slice`.
  - *Beginner*: En este caso la variable toma 30 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,0066%. El valor predominante que toma esta variable es el de `Variable`. Los valores `ComplexLiteral`, `Shift`, `Pow`, `Await` y `SetComprehension` son anómalos.
  - *Expert*: En este caso la variable toma 34 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,0058%. El valor predominante que toma esta variable es el de Variable. Los valores `MatMult`, `UnaryBWNot`, `Shift`, `Pow`, `GeneratorComprehension`, `AssignmentExp`, `SetLiteral` y `SetComprehnsion` son anómalos. Esta variable nunca toma los valores `FormattedValue`, `Star`, `YieldFrom`, `EllipsisLiteral`, `Yield`, `Parameter` y `Slice`.
- Categoría sintáctica del segundo hijo (Nominal):
  - Todos: En este caso la variable toma 36 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,0055%. El valor predominante que toma esta variable es el de `None`. Los valores `EllipsisLiteral`, `YieldFrom`, `Yield` y `UnaryBWNot` son anómalos. Esta variable nunca toma los posibles valores `AssignmentExp`, `FormattedValue`, `Star`, `Parameter` y `Slice`.
  - *Beginner*: En este caso la variable toma 33 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,006%. El valor predominante que toma esta variable es el de `None`. Los valores `Yield`, `Await` y `UnaryBWNot` son anómalos. Esta variable nunca toma los posibles valores `YieldFrom`, `AssignmentExp`, `MatMult`, `Star`, `Parameter`, `EllipsisLiteral`, `FormattedValue` y `Slice`.
  - *Expert*: En este caso la variable toma 36 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,0055%. El valor predominante que toma esta variable es el de `None`. Los valors `EllipsisLiteral`, `Yield`,

`YieldFrom` y `UnaryBWNot` son anómalos. Esta variable nunca toma los posibles valores `AssignmentExp`, `FormattedValue`, `Star`, `Parameter` y `Slice`.

- Categoría sintáctica del tercer hijo (Nominal):
  - Todos: En este caso la variable toma 30 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,0066%. El valor predominante que toma esta variable es el de None. Los valores `Pow`, `ComplexLiteral`, `UnaryNot`, `SetComprehension`, `BWLogical`, `FString`, `GeneratorComprehension`, `Compare`, `SetLiteral`, `UnaryArithmentic`, `Logical`, `Lambda`, `DictComprehension`, `Await`, `ListComprehension`, `Arithmetic` y `Ternary` son anómalos. Esta variable nunca toma los posibles valores `AssignmentExp`, `Shift`, `FormattedValue`, `Star`, `MatMult`, `YieldFrom`, `Yield`, `EllipsisLiteral`, `UnaryBWNot`, `Parameter` y `Slice`.
  - *Beginner*: En este caso la variable toma 23 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,0087%. El valor predominante que toma esta variable es el de None. Los valores `SetLiteral`, `FloatLiteral`, `UnaryArithmetic`, `Logical`, `BWLogical`, `BoolLiteral`, `ListComprehension`, `DictComprehension`, `GeneratorComprehension`, `Ternary`, `Lambda` y `Arithmetic` son anómalos. Esta variable nunca toma los valores `UnaryBWNot`, `YieldFrom`, `Shift`, `AssignmentExp`, `MatMult`, `Star`, `Parameter`, `Pow`, `Yield`, `SetComprehension`, `ComplexLiteral`, `FString`, `UnaryNot`, `EllipsisLiteral`, `Compare`, `FormatterValue`, `Await` y `Slice`.
  - *Expert*: En este caso la variable toma 28 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,0071%. El valor predominante que toma esta variable es el de None. Los valores `ComplexLiteral`, `Pow`, `UnaryNot`, `Lambda`, `SetComprehension`, `FString`, `SetLiteral`, `UnaryArithmetic`, `Compare`, `Logical`, `DictComprehension`, `Arithmetic`, `TupleLiteral`, `ListComprehension` y `Ternary` son anómalos. Esta variable nunca toma los posibles valores `YieldFrom`, `EllipsisLiteral`, `Star`, `Yield`, `FormattedValue`, `UnaryBWNot`, `Slice`, `AssignmentExp`, `Parameter`, `MatMult`, `GeneratorComprehension`, `BWLogical` y `Shift`.
- Análisis Multivariante
  - Todos: Se detectaron 1.278 instancias anómalas.
  - *Beginner*: Se detectaron 545 instancias anómalas.
  - *Expert*: Se detectaron 740 instancias anómalas.

9.2.8 Cases

- Número de cláusulas case en el Match. (Numérica):
  - Todos: Se detecta como anómalo cuando es superior a 10.
- Número guards. (Numérica):
  - Todos: No se detectaron valores anómalos para esta característica.
- Número medio de sentencias en el cuerpo de los case. (Numérica):
  - Todos: No se detectaron valores anómalos para esta característica.
- Número de cases del tipo MatchValue. (Numérica):
  - Todos: No se detectaron valores anómalos para esta característica.
- Número de cases del tipo MatchSingleton. (Numérica):
  - Todos: No se detectaron valores anómalos para esta característica.
- Número de cases del tipo MatchSequence. (Numérica):
  - Todos: No se detectaron valores anómalos para esta característica.

- Número de cases del tipo MatchMapping. (Numérica):
    - Todos: No se detectaron valores anómalos para esta característica.
- Número de cases del tipo MatchStar. (Numérica):
    - Todos: No se detectaron valores anómalos para esta característica.
- Número de cases del tipo MatchOr. (Numérica):
    - Todos: No se detectaron valores anómalos para esta característica.
- Número de cases del tipo MatchClass. (Numérica):
    - Todos: Se detecta como anómalo cuando es superior a 0.
- Número de cases del tipo MatchAs. (Numérica):
    - Todos: No se detectaron valores anómalos para esta característica.
- Análisis Multivariante
    - Todos: Se detectó 1 instancias anómala.
    - Todos: Se detectó 1 instancias anómala.
    - *Expert*: No hay instancias para analizar.

9.2.9 Handlers

- Contiene un handler star, es decir, es un TryStar (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
    - Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
    - *Beginner*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
    - *Expert*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
- Contiene un handler que capture todas las excepciones (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
    - Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
    - *Beginner*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
    - *Expert*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
- Contiene una cláusula finally (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
    - Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
    - *Beginner*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
    - *Expert*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
- Número de handlers. (Numérica):
    - Todos: Se detecta como anómalo cuando es superior a 1 o es inferior a 1.
    - *Beginner*: Se detecta como anómalo cuando es superior a 1 o es inferior a 1.
    - *Expert*: Se detecta como anómalo cuando es superior a 1 o es inferior a 1.
- Número medio de sentencias en el cuerpo de los handlers. (Numérica):
    - Todos: Se detecta como anómalo cuando es superior a 1 o es inferior a 1.
    - *Beginner*: Se detecta como anómalo cuando es superior a 1 o es inferior a 1.
    - *Expert*: Se detecta como anómalo cuando es superior a 5.

- Análisis Multivariante
    - Todos: Se detectaron 16 instancias anómalas.
    - *Beginner*: Se detectaron 7 instancias anómalas.
    - *Expert*: Se detectaron 9 instancias anómalas.

9.2.10 Expressions

- Altura (Numérica):
    - Todos: Se detecta como anómalo cuando es superior a 12.
    - *Beginner*: Se detecta como anómalo cuando es superior a 12.
    - *Expert*: Se detecta como anómalo cuando es superior a 12.
- Profundidad (Numérica):
    - Todos: Se detecta como anómalo cuando es superior a 4.
    - *Beginner*: Se detecta como anómalo cuando es superior a 4.
    - *Expert*: Se detecta como anómalo cuando es superior a 4.
- Altura (Numérica):
    - Todos: Se detecta como anómalo cuando es superior a 12.
    - *Beginner*: Se detecta como anómalo cuando es superior a 12.
    - *Expert*: Se detecta como anómalo cuando es superior a 12.
- Profundidad (Numérica):
    - Todos: Se detecta como anómalo cuando es superior a 4.
    - *Beginner*: Se detecta como anómalo cuando es superior a 4.
    - *Expert*: Se detecta como anómalo cuando es superior a 4.
- Categoría sintáctica (Nominal):
    - Todos: En este caso la variable toma 39 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,0051%. El valor predominante que toma esta variable es el de `Variable`. Los valores `AssignmentExp`, `SetComprehension`, `MatMult`, `YieldFrom` y `UnaryBWNot` son anómalos. Esta variable nunca toma el posible valor `NoneType`.
    - *Beginner*: En este caso la variable toma 38 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,0052%. El valor predominante que toma esta variable es el de `Variable`. Los valores `AssignmentExp`, `SetComprehension` y `Await` son anómalos. Esta variable nunca toma los posibles valores `MatMult` y `NoneType`.
    - *Expert*: En este caso la variable toma 39 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,0051%. El valor predominante que toma esta variable es el de `Variable`. Los valores `AssingmentExp`, `UnaryBWNot`, `MatMult` y `YieldFrom` son anómalos. Esta variable nunca toma el posible valor `NoneType`.
- Categoría sintáctica del padre (Nominal):
    - Todos: En este caso la variable toma 51 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,0039%. El valor predominante que toma esta variable es el de `Call`. Los valores `AsyncWith`, `AssignmentExp`, `Match`, `UnaryBWNot` y `MatMult` son anómalos. Esta variable nunca toma los posibles valores `Import`, `Break`, `Nonlocal`, `StringLiteral`, `IntLiteral`, `FloatLiteral`, `Global`, `Variable`, `NoneLiteral`, `EllipsisLiteral`, `Continue`, `NoneType`, `TypeAlias`, `ComplexLiteral`, `BoolLiteral`, `ImportFrom` y `Pass`.

- *Beginner*: En este caso la variable toma 47 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,0042%. El valor predominante que toma esta variable es el de `Call`. Los valores `AssignmentExp` y `Await` son anómalos. Este valor nunca toma los posibles valores `NoneLiteral`, `Import`, `Global`, `FloatLiteral`, `NoneType`, `EllipsisLiteral`, `TypeAlias`, `Match`, `Pass`, `Continue`, `Nonlocal`, `ComplexLiteral`, `ImportFrom`, `MatMult`, `Variable`, `Break`, `StringLiteral`, `BoolLiteral`, `IntLiteral`, `AsyncFor` y `AsyncWith`.
- *Expert*: En este caso la variable toma 51 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,0039%. El valor predominante que toma esta variable es el de `Call`. Los valores `AssignmentExp`, `UnaryBWNot`, `Match` y `YieldFrom` son anómalos. Esta variable nunca toma los posibles valores `ComplexLiteral`, `Global`, `ImportFrom`, `Continue`, `StringLiteral`, `NoneLiteral`, `Nonlocal`, `FloatLiteral`, `EllipsisLiteral`, `Import`, `NoneType`, `Break`, `IntLiteral`, `Pass`, `TypeAlias`, `BoolLiteral` y `Variable`.

– Categoría sintáctica del primer hijo (Nominal):
  - Todos: En este caso la variable toma 39 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,0051%. El valor predominante que toma esta variable es el de `None`. Los valores `AssignmentExp`, `SetComprehension`, `DictComprehension`, `NoneType`, `Shift`, `Lambda`, `GeneratorComprehension`, `MatMult`, `Await`, `UnaryBWNot`, `Star` y `EllipsisLiteral` son anómalos. Esta variable nunca toma los posibles valores `YieldFrom`, `Yield` y `Parameter`.
  - *Beginner*: En este caso la variable toma 34 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,0059%. El valor predominante que toma esta variable es el de `None`. Los valores `AssignmentExp`, `SetComprehension`, `NoneType`, `Star`, `GeneratorComprehension`, `ListComprehension`, `UnaryBWNot`, `Shift`, `FString` y `DictionaryLiteral` son anómalos. Esta variable nunca toma los posibles valores `Parameter`, `DictComprehension`, `Await`, `Yield`, `YieldFrom`, `Slice`, `Lambda` y `MatMult`.
  - *Expert*: En este caso la variable toma 38 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,0052%. El valor predominante que toma esta variable es el de `None`. Los valores `UnaryBWNot`, `SetComprehension`, `DictComprehension`, `GeneratorComprehension`, `Lambda`, `Await`, `Star`, `MatMult`, `EllipsisLiteral` y `Shift` son anómalos. Esta variable nunca toma los posibles valores `YieldFrom`, `Parameter`, `YIeld` y `AssignmentExp`.

– Categoría sintáctica del segundo hijo (Nominal):
  - Todos: En este caso la variable toma 37 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,0054%. El valor predominante que toma esta variable es el de `None`. Los valores `AssignmentExp`, `Await`, `SetComprehension`, `Shift`, `DictComprehension`, `UnaryBWNot` y `SetLiteral` son anómalos. Esta variable nunca toma los posibles valores `Parameter`, `YieldFrom`, `MatMult`, `NoneType` y `Yield`.
  - *Beginner*: En este caso la variable toma 36 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,0055%. El valor predominante que toma esta variable es el de `None`. Los valores `AssignmentExp`, `Shift`, `DictComprehension`, `SetLiteral`, `SetComprehension`, `UnaryBWNot` y

EllipsisLiteral son anómalos. Esta variable nunca toma los posibles valores Parameter, Await, Yield, YieldFrom, NoneType y MatMult.

- *Expert*: En este caso la variable toma 37 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,0054%. El valor predominante que toma esta variable es el de None. Los valores AssignmentExp, SetComprehension, UnaryBWNot, Await, Shift y DictComprehension son anómalos. Esta variable nunca toma los posibles valores MatMult, YieldFrom, NoneTyp, Parameter y Yield.

- Categoría sintáctica del tercer hijo (Nominal):
  - Todos: En este caso la variable toma 38 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,0052%. El valor predominante que toma esta variable es el de None. Los valores SetComprehension, UnaryBWNot, AssignmentExp, Shift, EllipsisLiteral, Await, DictComprehension, GeneratorComprehension, Pow, SetLiteral, ListComprehension, BWLogical y NoneType son anómalos. Esta variable nunca toma los posibles valores YieldFrom, Yield, Parameter y MatMult.
  - *Beginner*: En este caso la variable toma 32 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,00625%. El valor predominante que toma esta variable es el de None. Los valores AssignmentExp, DictComprehension, Pow, NoneType, Setliteral, ListComprehension, GeneratorComprehension, ComplexLiteral, FString, Ternary, Lambda y FloatLiteral son anómalos. Esta variable nunca toma los posibles valores Parameter, Await, Shift, Yield, UnaryBWNot, YieldFrom, EllipsisLiteral, Slice, SetComprehension y MatMult.
  - *Expert*: En este caso la variable toma 38 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,0052%. El valor predominante que toma esta variable es el de None. Los valores AssignmentExp, SetComprehension, UnaryBWNot, Shift, GeneratorComprehension, EllipsisLiteral, Await, DictComprehension, BWLogical, Pow, SetLiteral y ListComprehension son anómalos. Esta variable nunca toma los posibles valores YieldFrom, Parameter, Yield y MatMult.
- Categoría sintáctica del cuarto hijo (Nominal):
  - Todos: En este caso la variable toma 35 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,0057%. El valor predominante que toma esta variable es el de None. Los valores Await, SetComprehension, EllipsisLiteral, Shift, GeneratorComprehension, DictComprehension, BWLogical, Pow, SetLiteral, ListComprehension, ComplexLiteral, Slice, Star, Ternary, Lambda, FString y UnaryNot son anómalos. Esta variable nunca toma los posibles valores Parameter, YieldFrom, MatMult, UnaryBWNot, NoneType, AssignmentExp y Yield.
  - *Beginner*: En este caso la variable toma 29 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,0069%. El valor predominante que toma esta variable es el de None. Los valores DictComprehension, ListComprehension, SetLiteral, GeneratorComprehension, ComplexLiteral, BWLogical, FString, Ternary, FloatLiteral, Lambda, Star, UnaryNot y DictionaryLiteral son anómalos. Esta variable nunca toma los posibles valores Parameter, Await, Shift, Yield, UnaryBWNot, YieldFrom,

EllipsisLiteral, Slice, Pow, AssignmentExp, NoneType, SetComprehension y MatMult.

- *Expert*: En este caso la variable toma 34 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,0059%. El valor predominante que toma esta variable es el de None. Los valores SetComprehension, Await, EllipsisLiteral, Shift, DictComprehension, BWLogical, Star, SetLiteral, Pow, ComplexLiteral, ListComprehension, Slice, UnaryNot, Ternary, Lambda, FString y Logical son anómalos. Esta variable nunca toma los posibles valores MatMult, GeneratorComprehension, YieldFrom, NoneType, Parameter, UnaryBWNot, Yield y AssignmentExp.

– Rol de la expresión en su padre (Nominal):
  - Todos: En este caso la variable toma 79 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,0025%. El valor predominante que toma esta variable es el de CallArg. Los valores CaseBody, WhileElseBody, MatchCondition, AsyncWithBody, AssignExpLHS, AssignExpRHS, ForElseBody, AsyncWithAs, CaseCondition, AsyncForBody, AsyncWithElement, AsyncForEnumerable y AsyncForElement son anómalos. Esta variable nunca toma los posibles valores TryElse, ComprehensionElement, AsyncForElseBody, MatMult, DefaultParamValue, TypeVar, In, Shift, AugmentedAssignmentLHS, TypeAliasLHS, CaseGuard, Pow, Is, AugmentedAssignmentRHS, TypeAliasRHS y BWLogical.
  - *Beginner*: En este caso la variable toma 70 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,0028%. El valor predominante que toma esta variable es el de TupleLiteral. Los valores WhileElseBody, ClassDecorator, AssignExpRHS, AssignExpLHS, ForElseBody y FormattedFormat son anómalos. Esta variable nunca toma los posibles valores AsyncWithElement, AugmentedAssignmentLHS, Shift, DefaultParamValue, MatchCondition, AsyncForBody, BWLogical, TryElse, In, Pow, TypeVar, AsyncWithAs, CaseBody, MatMult, CaseCondition, ComprehensionElement, AugmentedAssignmentRHS, TypeAnnotation, AsyncForEnumerable, CaseGuard, AsyncForElseBody, TypeAliasRHS, TypeAliasLHS, AyncWithBody, Is y AsyncForElement.
  - Expert: En este caso la variable toma 79 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,0025%. El valor predominante que toma esta variable es el de CallArg. Los valores WhileElseBody, CaseBody, ForElseBody, MatchCondition, AsyncWithBody, AssignExpRHS, AssignExpLHS, TryElseBody y AsyncWithAs son anómalos. Esta variable nunca toma los posibles valores TypeVar, MatMult, BWLogical, CaseGuard, AugmentedAssignmentLHS, TryElse, TypeAnnotation, TypeAliasRHS, AsyncForElseBody, Is, TypeAliasLHS, Pow, DefaultParamValue, AugmentedAssignmentRHS, ComprehensionElement, In y Shift.

– Análisis Multivariante
  - Todos: Se detectaron 9.929 instancias anómalas.
  - *Beginner*: Se detectaron 4.030 instancias anómalas.
  - *Expert*: Se detectaron 5.751 instancias anómalas.

9.2.11 Comprehensions

– Si es asíncrono (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.

  - Todos: El valor predominante es false. El valor true es anómalo.
  - *Beginner*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  - *Expert*: El valor predominante es false. El valor true es anómalo.
- Número de sentencias condicionales (Numérica):
  - Todos: Se detecta como anómalo cuando es superior a 0.
  - *Beginner*: No se detectaron valores anómalos para esta característica.
  - *Expert*: Se detecta como anómalo cuando es superior a 0.
- Número de generadores (Numérica):
  - Todos: Se detecta como anómalo cuando es superior a 1.
  - *Beginner*: Se detecta como anómalo cuando es superior a 1.
  - *Expert*: Se detecta como anómalo cuando es superior a 1.
- Categoría sintáctica (Nominal): En todos los casos esta variable toma 4 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,05%.
  - Todos: El valor predominante que toma esta variable es el de `ListComprehension`. No se han detectado valores anómalos para esta característica.
  - *Beginner*: El valor predominante que toma esta variable es el de `GeneratorComprehension`. No se han detectado valores anómalos para esta característica.
  - *Expert*: El valor predominante que toma esta variable es el de `ListComprehension`. No se han detectado valores anómalos para esta característica.
- Análisis Multivariante
  - Todos: Se detectaron 21 instancias anómalas.
  - *Beginner*: Se detectaron 6 instancias anómalas.
  - *Expert*: Se detectaron 6 instancias anómalas.

9.2.12 CallArgs

- Número de argumentos (Numérica):
  - Todos: Se detecta como anómalo cuando es superior a 5.
  - *Beginner*: Se detecta como anómalo cuando es superior a 5.
  - *Expert*: Se detecta como anómalo cuando es superior a 5.
- Proporción de argumentos pasados por nombre (Numérica):
  - Todos: Se detecta como anómalo cuando es superior a 0.
  - *Beginner*: Se detecta como anómalo cuando es superior a 0.
  - *Expert*: Se detecta como anómalo cuando es superior a 0.
- Proporción de argumentos double star (Numérica):
  - Todos: Se detecta como anómalo cuando es superior a 0.
  - *Beginner*: Se detecta como anómalo cuando es superior a 0.
  - *Expert*: Se detecta como anómalo cuando es superior a 0.
- Análisis Multivariante
  - Todos: Se detectaron 1.027 instancias anómalas.
  - *Beginner*: Se detectaron 406 instancias anómalas.
  - *Expert*: Se detectaron 569 instancias anómalas.

9.2.13 FStrings

- Número de elementos (Numérica):
  - Todos: Se detecta como anómalo cuando es superior a 10.
  - *Beginner*: Se detecta como anómalo cuando es superior a 10.

  - *Expert*: Se detecta como anómalo cuando es superior a 10.
- Proporción de constantes (Numérica):
  - Todos: Se detecta como anómalo cuando es superior a 0,91 o cuando es inferior a 0,25.
  - *Beginner*: Se detecta como anómalo cuando es superior a 0,91 o cuando es inferior a 0,25.
  - *Expert*: Se detecta como anómalo cuando es superior a 0,91 o cuando es inferior a 0,25.
- Proporción de expresiones (Numérica):
  - Todos: Se detecta como anómalo cuando es superior a 0,75 o cuando es inferior a 0,08.
  - *Beginner*: Se detecta como anómalo cuando es superior a 0,75 o cuando es inferior a 0,08.
  - *Expert*: Se detecta como anómalo cuando es superior a 0,75 o cuando es inferior a 0,08.
- Análisis Multivariante
  - Todos: No se detectaron instancias anómalas.
  - *Beginner*: Se detectaron 17 instancias anómalas.
  - *Expert*: Se detectaron 9 instancias anómalas.

9.2.14 Variables

- Si es privada (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
  - Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  - *Beginner*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  - *Expert*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
- Si es mágica (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
  - Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  - *Beginner*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  - *Expert*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
- Número de caracteres (Numérica):
  - Todos: Se detecta como anómalo cuando es superior a 28.
  - *Beginner*: Se detecta como anómalo cuando es superior a 20.
  - *Expert*: Se detecta como anómalo cuando es superior a 28.
- Convención de nombrado (Nominal): En todos los casos esta variable toma 7 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,028%.
  - Todos: El valor predominante es Lower. No se detectaron valores anómalos para esta característica.
  - *Beginner*: El valor predominante es Lower. No se detectaron valores anómalos para esta característica.

  - *Expert*: El valor predominante es Lower. No se detectaron valores anómalos para esta característica.
- Análisis Multivariante
  - Todos: Se detectaron 3.389 instancias anómalas.
  - *Beginner*: Se detectaron 1.329 instancias anómalas.
  - *Expert*: Se detectaron 2.015 instancias anómalas.

9.2.15 Vectors

- Si es homogénea (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
  - Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  - *Beginner*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  - *Expert*: El valor predominante es true. No se detectaron valores anómalos para esta característica.
- Número de elementos (Numérica):
  - Todos: Se detecta como anómalo cuando es superior a 81 o cuando es inferior a 1,9.
  - *Beginner*: Se detecta como anómalo cuando es superior a 77 o cuando es inferior a 1,9.
  - *Expert*: Se detecta como anómalo cuando es superior a 2 o cuando es inferior a 2. **
- Categoría sintáctica (Nominal): En todos los casos esta variable toma 4 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,05%.
  - Todos: El valor predominante que toma esta variable es el de `TupleLiteral`. No se han detectado valores anómalos para esta característica.
  - *Beginner*: El valor predominante que toma esta variable es el de `TupleLiteral`. No se han detectado valores anómalos para esta característica.
  - *Expert*: El valor predominante que toma esta variable es el de `TupleLiteral`. No se han detectado valores anómalos para esta característica.
- Análisis Multivariante
  - Todos: Se detectaron 64 instancias anómalas.
  - *Beginner*: Se detectaron 145 instancias anómalas.
  - *Expert*: Se detectaron 276 instancias anómalas.

9.2.16 Parameters

- Contiene al menos un var param (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
  - Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  - *Beginner*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  - *Expert*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
- Contiene al menos un keyword param (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
  - Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.

    - *Beginner*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
    - *Expert*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
- Número de parámetros (Numérica):
    - Todos: Se detecta como anómalo cuando es superior a 11 o inferior a 1.
    - *Beginner*: Se detecta como anómalo cuando es superior a 9 o inferior a 1.
    - *Expert*: Se detecta como anómalo cuando es inferior a 1.
- Proporción de argumentos posicionales (Numérica):
    - Todos: Se detecta como anómalo cuando es superior a 0.
    - *Beginner*: Se detecta como anómalo cuando es superior a 0.
    - *Expert*: Se detecta como anómalo cuando es superior a 0.
- Proporción de anotaciones de tipos (Numérica):
    - Todos: Se detecta como anómalo cuando es superior a 0.
    - *Beginner*: Se detecta como anómalo cuando es superior a 0.
    - *Expert*: Se detecta como anómalo cuando es superior a 0.
- Proporción de var params (Numérica):
    - Todos: Se detecta como anómalo cuando es superior a 0.
    - *Beginner*: Se detecta como anómalo cuando es superior a 0.
    - *Expert*: Se detecta como anómalo cuando es superior a 0.
- Proporción de argumentos por keyword (Numérica):
    - Todos: Se detecta como anómalo cuando es superior a 0.
    - *Beginner*: Se detecta como anómalo cuando es superior a 0.
    - *Expert*: Se detecta como anómalo cuando es superior a 0.
- Proporción de argumentos con valor por defecto (Numérica):
    - Todos: Se detecta como anómalo cuando es superior a 0.
    - *Beginner*: Se detecta como anómalo cuando es superior a 0.
    - *Expert*: Se detecta como anómalo cuando es superior a 0.
- Rol de los parámetros (Nominal): En todos los casos esta variable toma 2 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,1%.
    - Todos: El valor predominante es FunctionParams. No se detectaron valores anómalos para esta característica.
    - *Beginner*: El valor predominante es FunctionParams. No se detectaron valores anómalos para esta característica.
    - *Expert*: El valor predominante es FunctionParams. No se detectaron valores anómalos para esta característica.
- Convención de nombrado más usada (Nominal): En todos los casos esta variable toma 7 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,028%.
    - Todos: El valor predominante es Lower. No se detectaron valores anómalos para esta característica.
    - *Beginner*: El valor predominante es Lower. No se detectaron valores anómalos para esta característica.
    - *Expert*: El valor predominante es Lower. No se detectaron valores anómalos para esta característica.
- Análisis Multivariante
    - Todos: Se detectaron 171 instancias anómalas.
    - *Beginner*: Se detectaron 63 instancias anómalas.

- *Expert*: Se detectaron 74 instancias anómalas.

9.3 Repositorios GitHub

En este anexo se van a listar y describir resumidamente los repositorios de *GitHub* utilizados como fuentes de código de expertos.

NOMBRE	CONTRIBUIDORES	FORKS	STARS	USOS
ray-project/ray	974	5.300	31.607	15.000
certbot/certbot	478	3.400	30.992	1.900
run-llama/llama_index	994	4.500	32.387	8.600
sqlmapproject/sqlmap	125	5.600	30.910	X*
geekcomputers/Python	691	12.000	30.340	X*
huggingface/pytorch-image-models	120	4.600	30.281	28.900
babysor/MockingBird	37	5.100	34.174	X*
testerSunshine/12306	23	9.800	33.723	X*
lm-sys/FastChat	249	4.300	35.098	740
shadowsocks/shadowsocks	X*	18.700	33.522	X*
XX-net/XX-Net	73	7.700	32.736	X*
microsoft/DeepSpeed	341	3.900	33.252	7.100
fxsjy/jieba	40	6.700	32.602	28.000
comfyanonymous/ComfyUI	111	3.900	36.823	X*
hankcs/HanLP	36	9.700	32.710	184
httpie/cli	155	3.700	32.400	X*
karpathy/nanoGPT	36	5.000	32.735	X*
streamlit/streamlit	227	2.800	32.589	395.000

* Esta información no consta en la descripción del repositorio a la fecha de realización del trabajo.

9.4 Referencias a los Notebooks

A continuación, se lista todos los Jupyter Notebooks utilizados para el análisis de outliers. Cada uno de ellos están almacenados en el repositorio de GitHub del proyecto.

9.4.1 Programs

- 01_programs
- 01_programs_beginner
- 01_programs_expert

9.4.2 Modules

- 02_modules
- 02_modules_beginner
- 02_modules_expert

9.4.3 Imports

- 03_imports
- 03_imports_beginner
- 03_imports_expert

9.4.4 Classdefs

- 04_class_defs
- 04_class_defs_beginner
- 04_class_defs_expert

9.4.5 Functiondefs

- 05_function_defs
- 05_function_defs_beginner
- 05_function_defs_expert

9.4.6 Methoddefs

- 06_method_defs
- 06_method_defs_beginner
- 06_method_defs_expert

9.4.7 Statements

- 07_statements
- 07_statements_beginner
- 07_statements_expert

9.4.8 Cases

- 08_cases

9.4.9 Handlers

- 09_handlers
- 09_handlers_beginner
- 09_handlers_expert

9.4.10 Expressions

- 10_expressions
- 10_expressions_beginner

- 10_expressions_expert

9.4.11 Comprehensions

- 11_comprehensions
- 11_comprehensions_beginner
- 11_comprehensions_expert

9.4.12 Callargs

- 12_call_args
- 12_call_args_beginner
- 12_call_args_expert

9.4.13 Fstrings

- 13_fstrings
- 13_fstrings_beginner
- 13_fstrings_expert

9.4.14 Variables

- 14_variables
- 14_variables_beginner
- 14_variables_expert

9.4.15 Vectors

- 15_vectors
- 15_vectors_beginner
- 15_vectors_expert

9.4.16 Parameters

- 16_parameters
- 16_parameters_beginner
- 16_parameters_expert