

Advanced Data Science

Zuil Pirola

How Data Science Research Should Be Reported

Why reproducible reporting matters

- Scientific credibility depends on **transparency and reproducibility**
- Data, code, and methodology should be **openly accessible**
- Enables **validation, reuse**, and further research

Where modern research is shared

- Research paper hubs (e.g., Hugging Face Papers trending section)
- Public **GitHub repositories** accompanying publications
- Increasing expectation of **open science practices**

How Data Science Research Should Be Reported

Core reproducibility principles

- Share datasets whenever **legally/ethically** possible
- Provide **clean, executable code** and **documentation**
- Ensure computational **environments** can be recreated
- Maintain clear project structure and reporting standards

Development Environments in Data Science

Why use an **IDE**?

- Improves productivity with **debugging**, **linting***, and **code navigation**
- Facilitates project **organization** and **version control** integration
- Common choices: VS Code, PyCharm, Jupyter-based workflows

Best practices

- Create one **environment per project**
- Document dependencies (**requirements.txt** / `environment.yml`)

*<https://peps.python.org/pep-0008/>

Development Environments in Data Science

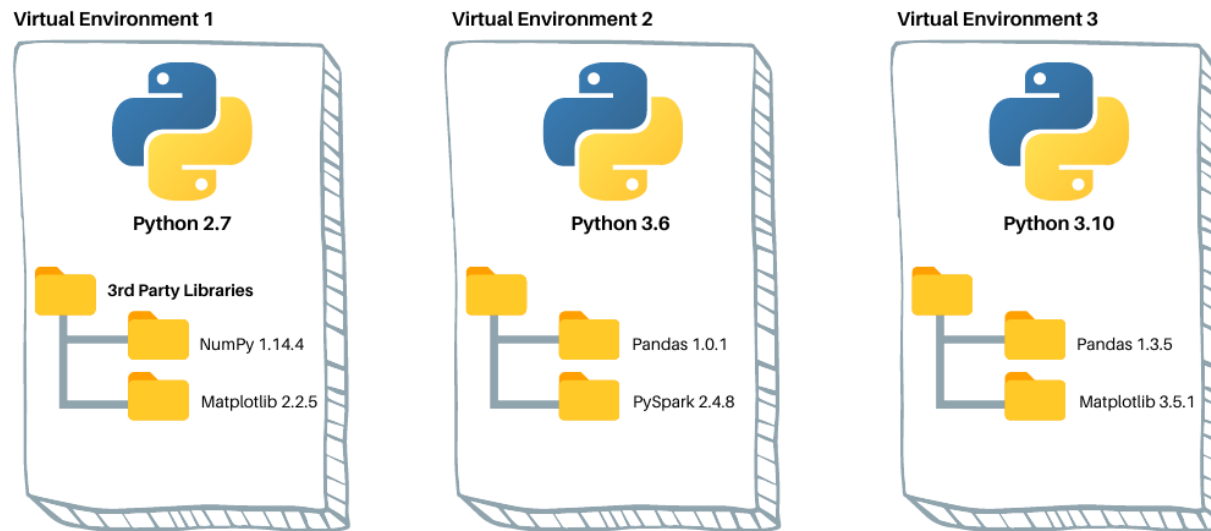
Environment management is essential

- **Avoids** dependency **conflicts** across projects
- Ensures **reproducibility** of experiments

Common tools

- **venv** → lightweight built-in Python virtual environments
- **conda** → environment + package manager widely used in data science

Configuration



dataquest.io

<https://www.dataquest.io/blog/a-complete-guide-to-python-virtual-environments/>

Configuration

Recommended Downloads (IDEs & Tools)

Local development environments for Python:



Installation Guides

- Visual Studio Code

VS CODE SETUP

- PyCharm + Jupyter Notebook


PC PYCHARM SETUP

Portuguese Version

PC PYCHARM SETUP

Example

Trending Papers

by  and the research community

[Daily](#) [Weekly](#) [Monthly](#) [★](#)


[Trending Papers](#)

Flavors of Moonshine: Tiny Specialized ASR Models for Edge Devices

Erin King · Adam Selzer · Manjusha Kulkarni · James Wang · Priti Warden · Moonshine II


Abstract

We present the Flavors of Moonshine, a suite of tiny monolingual speech recognition (ASR) models specialized for a range of underrepresented languages. These models maintain the same architecture as the larger Moonshine models, but are trained on a balanced mix of high-quality, pseudo-labeled, and synthetic data. We challenge this monolingual approach by training cross-lingual models, showing that for underrepresented languages, monolingual models can outperform cross-lingual models. This performance gap motivates the development of tiny monolingual ASR models for edge devices.

Submitted by  evanking

Flavors of Moonshine: Tiny Specialized ASR Models for Edge Devices

Monolingual ASR models trained on a balanced mix of high-quality, pseudo-labeled, and synthetic data outperform multilingual models for small model sizes, achieving superior error rates and enabling on-device ASR for underrepresented languages.

 5 authors · Published on Sep 2, 2025

[▲ Upvote 10](#)
[GitHub ★ 3.78k](#)
[arXiv Page](#)

Moonshine: Speech Recognition for Live Transcription and Voice Commands


Nat Jeffries · Evan King · Manjusha Kulkarni · Gay Nicholas · James Wang · Priti Warden · Moonshine

Abstract

This paper introduces Moonshine, a family of speech recognition models optimized for live transcription and voice commands. Moonshine is based on an encoder-decoder transformer architecture and employs Rotary Position Embedding (RoPE) for better handling of long-range dependencies. The model is trained on a large dataset of live transcription and voice commands, achieving state-of-the-art performance on these tasks. Moonshine is designed to be efficient and scalable, making it suitable for deployment on edge devices. The model is trained on a large dataset of live transcription and voice commands, achieving state-of-the-art performance on these tasks. Moonshine is designed to be efficient and scalable, making it suitable for deployment on edge devices.

Moonshine: Speech Recognition for Live Transcription and Voice Commands

Moonshine, an encoder-decoder transformer architecture for speech recognition, uses Rotary Position Embedding, reducing compute requirements without decreasing accuracy.

 6 authors · Published on Oct 21, 2024

[▲ Upvote -](#)
[GitHub ★ 3.72k](#)
[arXiv Page](#)

<https://huggingface.co/papers/trending>

Hands-On Practice: Building a Reproducible Repository (Ames Housing Dataset)

Objective

- Apply reproducibility principles by **creating a structured GitHub repository**
- Work with the **Ames** Housing dataset as a real Data Science **case study**

Dataset

- Ames Housing Dataset (Kaggle)
- Focus on **data loading, initial exploration**, and **reporting** findings

Hands-On Practice: Building a Reproducible Repository (Ames Housing Dataset)

```
# ames-project/  
# |  
# ├── data/  
# |   ├── raw/  
# |   └── processed/  
# |  
# ├── notebooks/  
# ├── src/  
# ├── reports/  
# |   └── figures/  
# |  
# ├── README.md  
# ├── requirements.txt  
# └── .gitignore
```

Expected Outcome

- A shared GitHub repository as a course **assignment**
- Clear organization, reproducible environment, and documented insights

The folder descriptions shown in the tree above are available in the Python notebook.

Hands-On Practice: Building a Reproducible Repository (Ames Housing Dataset)

Goal

- Carefully review the dataset features before any modeling step
- **Develop** a clear **understanding** of what **each variable** represents

Your Task

- Briefly describe each column in the dataset (1–2 lines per feature)
- Identify the **variable type** for each feature.

Hands-On Practice: Building a Reproducible Repository (Ames Housing Dataset)

Why This Matters

- **Prevents misinterpretation** of variables
- Helps **detect** inconsistencies or data **quality issues**
- **Guides** preprocessing, visualization, and modeling decisions

Deliverable

- A summarized **feature description** documented in your GitHub repository (README or reports)

Feature Types in Data Science

Numerical Variables

- Continuous: Real-valued measurements (e.g., price, height, temperature)
- Discrete: Countable integers (e.g., number of rooms, transactions)

Categorical Variables

- Nominal: Categories without intrinsic order (e.g., neighborhood, color)
- Ordinal: Ordered categories (e.g., quality rating, education level)

Binary Variables

- Two possible states (0/1, yes/no, true/false)
- Often treated as categorical but sometimes numeric for modeling

Feature Types in Data Science

Temporal Variables

- Dates, timestamps, durations
- Important for trends, seasonality, and time series analysis

Text / Unstructured Variables

- Free text, documents, comments
- Require NLP preprocessing (tokenization, embeddings, etc.)

Spatial / Geographical Variables

- Coordinates, regions, spatial relationships
- Used in geospatial and environmental analysis

Preliminary Data Quality Check

Start Evaluating Data Consistency

- Are there missing values in the dataset?
- Are they random, systematic, or meaningful?

Validate Data Types

- Check if stored types match expected semantics:
 - Dates → proper datetime format
 - Age, price, counts → numeric (int/float)
 - Categories → consistent categorical encoding

Preliminary Data Quality Check

Look for Incompatibilities

- Unexpected strings in numeric fields
- Incorrect formats (e.g., dates as text)
- Mixed data types within the same feature

Why This Step Matters

- Prevents modeling errors and biased conclusions
- Improves data cleaning and preprocessing decisions
- Strengthens reproducibility and reporting quality

Obrigado!