# Data Science – Khoa Học Dữ Liệu

Nguyễn Quản Bá Hồng*

Ngày 10 tháng 1 năm 2025

**Tóm tắt nội dung**

This text is a part of the series *Some Topics in Advanced STEM & Beyond*:
URL: https://nqbh.github.io/advanced_STEM/.
Latest version:

- *Data Science – Khoa Học Dữ Liệu*.
  PDF: URL: https://github.com/NQBH/advanced_STEM_beyond/blob/main/data_science/NQBH_data_science.pdf.
  TEX: URL: https://github.com/NQBH/advanced_STEM_beyond/blob/main/data_science/NQBH_data_science.tex.

## Mục lục



## 1 Basic Data Science – Khoa Học Dữ Liệu Cơ Bản

**Resources – Tài nguyên.**

1. [McK22]. Wes McKinney. *Python for Data Analysis: Data Wrangling with pandas, NumPy & Jupyter*. [356 Amazon ratings][25357 Goodreads ratings]

   Amazon review. Get definitive handbook for manipulating, processing, cleaning, & crunching datasets in Python. Updated for Python 3.10 & pandas 1.4, 3e of this hand-on guide is packed with practical case studies that show you how to solve a broad set of data analysis problems effectively. Learn latest versions of pandas, NumPy, & Jupyter in process.

   Written by Wes McKinney, creator of Python pandas project, this book is a practical, modern introduction to data science tools in Python. Ideal for analysts new to Python & for Python programmers new to data science & scientific computing. Data files & related material are available on GitHub.

   - use Jupyter notebook & IPython shell for exploratory computing
   - Learn basic & advanced features in NumPy
   - Get started with data analysis tools in pandas library
   - Use flexible tools to load, clean, transform, merge, & reshape data
   - Create informative visualizations with matplotlib
   - Apply pandas groupby facility to slice, dice, & summarize datasets
   - Analyze & manipulative regular & irregular time series data
   - Learn how to solve real-world data analysis problems with thorough, detailed examples

   About the Author. Wes McKinney is a Nashville-based software developer & entrepreneur. After finishing his undergraduate degree in mathematics at MIT in 2007, he went on to do quantitative finance work at AQR Capital Management in Greenwich, CT. Frustrated by cumbersome data analysis tools, he learned Python & started building what would later become pandas project. He's now an active member of Python data community & is an advocate for Python use in data analysis, finance, & statistical computing applications.

   Wes was later cofounder & CEO of DataPad, whose technology assets & team were acquired by Cloudera in 2014. He has since become involved in big data technology, joining Project Management Committees for Apache Arrow & Apache Parquet projects in Apache Software Foundation. In 2018, he founded Ursa Labs, a not-for-profit organization focused Apache Arrow

*A Scientist & Creative Artist Wannabe. E-mail: nguyenquanbahong@gmail.com. Bến Tre City, Việt Nam.

development, in partnership with RStudio & 2 Sigma Investments. In 2021, he cofounded technology startup Voltron Data, where he currently works as Chief Technology Officer.

"With this new edition, Wes has updated his book to ensure it remains go-to resource for all things related to data analysis with Python & pandas. I cannot recommend this book highly enough." – Paul Barry, Lecturer & author of O'Reiley; *Head 1st Python*

Wes McKinney, cofounder & chief technology officer of Voltron Data, is an active member of Python data community & an advocate for Python use in data analysis, finance, & statistical computing applications. A graduate of MIT, he's also a member of project management committees for Apache Software Foundation's Apache Arrow & Apache Parquet projects.

Preface. 1e of this book was published in 2012, during a time when open source data analysis libraries for Python, especially pandas, were very new & developing rapidly. When time came to write 2e in 2016–2017, needed to update book not only for Python 3.6 (1e used Python 2.7) but also for many changes in pandas that had occurred over previous 5 years. 2022, there are fewer Python language changes (now at Python 3.10, with 3.11 coming out at end of 2022), but pandas has continued to evolve.

In 3e, goal: bring content up to date with current versions of Python, NumPy, pandas, & other projects, while also remaining relatively conservative about discussing newer Python projects having appeared in last few years. Since this book has become an important resource for many university courses & working professionals, try to avoid topics that are at risk of falling out of date within 1–2 year. That way paper copies won't be too difficult to follow in 2023 or 2024 or beyond.

A new feature of 3e: open access online version hosted on website https://wesmckinney.com/book, to serve as a resource & convenience for owners of print & digital editions. Intend to keep content reasonably up to date there, so if you paper paper book & run into sth that doesn't work properly, should check there for latest content changes.

**Using Code Examples.** Can find data files & related material for each chap in this book's GitHub repository at https://github.com/wesm/pydata-book, which is mirrored to Gitee (for those who cannot access GitHub) at https://gitee.com/wesmckinn/pydata-book.

This book is here to help get job done. In general, if example code is offered with this book, may use it in your programs & documentation. Do not need to contact for permission unless you're reproducing a significant portion of code. E.g., writing a program that uses several chunks of code from this book does not require permission. Selling or distributing examples from O'Reilly books does not require permission. Answering a question by citing this book & quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

**Acknowledgments for 3e (2022).** $>1$ decade since started writing 1e of this book & $>15$ years since originally started journey as a Python programmer. A lot has changed since then! Python has evolved from a relatively niche (ngách) language for data analysis to most popular & most widely used language powering plurality (if not majority!) of DS, ML, & AI work.

Have not been an active contributor to pandas open source project since 2013, but its worldwide developer community has continued to thrive, serving as a model of community-centric open source software development. Many "next-generation" Python projects that deal with tabular data are modeling their user interfaces directly after pandas, so project has proved to have an enduring influence on future trajectory of Python DS ecosystem.

**Acknowledgments for 2e (2017).** 5 years almost to day since completed manuscript for this book's 1e in Jul 2012. A lot has changed. Python community has grown immensely, & ecosystem of open source software around it has flourished.

This new edition of book would not exist if for tireless efforts of pandas core developers, who have grown project & its user community into 1 of cornerstones of Python DS ecosystem.

With open source software projects more thinly resourced than ever relative to size of user bases, it is becoming increasingly important for businesses to provide support for development of key open source projects. It's the right thing to do.

- 1. Preliminaries.
  - 1.1. What Is This Book About? This book is concerned with nuts & bolts of manipulating, processing, cleaning, & crunching (nhai giòn tan) data in Python. Goal: offer a guide to parts of Python programming language & its data-oriented library ecosystem & tools that will equip you to become an effective data analyst. While "data analysis" is in title of book, focus is specifically on Python programming, libraries, & tools as opposed to data analysis methodology. This is Python programming you need *for* data analysis.
    Sometime after Wes originally published this book in 2012, people started using term *data science* as an umbrella description for everything from simple descriptive statistics to more advanced statistical analysis & ML. Python open source ecosystem for doing data analysis (or DS) has also expanded significantly since then. There are now many other books which focus specifically on these more advanced methodologies. Hope: this book serves as adequate preparation to enable you to move on to a more domain-specific resource.

    **Remark 1.** *Some might characterize much of content of book as "data manipulation" as opposed to "data analysis." Also use terms* wrangling *or* munging *to refer to data manipulation.*

    What Kinds of Data? Primary focus is on *structured data*, a deliberately vague term that encompasses many different common forms of data, e.g.:
    - Tabular or spreadsheet-like data in which each column may be a different type (string, numeric, date, or otherwise). This includes most kinds of data commonly stored in relational databases or tab- or comma-delimited text files.

* Multidimensional arrays (matrices).
* Multiple tables of data interrelated by key columns (what would be primary or foreign keys for a SQL user).
* Evenly or unevenly spaced time series.

This is by no means a complete list. Even though it may not always be obvious, a large percentage of datasets can be transformed into a structured form that is more suitable for analysis & modeling. If not, it may be possible to extract features from a dataset into a structured form. E.g., a collection of news articles could be processed into a word frequency table, which could then be used to perform sentiment analysis.

Most users of spreadsheet programs like Microsoft Excel, perhaps most widely used data analysis tool in the world, will not be strangers to these kinds of data.

- 1.2. Why Python for Data Analysis? For many people, Python programming language has strong appeal. Since its 1st appearance in 1991, Python has become 1 of most popular interpreted programming languages, along with Perl, Ruby, & others. Python & Ruby have become especially popular since 2005 or so for building websites using their numerous web frameworks, like Rails (Ruby) & Django (Python). Such languages are often called *scripting* languages, as they can be used to quickly write small programs, or *scripts* to automate other tasks. I don't like term "scripting languages," as it carries a connotation that they cannot be used for building serious software. Among interpreted languages, for various historical & cultural reasons, Python has developed a large & active scientific computing & data analysis community. In last 20 years, Python has gone from a bleeding-edge or "at your own risk" scientific computing language to 1 of most important languages for DS, ML, & general software development in academia & industry.

  For data analysis & interactive computing & data visualization, Python will inevitably draw comparisons with other open source & commercial programming languages & tools in wide use, e.g. R, MATLAB, SAS, Stata, & others. In recent years, Python's improved open source libraries (e.g. pandas & scikit-learn) have made it a popular choice for data analysis tasks. Combined with Python's overall strength for general-purpose software engineering, it is an excellent option as a primary language for building data applications.

  * Python as Glue. Part of Python's success in scientific computing: ease of integrating C, C++, & FORTRAN code - -1 phần thành công của Python trong điện toán khoa học: dễ dàng tích hợp mã C, C++, & FORTRAN. Most modern computing environments share a similar set of legacy FORTRAN & C libraries for doing linear algebra, optimization, integration, fast Fourier transforms, & other such algorithms. Same story has held true for many companies & national labs that have used Python to glue together decades' worth of legacy software.

    Many programs consist of small portions of code where most of time is spent, with large amounts of "glue code" that doesn't run often. In many cases, execution time of glue code is significant; effort is most fruitfully invested in optimizing computational bottlenecks, sometimes by moving code to a lower-level language like C.
  * Solving "2-Language" Problem. In many organizations, common to research, prototype, & test new ideas using a more specialized computing language like SAS or R & then later port those ideas to be part of a larger production system written in, say, Java, C#, or C++. What people are increasingly finding: Python is a suitable language not only for doing research & prototyping but also for building production systems. *Why maintain 2 development environments when one will suffice?* Believe more & more companies will go down this path, as there are often significant organizational benefits to having both researchers & software engineers using same set of programming tools.

    Over last decade some new approaches to solving "2-language" problem have appeared, e.g. Julia programming language. Getting most out of Python in many cases *will* require programming in a low-level language like C or C++ & creating Python bindings to that code. I.e., "just-in-time" (JIT) compiler technology provided by libraries like Numba have provided a way to achieve excellent performance in many computational algorithms without having to leave Python programming environment.
  * **Why Not Python?** While Python is an excellent environment for building many kinds of analytical applications & general-purpose systems, there are a number of uses for which Python may be less suitable.

    As Python is an interpreted programming language, in general most Python code will run substantially slower than code written in a compiled language like Java or C++. As *programmer time* is often more valuable than *CPU time*, many are happy to make this trade-off. However, in an application with very low latency or demanding resource utilization requirements (e.g., a high-frequency trading systems), time spent programming in a lower-level (but also lower-productivity) language like C++ to achieve maximum possible performance might be time well spent.

    – Vì Python là ngôn ngữ lập trình được thông dịch, nhìn chung hầu hết mã Python sẽ chạy chậm hơn đáng kể so với mã được viết bằng ngôn ngữ biên dịch như Java hoặc C++. Vì *thời gian lập trình* thường có giá trị hơn *thời gian CPU*, nhiều người vui vẻ chấp nhận sự đánh đổi này. Tuy nhiên, trong một ứng dụng có độ trễ rất thấp hoặc yêu cầu sử dụng tài nguyên khắt khe (ví dụ: hệ thống giao dịch tần suất cao), thời gian dành cho việc lập trình bằng ngôn ngữ cấp thấp hơn (nhưng cũng có năng suất thấp hơn) như C++ để đạt được hiệu suất tối đa có thể là thời gian được sử dụng hợp lý.

    Python can be a challenging language for building highly concurrent, multithreaded applications, particularly applications with many CPU-bound threads. Reason for this: it has what is known as *global interpreter lock* (GIL), a mechanism that prevents interpreter from executing $> 1$ Python instruction at a time. Technical reasons for why GIL exists are beyond scope of this book. While it is true that in many big data processing applications, a cluster of computers may be required to process a dataset in a reasonable amount of time, there are still situations where a single-process, multithreaded system is desirable.

    This is not to say: Python cannot execute truly multithreaded, parallel code. Python C extensions that use native mutithreading (in C or C++) can run code in parallel without being impacted by GIL, as long as they do not need to

regularly interact with Python objects.

- 1.3. Essential Python Libraries. For those who are less familiar with Python data ecosystem & libraries used throughout book, a brief overview of some of them:
  - NumPy. NumPy, short for Numerical Python, has long been a cornerstone of numerical computing in Python. It provides data structures, algorithms, & library glue needed for most scientific applications involving numerical data in Python. NumPy contains, among other things:
    - A fast & efficient multidimensional array object `ndarray`
    - Functions for performing element-wise computations with arrays or mathematical operations between arrays
    - Tools for reading & writing array-based datasets to disk
    - Linear algebra operations, Fourier transform, & random number generation
    - A mature C API to enable Python extensions & native C or C++ code to access NumPy's data structures & computational facilities

    Beyond fast array-processing capabilities that NumPy adds to Python, 1 of its primary uses in data analysis is as a container for data to be passed between algorithms & libraries. For numerical data, NumPy arrays are more efficient for storing & manipulating data than the other built-in Python data structures. Also, libraries written in a lower-level language, e.g. C or FORTRAN, can operate on data stored in a NumPy array without copying data into some other memory representation. Thus, many numerical computing tools for Python either assume NumPy arrays as a primary data structure or else target interoperability with NumPy.
  - `pandas`. pandas provides high-level data structures & functions designed to make working with structured or tabular data intuitive & flexible. Since its emergence in 2010, it has helped enable Python to be a powerful & productive data analysis environment. Primary objects in pandas that will be used in this book are `DataFrame`, a tabular, column-oriented data structure with both row & column labels, & `Series`, a 1D labeled array object.
    `pandas` blends array-computing ideas of NumPy with kinds of data manipulation capabilities found in spreadsheets & relationship databases (e.g. SQL). It provides convenient indexing functionality to enable you to reshape, slice & dice, perform aggregations (thực hiện tổng hợp), & select subsets of data. Since data manipulation, preparation, & cleaning are such important skills in data analysis, pandas is 1 of primary focuses of this book.
    As a bit of background, McKinney started building pandas in early 2008 during his tenure at AQR Capital Management, a quantitative investment management firm. At time, McKinney had a distinct set of requirements that were not addressed by any single tool at his disposal:
    - Data structures with labeled axes supporting automatic or explicit data alignment – this prevents common errors resulting from misaligned data & working with differently indexed data coming from different sources
    - Integrated time series functionality
    - Same data structures handle both time series data & non-time series data
    - Arithmetic operations & reductions that preserve metadata
    - Flexible handling of missing data
    - Merge & other relational operations found in popular databases (e.g., SQL-based)

    Wanted to be able to do all of these things in 1 place, preferably in a language well suited to general-purpose software development. Python was a good candidate language for this, but at that time an integrated set of data structures & tools providing this functionality did not exist. As a result of having been built initially to solve finance & business analytics problems, pandas features especially deep time series functionality & tools well suited for working with time-indexed data generated by business processes.
    McKinney spent a large part of 2011 & 2012 expanding pandas's capabilities with some of former AQR colleagues, Adam Klein, Chang She. In 2013, stopped being as involved in day-to-day project development, & pandas has since become a fully community-owned & community-maintained project with well $>$ 2000 unique contributors around world.
    For users of R language for statistical computing, `DataFrame` name will be familiar, as object was named after similar R `data.frame` object. Unlike Python, data frames are built into R programming language & its standard library. As a result, many features found in pandas are typically either part of R core implementation or provided by add-on packages.
    pandas name itself is derived from *panel data*, an econometrics term for multidimensional structured datasets, & a play on phrase *Python data analysis*.
  - matplotlib. matplotlib is most popular Python library for producing plots & other 2D data visualizations. It was originally created by John D. Hunter & is now maintained by a large team of developers. It is designed for creating plots suitable for publication. While there are other visualization libraries available to Python programmers, matplotlib is still widely used & integrates reasonably well with rest of ecosystem. Think it is a safe choice as a default visualization tool.
  - `IPython & Jupyter`. IPython project began in 2001 as Fernando Pérez's side project to make a better interactive Python interpreter. Over subsequent 20 years it has become 1 of most important tools in modern Python data stack. While it does not provide any computational or data analytical tools by itself, IPython is designed for both interactive computing & software development work. It encourages an *execute-explore* workflow instead of typical *edit-compile-run* workflow of many other programming languages. It also provides integrated access to OS's shell & filesystem; this

reduces need to switch between a terminal window & a Python session in many cases. Since much of data analysis coding involves exploration, trial & error, & iteration, IPython can help you get job done faster.

In 2014, Fernando & IPython team announced Jupyter project, a broader initiative to design language-agnostic interactive computing tools. IPython web notebook became Jupyter notebook, with support now for > 40 programming languages. IPython system can now be used as a *kernel* (a programming language mode) for using Python with Jupyter. IPython itself has become a component of much broader Jupyter open source project, which provides a productive environment for interactive & exploratory computing. Its oldest & simplest "mode" is as an enhanced Python shell designed to accelerate writing, testing, & debugging of Python code. You can also use IPython system through Jupyter notebook.

Jupyter notebook system also allows you to author content in Markdown & HTML, providing you a means to create rich documents with code & text.

McKinney personally uses IPython & Jupyter regularly in Python work, whether running, debugging, or testing code. In accompanying book materials on GitHub, you will find Jupyter notebooks containing all code examples from each chap. If cannot access GitHub where you are, can try mirror on Gitee.

* `SciPy.` SciPy is a collection of packages addressing a number of foundational problems in scientific computing. Some of tools it contains in its various modules:
  - `scipy.integrate`: Numerical integration routines & differential equation solvers
  - `scipy.linalg`: Linear algebra routines & matrix decompositions extending beyond those provided in `numpy.linalg`
  - `scipy.optimize`: Function optimizers (minimizers) & root finding algorithms
  - `scipy.signal`: Signal processing tools
  - `scipy.sparse`: Sparse matrices & sparse linear system solvers
  - `scipy.special`: Wrapper around SPECFUN, a FORTRAN library implementing many common mathematical functions, e.g. `gamma` function
  - `scipy.stats`: Standard continuous & discrete probability distributions (density functions, samplers, continuous distribution functions), various statistical tests, & more descriptive statistics

  Together, NumPy & SciPy form a reasonably complete & mature computational foundation for many traditional scientific computing applications.
* `scikit-learn`: Since project's inception in 2007, scikit-learn has become premier general-purpose ML toolkit for Python programmers. As of this writing, > 2000 different individuals have contributed code to project. It includes submodels for such models as:
  - Classification: SVM, nearest neighbors, random forest, logistic regression, etc.
  - Regression: Lasso, ridge regression, etc.
  - Clustering: $k$-means, spectral clustering, etc.
  - Dimensionality reduction: PCA, feature selection, matrix factorization, etc.
  - Model selection: Grid search, cross-validation, metrics
  - Preprocessing: Feature extraction, normalization

  Along with pandas, statsmodels, & IPython, scikit-learn has been critical for enabling Python to be a productive DS programming language. While I won't be able to include a comprehensive guide to scikit-learn in this book, I will give a brief introduction to some of its models & how to use them with other tools presented in book.
* `statsmodels` is a statistical analysis package that was seeded by work from Stanford University statistics professor Jonathan Taylor, who implemented a number of regression analysis models popular in R programming language. Skipper Seabold & Josef Perktold formally created new statsmodels project in 2010 & since then have grown project to a critical mass of engaged users & contributors. Nathaniel Smith developed Patsy project, which provides a formula or model specification framework for statsmodels inspired by R's formula system.

  Compared with scikit-learn, statsmodels contains algorithms for classical (primarily frequentist) statistics & econometrics. This includes such submodules as:
  - Regression models: linear regression, generalized linear models, robust linear models, linear mixed effect models, etc.
  - Analysis of variance (ANOVA)
  - Time series analysis: AR, ARMA, ARIMA, VAR, & other models
  - Nonparametric methods: Kernel density estimation, kernel regression
  - Visualization of statistical model results

  statsmodels is more focused on statistical inference, providing uncertainty estimates & $p$-values for parameters. scikit-learn, by contrast, is more prediction focused.

  As with scikit-learn, give a brief introduction to statsmodels & how to use it with NumPy & pandas.
* `Other Packages.` In 2022, there are many other Python libraries which might be discussed in a book about DS. This includes some newer projects like TensorFlow or PyTorch, which have become popular for ML or AI work. Now that there are other books out there that focus more specifically on those projects, recommend using this book to build a foundation in general-purpose Python data wrangling. Then, you should be well prepared to move on to a more advanced resource that may assume a certain level of expertise.

### ○ 1.4. Installation & Setup.

- 2. Python Language Basics, IPython, & Jupyter Notebooks.
- 3. Built-In Data Structures, Functions, & Files.
- 4. NumPy Basics: Arrays & Vectorized Computation.
- 5. Getting Started with pandas.
- 6. Data Loading, Storage, & File Formats.
- 7. Data Cleaning & Preparation.
- 8. Data Wrangling: Join, Combine, & Reshape.
- 9. Plotting & Visualization.
- 10. Data Aggregation & Group Operations.
- 11. Time Series.
- 12. Introduction to Modeling Libraries in Python.
- 13. Data Analysis Examples.
- A. Advanced NumPy.
- B. More on IPython System.

## 2 Miscellaneous

## Tài liệu

[McK22] Wes McKinney. *Python for Data Analysis: Data Wrangling with pandas, NumPy, & Jupyter*. 3rd edition. O'Reilly Media Publisher, 2022, p. 579.