Python Source Code Analysis

Objective

Create a dataset for Python source code and fill it in with many projects. The code will be written by:

a) Novice programmers
b) Expert developers.

Once the database has been filled in, different data-mining algorithms could be executed. Examples are anomaly detection, frequency analysis, feature reduction and visualization, logistic regression, classification rules, clustering, and association rules.

Program insertion should be efficient, given the number of projects to be inserted. It must also be maintainable to allow its easy modification and extension.

Python references

First, try to be fluent as a Python programmer. This is a standard tutorial.

https://docs.python.org/3/tutorial/index.html

Python provides AST creation and traversal. Here, you can find a brief explanation.

https://www.scaler.com/topics/python-ast/

The Python ast module is an important resource for checking the structure of Python ASTs.

https://docs.python.org/3/library/ast.html

Database

The fields annotated with ψ should not be considered in the machine-learning or data-mining algorithms. The reason is that they correlate with problem complexity more than with programmer expertise.

What follows is an enumeration of the different tables to be defined.

Projects:

Name	**Description**	**Domain**
ProjectID$^{\psi}$	Primary key of the project	Unique ID (Integer)
Name$^{\psi}$	Project name	String
HasSubDirsWithCode	Within the root source code directories, there are subdirectories with Python code (.py files) *but* there is not an __init__.py file in it	Boolean

Name	Description	Domain
HasPackages	Within the root source code directories, there are subdirectories with Python code (.py files) and an __init__.py file in it	Boolean
NumberOfModules[Ψ]	Number of .py files in the project	Integer
NumberOfSubDirsWithCode[Ψ]	Number of subdirectories in the project with modules inside of them *but* without an __init__.py file	Integer
NumberOfPackages[Ψ]	Number of subdirectories in the project with modules inside of them and an __init__.py file in it	Integer
ClassDefsPct	Proportion of definitions (functions, enums, or classes) that are classes	Real [0, 1]
FunctionDefsPct	Proportion of definitions (functions, enums or classes) that are functions	Real [0, 1]
EnumDefsPct	Proportion of classes derived from Enum	Real [0, 1]
HasCodeRootPackage	Whether the project has code in the root package	Boolean
AverageDefsPerModule	Average number of definitions (enum, class and function) per module	Real
IsExpert	Whether the programmer is expert or beginner	Boolean

Modules (in Python, a module is a .py file):

Name	Description	Domain
ModuleID[Ψ]	Primary key of the module	Unique ID (Integer)
Name[Ψ]	Module name	String
Path[Ψ]	Path to the .py file	String
NameConvention	Its naming convention	NamingConvention[2]
HasDocString	Whether the module has a module comment. It is true when its first entry in `body` is a `Constant` with a string `value`	Boolean
GlobalStmtsPct	Proportion of statements written in the global scope (imports, class and function definitions are not considered)	Real [0, 1]
GlobalExpressions	Proportion of expressions in the global scope (most of them would be statements)	Real [0, 1]
NumberOfClasses[Ψ]	Number of classes defined in the module	Integer
NumberOfFunctions[Ψ]	Number of functions defined in the module	Integer
ClassDefsPct	Proportion of definitions (functions, enums, or classes) that are classes	Real [0, 1]

Name	Description	Domain
FunctionDefsPct	Proportion of definitions (functions, enums or classes) that are functions	Real [0, 1]
EnumDefsPct	Proportion of classes derived from Enum	Real [0, 1]
AverageStmtsFunctionBody	Average number of statements in the function bodies	Real
AverageStmtsMethodBody	Average number of statements in the method bodies	Real
TypeAnnotationPct	Proportion of type annotations for the function and method parameters and return values	Real [0, 1]
HasEntryPoint	If the module has the Python idiom `if __name__ == '__main__'`	Boolean
Path	Module path inside the project	String
ProjectID$^{\Psi}$	Primary key of the associated project	Unique ID (Integer)
ImportID$^{\Psi}$	Primary key where imports were written if any	Unique ID (Integer)

Imports (one entry per module/file):

Name	Description	Domain
ImportID$^{\Psi}$	Primary key of the import	Unique ID (Integer)
NumberImports$^{\Psi}$	Number of imports in a module	Integer
ModuleImportsPct	Proportion of simple imports (`Import` nodes)	Real [0, 1]
AverageImportedModules$^{\Psi}$	Average of modules (`alias`) imported in the same import statement	Real
FromImportsPct	Proportion of "from" imports (`ImportFrom` nodes)	Real [0, 1]
AverageFromImportedModules$^{\Psi}$	Average of modules (`alias`) imported in the same from import statement	Real
AverageAsInImportedModules	Average of "as" aliases (`asname`) in the same from import statement, relative to the imported names (`alias`)	Real
LocalImportsPct	Proportion of imports (any type) not written at the beginning of the module	Real [0, 1]

ClassDefs:

Name	Description	Domain
ClassDefID$^{\Psi}$	Primary key of the class definition	Unique ID (Integer)
NameConvention	Its naming convention	NamingConvention[2]
IsEnumClass	Whether the class inherits from Enum	Boolean

Name	Description	Domain
NumberOfCharacters	The number of characters of the function ID	Integer
NumberOfDecorators	Number of decorators	Integer
NumberOfMethods	Number of methods	Integer
NumberOfBaseClasses	Number of base classes	Integer
HasGenericTypeAnnotations	In Python 3.12, classes can have generic type annotations (e.g., `class list[T]: pass`). These type annotations can be obtained in the `type_params` field of the `ClassDef` ast node.	Boolean
HasDocString	Whether the class has a class comment. It is true when its first entry in `body` is a `Constant` with a string `value`	Boolean
BodyCount	Number of statements in the class body (not inside of the methods)	Integer
AssignmentsPct	Proportion of assignments in the class body (not inside of the methods); this kind of assignment is an alternative way to define fields	Real [0, 1]
ExpressionsPct	Proportion of expressions in its body (most of them would be statements)	Real [0, 1]
UsesMetaclass	Whether the class uses a metaclass (defines “meta=” in the inheritance clause)	Boolean
NumberOfKeyWords	Number other keywords, different to metaclass	Integer
Height	Distance (number of edges) from the current node to the root node in the enclosing module (not program)	Integer
AverageStmtsMethodBody	Average number of statements in the method bodies	Real
TypeAnnotationsPct	Proportion of type annotations for the method parameters and return values	Real [0, 1]
PrivateMethodsPct	Proportion of private methods (starting with “_”) in the class	Real [0, 1]
MagicMethodsPct	Proportion of magic methods in the class	Real [0, 1]
AsyncMethodsPct	Proportion of async methods in the class	Real [0, 1]
ClassMethodsPct	Proportion of class methods in the class	Real [0, 1]
StaticMethodsPct	Proportion of static methods in the class	Real [0, 1]
AbstractMethodsPct	Proportion of abstract methods in the class	Real [0, 1]
PropertyMethodsPct	Proportion of property methods in the class	Real [0, 1]
SourceCode$^{\Psi}$	Python code of the expression (use `ast.unparse()`)	String
ModuleID$^{\Psi}$	Primary key of the module where this class was defined	Unique ID (Integer)

FunctionDefs:

Name	Description	Domain
FunctionDefID[Ψ]	Primary key of the function definition	Unique ID (Integer)
NameConvention	Its naming convention	NamingConvention[2]
NumberOfCharacters	The number of characters of the function ID	Integer
IsPrivate	Whether the variable name starts with the "_" character	Boolean
IsMagic	Whether the variable name starts and ends with two "_" characters. In that case, IsPrivate must be false.	Boolean
BodyCount	Number of statements in the function body	Integer
ExpressionsPct	Proportion of expressions in its body (most of them would be statements)	Real [0, 1]
IsAsync	Whether the method is asynchronous	Boolean
NumberOfDecorators	Number of decorators	Integer
HasReturnTypeAnnotation	Whether the function has a return type annotation	Boolean
HasDocString	Whether the function has a function comment. It is true when its first entry in `body` is a `Constant` with a string `value`	Boolean
Height	Distance (number of edges) from the current node to the root node in the enclosing module (not program)	Integer
TypeAnnotationsPct	Proportion of type annotations for the function parameters and return values	Real [0, 1]
SourceCode[Ψ]	Python code of the expression (use `ast.unparse()`)	String
ModuleID[Ψ]	Primary key of the module where these function was written	Unique ID (Integer)
ParametersID[Ψ]	Primary key of a Parameters entry where its parameters are defined	Unique ID (Integer)

MethodDefs (inherits all the columns from FunctionDefs; only the new columns are specified):

Name	Description	Domain
MethodDefID[Ψ]	Primary key of the method definition	Unique ID (Integer)
IsClassMethod	Whether the method has a @classmethod decorator	Boolean
IsStaticMethod	Whether the method has a @staticmethod decorator	Boolean
IsConstructorMethod	Whether the method is a constructor (__init__)	Boolean
IsAbstractMethod	Whether the method has an @abstract decorator	Boolean
IsProperty	Whether the method has a @property decorator	Boolean
IsWrapper	Whether the method has a @wraps decorator	Boolean

Name	Description	Domain
IsCached	Whether the method has a @cache decorator	Boolean
ClassDefID$^{\Psi}$	Primary key of its Class Definition	Unique ID (Integer)

Statements (if an expression is created as a statement, it is will not be included in this table but in Expressions):

Name	Description	Domain
StatementID$^{\Psi}$	Primary key of the statement	Unique ID (Integer)
Category	Syntactic category of the current node	StatementCategory[4]
Parent	Syntactic category of the parent node	Module | ClassDef | FunctionDef | MethodDef | StatementCategory[4]
StatementRole	Role played by the current node in the structure of its parent node	StatementRole[5]
Height	Distance (number of edges) from the current node to the root node in the enclosing module (not program)	Integer
Depth	Maximum distance (number of edges) of the longest path from the current node to a leaf node.	Integer
SourceCode$^{\Psi}$	Python code of the expression (use `ast.unparse()`)	String
HasOrElse	Whether the statement has else body or not. True or false for Try, TryStar, If, For, AsyncFor and While (N/A otherwise)	N/A, True, False
BodySize	The number of statements in the body. Only valid for While, If (no else), While, For, AsyncFor, Try, TryStar, With, AsyncWith (N/A otherwise).	N/A, Integer
First, second and third child's IDs (expressions)	Foreign keys of Expressions	Unique ID (integer)
ParentID$^{\Psi}$	Primary key of the parent	Unique ID (Integer)

[4]StatementCategory (the AST node of the Python `ast` module is stated between parenthesis):

- Return (one expression child, at most).
- Delete (one expression child).
- AssignmentStmt (still binary for multiple assignments because operands are modelled as tuples)
- TypeAlias (two children: first, its `name`; second the first type parameter (`type_params`), if any; third, the `value`).
- AugmentedAssignment (the operator is not stored)
- AnnotatedAssignment (first child is the variable name (`target`), second one the type annotation (`annotation`), third one the assigned `value`, if any).

- For (first child is the for element (`target`), second one the enumerable (`iter`); no third child (bodies are not included)).
- If (first child is the condition (`test`); no more children).
- While (first child is the condition (`test`); no more children).
- With (first child is the first element (`withitem` plus `context_expr`), second one the first "as" (`withitem` plus `optional_vars`), third one the second with element, if any).
- AsyncWith (first child is the first element (`withitem` plus `context_expr`), second one the first "as" (`withitem` plus `optional_vars`), third one the second with element, if any).
- Match (its only child is the condition expression (`subject`)).
- Raise (first child is the expression risen (`exc`); second one the "from" (`cause`)).
- Try (includes TryStar; no children).
- Assert (first child is `test`; second is `msg`, if any).
- Global (no children).
- NonLocal (no children).
- Pass (no children).
- Break (no children).
- Continue (no children).
- ExceptHandler

StatementRole[5]: Module (global statement), FunctionDef, AsyncFunctionDef, MethodDef, AsyncMethodDef, ClassDef (inside the class, not inside a method), For, ForElse, AsyncFor, AsyncForElse, With, AsyncWith, Try (includes TryStar), TryElse (includes TryStar), TryFinally (includes TryStar), TryHandler (does not include TryStar), TryHandlerStar (handler for TryStar), Case

Cases (one entry per Match statement):

Name	**Description**	**Domain**
NumberOfCases	Number of cases in a match statement	Integer
Guards	Number of guards / number of cases (in a match statement)	Real [0, 1]
AverageBodyCount	Average number of statements in the cases' bodies	Real
AverageMatchValue	Average number of MatchValue patterns in the cases of the match statement	Real [0 ,1]
AverageMatchSingleton	Average number of MatchSingleton patterns in the cases of the match statement	Real [0 ,1]
AverageMatchSequence	Average number of MatchSequence patterns in the cases of the match statement	Real [0 ,1]
AverageMatchMapping	Average number of MatchMapping patterns in the cases of the match statement	Real [0 ,1]
AverageMatchClass	Average number of MatchClass patterns in the cases of the match statement	Real [0 ,1]
AverageMatchStar	Average number of MatchStar patterns in the cases of the match statement	Real [0 ,1]

Name	Description	Domain
AverageMatchAs	Average number of MatchAs patterns in the cases of the match statement	Real [0 ,1]
AverageMatchOr	Average number of MatchOr patterns in the cases of the match statement	Real [0 ,1]
StatementID$^{\Psi}$	Primary key of the match statement	Unique ID (Integer)

Handlers (one entry per Try or TryStar statement):

Name	Description	Domain
NumberOfHandlers	Number of handlers in a try statement	Integer
HasFinally	Whether the handlers include a finally body	Boolean
HasCatchAll	Whether the handlers include a catch-all (`type==None`) handler	Boolean
AverageBodyCount	Average number of statements in the bodies of the handlers	Real
HasStar	Whether it includes a handler with star (TryStar)	Boolean
StatementID$^{\Psi}$	Primary key of the Try or TryStar statement	Unique ID (Integer)

Expressions:

Name	Description	Domain
ExpressionID$^{\Psi}$	Primary key of the expression	Unique ID (Integer)
Category	Syntactic category of the current node	ExpressionCategory[1]
First, second, third and fourth child	Syntactic category of the i-th child (one column per child, four max)	Parameter | ExpressionCategory[1]
Parent	Syntactic category of the parent node	Module | ClassDef | FunctionDef | MethodDef | StatementCategory[4]| ExpressionCategory[1]
ExpressionRole	Role played by the current node in the structure of its parent node	ExpressionRole[3]
Height	Distance (number of edges) from the current node to the root node in the enclosing module (not program)	Integer
Depth	Maximum distance (number of edges) of the longest path from the current node to a leaf node.	Integer
SourceCode$^{\Psi}$	Python code of the expression (use `ast.unparse()`)	String
ParentID$^{\Psi}$	Primary key of the parent	Unique ID (Integer)
First, second, third and fourth child ID	ID from the i-th child	Unique ID (Integer)

[1]ExpressionCategory (the AST node of the Python `ast` module is stated between parenthesis):

- Logical (`BoolOp`)
- AssignmentExp (`NamedExpr`)
- Arithmetic (`BinOp` with `Add`, `Sub`, `Mult`, `Div`, `FloorDiv` and `Mod` operators)

- Pow (`BinOp` with `Pow` operator)
- Shift (`BinOp` with `LShift` and `RShift` operators)
- BW[1]Logical (`BinOp` with `BitOr`, `BitXor` and `BitAnd` operators)
- MatMult (`BinOp` with `MatMult` operator)
- UnaryAritmetic (BinOp with `UAdd` y `USub` operators)
- UnaryNot (`BinOp` with `Not` operator)
- UnaryBWNot (`BinOp` with `Invert` operator)
- Lambda (`Lambda`). A lambda expression is a unary expression, where its only child is the `body` field from the ast. However, it must also contain a link to the Parameters table.
- Ternary (`IfElse`)
- SetLiteral (`Set`). This node has as many children as elements, but it also has a link to the Vectors table.
- ListLiteral (`List`) This node has as many children as elements, but it also has a link to the Vectors table.
- TupleLiteral (`Tuple`) This node has as many children as elements, but it also has a link to the Vectors table.
- DictionaryLiteral (`Dict`). This node has as many children as elements, but it also has a link to the Vectors table. Different from the Python ast representation, we consider each child as a 2-typle collecting (key, value).
- ListComprehension (`ListComp`). This node has as many children as elements, but it also has a link to the Comprehensions table. The first child represents the ExpressionCategory[1] of the element (`elt`). The second, the ExpressionCategory[1] of the `target`. Third, the ExpressionCategory[1] of the iteration (`it`). Fourth, the ExpressionCategory[1] of the first if (`ifs`).
- SetComprehension (`SetComp`). The same as ListComprehension.
- DictComprehension (`DictComp`) . The same as ListComprehension.
- GeneratorComprehension (`GeneratorExp`) . The same as ListComprehension.
- Await (`Await`)
- Yield (`Yield`)
- YieldFrom (`YieldFrom`)
- Compare
- Call (`Call`). This node has as many children as arguments, but it also has a link to the CallArgs table.
- FString (`JoinedStr`). This node has a link to the Fstrings table.
- IntLiteral (Constant with int value)
- FloatLiteral (`Constant` with float value)
- ComplexLiteral (`Constant` with complex value)
- NoneLiteral (`Constant` with `None` value)
- BoolLiteral (`Constant` with `True` or `False` value)
- StringLiteral (`Constant` with str value)
- EllipsisLiteral (`Constant` with `Ellipsis` value)
- Dot (`Attribute`)
- Variable (`Name`). This node has a link to the Variables table.

[1] BT stands for bitwise.

- Slice (`Slice`)
- Indexing (`Subscript`). Notice that, by using tuples and slices, this expression is always binary: `l[1:2,3]` has a tuple as the second parameter, where the first child is a Slice and the second one a IntLiteral.
- Star (`Starred`)

[2] ExpressionRole: Module, FuncDecorator, FuncBody, ReturnType (`returns`), ClassBase, ClassDecorator, MethodBody, ClassBody (expressions inside a class definition, not inside methods), Return, Delete, AssignLHS, AssignRHS, TypeAliasLHS, TypeAliasRHS, AugmentedAssignmentLHS, AugmentedAssignmentRHS, VarDefVarName (`target` field of `AnnAssign`), VarDefType (`annotation` field of `AnnAssign`), VarDefInitValue (`value` field of `AnnAssign`), ForElement (`target`), ForEnumerable (`iter`), ForBody, ForElseBody, AsyncForElement (`target`), AsyncForEnumerable (`iter`), AsyncForBody, AsyncForElseBody, WhileCondition, WhileBody, WhileElseBody, IfCondition, IfBody, IfElseBody, WithElement (`withitem` plus `context_expr`), WithAs (`withitem` plus `optional_vars`), WithBody, AsyncWithElement (`withitem` plus `context_expr`), AsyncWithAs (`withitem` plus `optional_vars`), AsyncWithBody, MatchCondition (`subject`), CaseCondition (`value` in match_ statements), CaseGuard (`guard`, either `left` or `comparators`), CaseBody, Raise (`exc`), RaiseFrom (`cause`), TryBody, ExceptType, ExceptBody, TryElse, FinallyBody, AssertCondition, AssertMessage, Logical (both binary and unary), AssignExpLHS, AssignExpRHS, Arithmetic (both binary and unary), Pow, Shift, BWLogical (both binary and unary), 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.

The following are weak entities derived from Expressions.

Comprehensions:

Name	**Description**	**Domain**
Category	Kind of comprehension	ListComp, SetComp, DictComp, GenComp
Number of ifs	Number of elements in the `ifs` field	Integer
Number of generators	Number of generators in the comprehension (`generators` field)	Integer
IsAsync	Whether the comprehension is async	Boolean
ExpressionID$^{\Psi}$	Primary key of the corresponding expression	Unique ID (Integer)

CallArgs:

Name	**Description**	**Domain**
NumberArgs	The number of all the arguments	Integer

Name	Description	Domain
NamedArgsPct	Proportion of arguments with the syntax: `name=arg_value`	Real [0, 1]
DoubleStarArgsPct	Proportion of arguments with the following syntax: `**args` (there could be more than one)	Real [0, 1]
ExpressionIDᵠ	Primary key of the corresponding expression	Unique ID (Integer)

Fstrings:

Name	Description	Domain
NumberOfElements	Number of elements in the f-string (values that are `Constant` plus values that are `FormattedValue`)	Integer
ConstantsPct	Proportion of string literal fragments in the f-string (number of `values` that are `Constant` in the ast)	Real [0, 1]
ExpressionsPct	Proportion of expressions in the f-string (number of `FormattedValues` in the `values` list of the ast node)	Real [0, 1]
ExpressionIDᵠ	Primary key of the corresponding expression	Unique ID (Integer)

Variables:

Name	Description	Domain
NameConvention	Its naming convention	NamingConvention[2]
NumberOfCharacters	The number of characters of the variable ID	Integer
IsPrivate	Whether the variable name starts with the "_" character	Boolean
IsMagic	Whether the variable name starts and ends with two "_" characters. In that case, IsPrivate must be false.	Boolean
ExpressionIDᵠ	Primary key of the corresponding expression	Unique ID (Integer)

[2] NamingConvention:

- Lower: all the letters are lowercase.
- Upper: all the letters are uppercase, and _ could be included.
- CamelLow: camel case, starting with a lowercase letter.
- CammelUp: camel case, starting with an uppercase letter.
- SnakeCase: snake case (everything lowercase but with _s).
- Discard: just the _ character.

Vectors:

Name	Description	Domain
Category	Kind of literal	ListLiteral, SetLiteral, DictLiteral, GeneratorLiteral
NumberOfElements	The number of elements in the literal	Integer

Name	Description	Domain
Homogeneous	Whether the elements in the literal are the same type of not. Notice that `[1, 1.2]` are different (IntLiteral and FloatLiteral), whereas `[my_int, my_float]` are not (syntactically, `my_int` and `my_float` are both Variables)	Boolean
ExpressionID[Ψ]	Primary key of the corresponding expression	Unique ID (Integer)

Parameters:

Name	Description	Domain
NumberOfParams	Number of all the parameters	Integer
PosOnlyParamPct	Proportion of positional only parameters (see the / char in here).	Real [0, 1]
VarParamPct	Proportion of var params	Real [0, 1]
HasVarParam	Whether it has a `vararg` value	Boolean
TypeAnnotationPct	Proportion of parameters with type annotations	Real [0, 1]
KwOnlyParamPct	Proportion of parameters with keyword only option (`kwonlyargs`)	Real [0, 1]
DefaultValuePct	Proportion of parameters with default values (`defaults` plus `kw_defaults`)	Real [0, 1]
HasKWParam	Whether it has a `kwarg`	Boolean
NameConvention	The naming convention used for most parameters	NamingConvention[2]
ParametersRole	Parameters role on the parent node	FunctionParameters, LambdaParameters
ParentID	ID of the parent node (FunctionDef or Lambda)	UniqueID (integer)
ParametersID	Identification ID	UniqueID (integer)

Common attributes for all the tables

Name	Description	Domain
UserID	Program author identificator	UniqueID (integer)
ExperticeLevel	Program author expertise	Beginner, Expert