Data Analysis Using Python

Samatrix Consulting Pvt Ltd

Samatrix.io

Course Introduction

About the course

- Hands on Data Analysis using Python
- Focus on learning by doing
  - Python
  - Numpy
  - Pandas
  - Matplotlib
  - Seaborn
- Hands on projects on Data Analysis

Getting Started

Topics

- Computational thinking
- Programs
- Programming Language
- Errors

Computational Thinking

- A computer performs only two actions: it calculates and remembers the results of the calculations.
- Important point is that it does those two things extremely well.
- A desktop computer or a laptop performs a billion or so calculations a second. We cannot imagine how truly fast that is.
- Let's assume that we are holding a ball a meter above the floor. Then we release it.
- The time it takes to reach the floor, in the same time, your computer could have executed over a billion commands.
- As for memory, a typical computer might have hundreds of gigabytes of storage.

Computational Thinking

- For most of human history, computation was limited by the speed of calculation of the human brain and the ability to record computational results with the human hand.
- This meant that only the smallest problems could be attacked computationally.
- To computationally solve any problem that we face in our day-to-day life, it is important to understand the concepts of **computational thinking**.

Declarative vs Imperative Knowledge

- **Declarative knowledge:** The statements of fact is known as declarative knowledge. For example, “the square root of x is a number y such that y*y = x.” This is a statement of fact. But from this statement, we do not know, how we can find the square root.
- **Imperative knowledge:** On the other hand, imperative knowledge is “how to” knowledge, or recipes for deducing information.

Sequence of Steps

- Consider, for example, finding the square root of 25.
  - Let's assign the variable x some arbitrary value, e.g., 3.
  - We check the difference between 3x3 = 9 and 25, which is high
  - We set x to (3 + 25/3)/2 = 5.67
  - We check the difference between 5.67x5.67 = 32.15 and 25, which is again high
  - We set x to (5.67 + 25/5.67)/2 = 5.04
  - We check the difference between 5.04*5.04 = 25.4 and 25, which is low, so we stop and declare 5.04 to be an adequate approximation to the square root of 25.

Algorithm

- The description of the method includes a sequence of simple steps.
- It also includes a flow of control which marks when each step should be executed.
- We call such a description as **algorithm**.
- The algorithm given above is guess and check algorithm which checks whether the guess was a good one.
- We can also define an algorithm as a finite list of instructions that proceed through a set of well-defined states and finally produce an output.

Algorithm

- An algorithm is a bit like a recipe from a cookbook:
  1. Put custard mixture over heat.
  2. Stir.
  3. Dip spoon in custard.
  4. Remove spoon and run finger across back of spoon.
  5. If clear path is left, remove custard from heat and let cool.
  6. Otherwise repeat.

Algorithm

- It includes
  - some tests for deciding when the process is complete
  - instructions about the order in which to execute instructions
  - jumping to some instruction based on a test.

Fixed Program Computers

- How can we capture the idea of a recipe in a mechanical process?
- We design a machine specifically to compute square roots.
- The earliest computing machines were fixed program computers.
- Means they were designed to do very specific things, and were mostly tools to solve a specific mathematical problem.
- For example, a calculator is a **fixed-program computer**.
- It can do basic arithmetic, but it cannot be used as a word processor or to run video games.

Stored Program Computer

- A **stored-program computer** stores (and manipulates) a sequence of instructions.
- It has a set of elements that will execute any instruction in that sequence.

Program

- The heart of the computer is a **program** (called an interpreter) that can execute any legal set of instructions.
- The program can be used to compute anything that can be describe as a set of instructions.

Programming Language

- In order to create sequence of instructions, we need a programming language.
- By using a programming language, we can give instructions to the computer.
- There are hundreds of **programming languages** in the world.
- There is no best language (though one could nominate some candidates for worst).
- Different languages are better or worse for different kinds of applications.
- MATLAB, for example, is an excellent language for manipulating vectors and matrices.
- C is a good language for writing the programs that control data networks.
- PHP is a good language for building Web sites.
- And Python is a good general-purpose language.

Primitive Construct

- Each programming language has a set of **primitive** constructs, a syntax, a static semantics, and a semantics.
- By analogy with a natural language, e.g., English, the primitive constructs are words, the syntax describes which strings of words constitute well-formed sentences, the static semantics defines which sentences are meaningful, and the semantics defines the meaning of those sentences.
- The primitive constructs in Python include literals (e.g., the number 3.2 and the string 'abc') and infix operators (e.g., + and /).

Syntax

- The **syntax** means how the words should be connected to form a sentence.
- For example, in English the string “Cat dog boy.” is not a syntactically valid sentence, because the syntax of English does not accept sentences of the form <noun> <noun> <noun>.
- In Python, the sequence of primitives 3.2 + 3.2 is syntactically well formed, but the sequence 3.2 3.2 is not.

Static Semantic

- The **static semantics** defines which syntactically valid strings have a meaning.
- In English, for example, the string “I are big,” is of the form <pronoun> <linking verb> <adjective>, which is a syntactically acceptable sequence.
- Nevertheless, it is not valid English, because the noun “I” is singular and the verb “are” is plural.
- This is an example of a static semantic error. In Python, the sequence 3.2/'abc' is syntactically well formed (<literal> <operator> <literal>), but produces a static semantic error since it is not meaningful to divide a number by a string of characters.

Semantic

- The **semantics** of a language associates a meaning with each syntactically correct string of symbols that has no static semantic errors.
- In natural languages, the semantics of a sentence can be ambiguous. For example, the sentence “I cannot praise this student too highly,” can be either flattering or damning.
- Programming languages are designed so that each legal program has exactly one meaning.

Program Errors

- Though syntax errors are the most common kind of error (especially for those learning a new programming language), they are the least dangerous kind of error.
- Every serious programming language does a complete job of detecting syntactic errors, and will not allow users to execute a program with even one syntactic error.
- Furthermore, in most cases the language system gives a sufficiently clear indication of the location of the error that it is obvious what needs to be done to fix it.

Program Errors

- The situation with respect to static semantic errors is a bit more complex.
- Some programming languages, e.g., Java, do a lot of static semantic checking before allowing a program to be executed.
- Others, e.g., C and Python (alas), do relatively less static semantic checking.
- Python does do a considerable amount of static semantic checking while running a program.
- However, it does not catch all static semantic errors.
- When these errors are not detected, the behavior of a program is often unpredictable

Program Errors

- One doesn’t usually speak of a program as having a semantic error.
- If a program has no syntactic errors and no static semantic errors, it has a meaning, i.e., it has semantics.
- Of course, that isn’t to say that it has the semantics that its creator intended it to have.
- When a program means something other than what its creator thinks it means, bad things can happen.

Program Errors

- What might happen if the program has an error, and behaves in an unintended way?
  - It might crash, i.e., stop running and produce some sort of obvious indication that it has done so. In a properly designed computing system, when a program crashes it does not do damage to the overall system. Of course, some very popular computer systems don't have this nice property. Almost everyone who uses a personal computer has run a program that has managed to make it necessary to restart the whole computer.
  - Or it might keep running, and running, and running, and never stop. If one has no idea of approximately how long the program is supposed to take to do its job, this situation can be hard to recognize.
  - Or it might run to completion and produce an answer that might, or might not, be correct.

Thanks

Samatrix Consulting Pvt Ltd

Samatrix.io