Introduction to Python Variables and Data Types

Md Kaisar Ahmed

ML Lectures

January 2026

Why Variables Matter

- Programming is about manipulating data
- Example: a computer game tracks
  - Player score
  - Number of lives
  - Current level
- These values change as the program runs

Question: How does a program store this data?

What Is a Variable?

Definition

A **variable** is a container for data.

- Every variable has:
  - a name
  - a value

Variable Example

- Variable name: `myAge`
- Variable value: 32

This is called **assigning a value** to a variable.

Creating a Variable in Python

```
myAge = 32
```

- `myAge` is the variable name
- `=` assigns a value

Displaying a Variable

```
print(myAge)
```

Output:

32

Updating a Variable

```
myAge = 33
print(myAge)
```

Output:

33

Variables in Calculations

```
print(myAge / 3)
```

Output:

```
11.0
```

Updating Using the Previous Value

```
myAge = myAge + 1
print(myAge)
```

- Right-hand side evaluated first
- Old value is overwritten

Practice Exercise

Restaurant Tip Calculation

1. Create `restaurantBill = 36.17`
2. Create `serviceCharge = 0.125`
3. Print the tip amount

Exercise Solution

```
restaurantBill = 36.17
serviceCharge = 0.125

print(restaurantBill * serviceCharge)
```

Output:

```
4.52
```

Case Sensitivity in Python

Important Rule

Python is case-sensitive.

- `restaurantBill` $\neq$ `restaurantbill`

Common Error Example

```
print(restaurantbill)
```

Error:

```
NameError: name 'restaurantbill' is not defined
```

Variables in Real Programs

```
data = ...
X = ...
y = ...
```

- These store many values
- Common in machine learning

Introduction to Data Types

- Integers
- Floating-point numbers
- Strings (text)
- Images, tables, audio, video

Checking Data Types

```
type(33)
```

Output:

```
int
```

Floating-Point Numbers

```
type(33.6)
```

Output:

```
float
```

Strings

```
type("Philip")
```

Output:

```
str
```

Strings are always inside quotes.

Variables and Data Types

```
type(myAge)
type(restaurantBill)
```

- But how do data types relate to variables? What is the connection between the two? Well, if the data has a certain type, then so does the variable containing the data, because after all, a variable is just a container.
- Variable type depends on stored data

Looking Ahead

- What are the data types of `data`, `X`, and `y`?
- Next: arrays, tables, datasets

Next topic: Data structures for machine learning

Lesson Goal

- Review variables briefly: single-value containers
- Introduce **collections**: variables holding many values
- Focus today on:
  - **Lists** (built into Python)
  - **Arrays** (NumPy, common in ML)

Why Collections?

- In ML and data science, we work with **lots of data**.
- Storing each value in a separate variable is inefficient.
- Instead, we store many values in one variable.

Collection

A **collection** is a single variable that contains multiple values.

- Examples: lists, arrays, DataFrames, Series

Python Lists

- A list is a **built-in** Python data structure.
- It stores an **ordered** sequence of items.
- Syntax uses square brackets: [ ]

Creating a List (Example)

```
primeNumbers = [3, 7, 61, 29, 199]
```

- primeNumbers stores 5 values
- Items are separated by commas

Type of a List

```
type(primeNumbers)
```

Expected output: list

Lists Can Store Different Data

- Lists can contain:
  - Numbers
  - Strings (text)
  - Mixed data types
- Even with different contents, the data type is still list.

List of Strings

```
coolPeople = ['Jay Z', 'Gandhi', 'Me']
```

Mixed-Type List

```
primesAndPeople = ['King Arthur', 17, 11, 'Jennifer
    Lopez']
```

- Lists are flexible
- They can hold different types in one collection

Retrieving Values: Indexing

- We access an item using its **index**.
- Index = position in the list.
- Python uses **zero-based indexing** (starts at 0).

Indexing Example

```
primeNumbers[2]
```

Expected output: 61

- Index 2 returns the **third** element

Zero-Based Indexing Table

Index	Value
0	3
1	7
2	61
3	29
4	199

- Valid indices go from 0 to (length 1)

Storing a Retrieved Value

```
bestPrimeEver = primeNumbers[4]
bestPrimeEver
```

Expected output: 199

Common Mistake: Off-by-One Errors

- Beginner mistake: counting from 1 instead of 0
- Leads to **IndexError**

Index Out of Range

```
primeNumbers[5]
```

Expected error: IndexError: list index out of range

- List has 5 items $\rightarrow$ valid indices: 0,1,2,3,4
- Index 5 refers to a 6th item (does not exist)

From Lists to Arrays

- Lists are flexible and general-purpose.
- In ML/scientific computing, we often use **arrays**.
- Arrays come from **NumPy**.

Lists vs Arrays (High-Level)

List (Python)	**Array (NumPy)**
Flexible	Optimized for speed and math
Can mix data types	Usually single data type
Easy to resize	Best for numerical data
Built-in	Requires NumPy

Arrays in Real ML Output

You may see something like this printed in a notebook:

array([-7236000])

This indicates the value is stored inside a **NumPy array**.

Checking the Array Type

```
type(regr.intercept_)
```

Expected output: numpy.ndarray

Indexing an Array

```
regr.intercept_[0]
```

- Arrays use the same indexing: [ ]
- Index 0 returns the first element

Key Takeaways

- Collections let one variable store many values
- Lists:
  - Built into Python
  - Can mix data types
  - Zero-based indexing
- Arrays:
  - From NumPy (numpy.ndarray)
  - Efficient for numerical computations
  - Same indexing idea

Looking Ahead

- Next: **Pandas Series** and **Pandas DataFrames**
- These are key structures for real-world datasets in ML.

Fun line: Why did the programmer quit his job? Because he didn't get arrays.

Lesson Overview

- Today: two important data structures for storing lots of data
  - **DataFrame** (2D: rows + columns)
  - **Series** (1D: single column)
- We will learn how to:
  - Load a CSV file using Pandas
  - Inspect a DataFrame and check its type
  - Extract columns (Series vs DataFrame)
  - Add, update, and delete columns
  - Create a smaller DataFrame (subset of columns)

Step 1: Upload the CSV File

- In your MLProjects folder, click **Upload**
- Select: $LSD_{m}ath_{s}core_{d}ata.csv$

- Return to the Jupyter notebook

Note

File name must match exactly (capitalization and spaces matter).

Step 2: Read CSV into a DataFrame

```
import pandas as pd
data = pd.read_csv('LSD_math_score_data.csv')
```

- pd.read$_c$*sv*(...)*loadsthedata*
- The variable data stores the loaded table

Step 3: Print the DataFrame

```
print(data)
```

- You should see a table with rows and columns
- Example structure: 7 rows and 3 columns

What is a DataFrame?

Definition

A **DataFrame** is a 2D table of data with **rows** and **columns**, like an Excel spreadsheet.

- Rows = observations (samples)
- Columns = variables (features)
- Very common in data science and machine learning

Check the Type of data

```
type(data)
```

- Expected: pandas.core.frame.DataFrame
- We usually say: “data is a DataFrame”

Extracting a Column from a DataFrame

- Lists/arrays use index positions: myList[2]
- DataFrames use column names: data['ColumnName']

Key Idea

In a DataFrame, square brackets often mean "select column(s)".

Selecting One Column (by name)

```
data['Avg_Math_Test_Score']
```

- Returns the values in that column
- Column name must be spelled exactly

Common Error: Wrong Column Name

```
data['Avg_Math_Test_Scor']   # typo (missing 'e')
```

- This produces a KeyError
- Python cannot find that column in the DataFrame

Common Error: Treating DataFrame Like a List

```
data[1]
```

- This also produces a KeyError
- DataFrames do not use numeric indexing in this way

Store a Column in a Variable

```
onlyMathScores = data['Avg_Math_Test_Score']
print(onlyMathScores)
```

- We extracted one column and saved it in onlyMathScores

Adding a New Column

- If the column does not exist, selecting it gives KeyError
- But if we assign to it, Pandas will create the column

Add a Column: Test$_S$*ubject*

```
data['Test_Subject'] = 'Jennifer Lopez'
print(data)
```

- Creates a new column Test$_S$*ubject*
- Sets every row to the same value

Creating and Updating Columns with Calculations

- Pandas can apply operations to an entire column at once
- This is called **vectorized computation**

Create a Column: High_Score

```
data['High_Score'] = 100
print(data)
```

- Adds $High_Score and sets all rows to 100$

Update: Add Two Columns

```
data['High_Score'] = data['High_Score'] + data['
    Avg_Math_Test_Score']
print(data)
```

- Overwrites $\text{High}_S core$
- Pattern is like: myAge = myAge + 1

Square the Values in a Column

```
data['High_Score'] = data['High_Score'] * data['
    High_Score']
print(data)
```

- Multiplies each value by itself (squares it)

Alternative: Power Operator

```
data['High_Score'] = data['High_Score'] ** 2
print(data)
```

- ** 2 means “raise to the power of 2”

Series: What Happens When We Extract One Column?

- DataFrame type: DataFrame
- One extracted column type: Series

Key Point

A **Series** is a 1D structure (only one column).

Check the Type of onlyMathScores

```
type(onlyMathScores)
```

- Expected: pandas.core.series.Series

DataFrames are Collections of Series

- Each column in a DataFrame is a Series
- A DataFrame is essentially multiple Series combined side-by-side

Mental Model

DataFrame = collection of **Series** (columns)

Creating a Smaller DataFrame (Subset)

- Sometimes we only need a few columns
- Example: keep only $\mathrm{LSD}_{p}pmand\,Avg_{M}ath_{T}est_{S}core$

Step 1: Create a List of Column Names

```
columnList = ['LSD_ppm', 'Avg_Math_Test_Score']
```

- This list contains two strings (column headers)

Step 2: Use the List to Create a New DataFrame

```
cleanData = data[columnList]
print(cleanData)
```

- cleanData is a DataFrame with only two columns

Nested Brackets: One-Line Version

```
cleanData = data[['LSD_ppm', 'Avg_Math_Test_Score']]
print(cleanData)
```

- Two brackets appear: data[[...]]
- It is just a list inside the DataFrame selection

Check the Type of cleanData

```
type(cleanData)
```

- Expected: pandas.core.frame.DataFrame

Series vs DataFrame When Selecting Columns

- If you pass a **string** (single bracket), you get a **Series**
- If you pass a **list** (double bracket), you get a **DataFrame**

Rule

data['col'] $\rightarrow$ Series data[['col']] $\rightarrow$ DataFrame

Create y as a Single-Column DataFrame

```
y = data[['Avg_Math_Test_Score']]
print(y)
type(y)
```

- y is a DataFrame (not a Series)

Exercise: Create X as a DataFrame

Task:

- Create X containing $\text{LSD}_p pm$
- Ensure X is a DataFrame
- Print X and type(X)

Hint: Use double brackets [['...']].

Solution: Create X

```
X = data[['LSD_ppm']]
print(X)
type(X)
```

- X is a single-column DataFrame

Deleting Columns

- Sometimes we add columns for testing or exploration
- To remove a column, we use del

Delete Test_Subject

```
del data['Test_Subject']
print(data)
```

- Removes the column from the DataFrame

Exercise: Delete High_Score

Task: Delete the High_Score column and print the DataFrame.

Hint: Use del data['High$_S$*core*'].

Solution: Delete High_Score

```
del data['High_Score']
print(data)
```

- DataFrame now has fewer columns

Summary

- Loaded CSV into a Pandas **DataFrame**
- Extracted columns:
  - data['col'] returns a **Series**
  - data[['col']] returns a **DataFrame**
- Added and updated columns using vectorized operations
- Created subsets of DataFrames using a list of column names
- Deleted columns using del

Next: More Pandas operations and preparing data for regression.

Lesson Overview

- Topic: **import statements**, **modules**, and **packages**
- We will focus on three import patterns you will see often:
  1. import module
  2. import module as alias
  3. from module import object
- Environment: **Google Colab** (instead of Jupyter)

Colab Note (Environment)

- Google Colab runs Python in the cloud.
- Many common data science libraries are already available.
- If a library is missing, you can install it using:

Typical installation in Colab

```
!pip install package_name
```

What is a Module?

Module

A **module** is code you can import and use in your notebook.

- Importing a module gives access to its functions, classes, and variables.
- You can think of modules as **libraries** (common term in other languages).

What is a Package?

Package

A **package** is a collection (bundle) of modules.

- Many popular tools in ML are packages:
  - NumPy
  - Pandas
  - Matplotlib
  - Scikit-learn
- A package contains many modules grouped together.

Import Pattern 1: import module

- Simplest form of importing:
- After importing, you use **dot notation** to access contents:

Example: Import a Built-in Module

```
import math
```

- math is part of Python's **standard library**
- Standard library modules are available without extra installation

Dot Notation: Access Values Inside a Module

```
import math

math.pi
math.e
```

- math.pi gives the value of π
- math.e gives the value of e

Python Standard Library

- Python includes many built-in modules known as the **standard library**.
- Example categories:
  - Math (math)
  - Dates and time (datetime)
  - File handling (os, pathlib)
  - Randomness (random)

Import Pattern 2: import module as alias

Alias

An **alias** is a short name that refers to a module.

- Used to avoid typing long module names repeatedly
- Also follows common community conventions

Example: Pandas Alias

```
import pandas as pd
```

- pd is a common convention for pandas
- Makes code shorter and cleaner

Example: Matplotlib Alias

```
import matplotlib.pyplot as plt
```

- matplotlib.pyplot is long
- plt is the standard alias used in most tutorials

Why Aliases Matter

- Without alias, you would repeatedly write: matplotlib.pyplot.plot(...)
- With alias, it becomes: plt.plot(...)

Key Point

Once you import using an alias, you must use the alias name.

Import Pattern 3: from module import object

- Imports a specific object (function/class/variable) from a module
- After importing, you can use the object name directly (no dot needed)

Example: Import LinearRegression

```
from sklearn.linear_model import LinearRegression
```

- LinearRegression is a class inside sklearn.linear$_m$*odel*
- After importing, you can use LinearRegression directly

Using the Imported Class

```
from sklearn.linear_model import LinearRegression

regr = LinearRegression()
```

- We often store long names in short variables:
- regr now references a LinearRegression model

Comparing the Three Import Styles

Style	How you use it
import math	math.pi
import pandas as pd	pd.read_csv(...)
from sklearn... import LinearRegression	LinearRegression()

Idea

All styles give access to external code; the choice is mostly about convenience and readability.

Modules, Packages, and Data Types

- When you check a data type, you often see where it came from:
  - pandas.core.frame.DataFrame $\rightarrow$ from pandas
  - numpy.ndarray $\rightarrow$ from NumPy
- This helps you understand what library defines an object.

Why Imports Matter in ML

- Imports are a huge productivity boost:
  - We reuse powerful, tested implementations
  - We avoid writing complex algorithms from scratch
- Example:
  - scikit-learn provides regression, classification, preprocessing, and more

Big Picture

We do cool ML quickly because we can build on the work of others.

Set Up for the Next Lesson (Colab)

In the next tutorial, we will analyze effects of drugs on math test scores.

Run these imports (Pandas assumed already imported earlier):

```
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
```

- plt for plotting
- LinearRegression to fit a best-fit line

What is Matplotlib?

- Matplotlib is the most popular plotting library for Python.
- It gives you fine control over almost every aspect of a figure.
- Designed to feel similar to MATLAB plotting.
- Works well with:
  - NumPy arrays
  - Pandas DataFrames
- Other libraries (e.g., Seaborn) are built on top of Matplotlib.

Matplotlib in Google Colab

- In most Colab notebooks, Matplotlib is already installed.
- If you ever need to install or upgrade:

Colab installation

```
!pip install matplotlib
```

- Then import for plotting:

Standard Import Convention

```
import matplotlib.pyplot as plt
```

- plt is the standard alias for matplotlib.pyplot
- You will see this convention in most tutorials and codebases

(Optional) Installation Outside Colab

- If you are working locally (not Colab), you can install Matplotlib with:

Conda (Anaconda)

```
conda install matplotlib
```

Pip (standard Python)

```
pip install matplotlib
```

- In this course, we will primarily use Colab.

Official Website

- Official site: matplotlib.org
- Useful sections:
  - Installation instructions
  - Documentation
  - Examples and tutorials

Tip

The website is a resource for you throughout this course.

The Most Helpful Page: Gallery

- The Matplotlib gallery shows many plot types and examples.
- Gallery link: matplotlib gallery
- You can browse:
  - Line plots, scatter plots, bar plots
  - Histograms, box plots, heatmaps
  - Pie charts, polar charts, and more

How to Use the Gallery (Workflow)

1. Go to the gallery and find the plot type you want
2. Click the example that looks closest to your goal
3. Read the page:
   - It shows the figure
   - It provides well-commented code
4. Copy and adapt the code to your own dataset

Example

Want a pie chart? Search "pie charts" in the gallery and open an example.

Why Matplotlib First?

- Many higher-level plotting libraries build on Matplotlib.
- Understanding Matplotlib makes it easier to:
  - Customize figures
  - Understand what Seaborn (and others) do under the hood
  - Debug plotting issues

Lecture Objectives

- Understand what **Seaborn** is
- Learn how Seaborn relates to Matplotlib
- See where to find official documentation
- Understand what kinds of plots Seaborn provides

What is Seaborn?

- Seaborn is a **statistical data visualization library** for Python.
- Built directly on top of **Matplotlib**.
- Designed to work especially well with **Pandas DataFrames**.
- Provides **beautiful default styles** with minimal code.

Key Idea

All Matplotlib knowledge carries over directly to Seaborn.

Seaborn in Google Colab

- Seaborn is usually **pre-installed** in Google Colab.
- If needed, installation can be done with:

Colab installation

```
!pip install seaborn
```

- Standard import pattern:

Standard Import Convention

```
import seaborn as sns
```

- sns is the standard alias for Seaborn.
- You will see this convention in most tutorials and examples.

Relationship to Matplotlib

- Seaborn is built on top of Matplotlib.
- Matplotlib handles:
  - Figure creation
  - Axes
  - Low-level rendering
- Seaborn focuses on:
  - Statistical plots
  - Better defaults
  - Cleaner syntax for common tasks

Official Documentation

- Seaborn is open-source and hosted on GitHub.
- Official documentation link:

https://seaborn.pydata.org

Recommendation

Bookmark the documentation — you will use it often.

Gallery Overview

- Seaborn provides a gallery of example plots.
- These examples demonstrate:
  - Heatmaps
  - Box plots
  - Violin plots
  - Joint plots
- Gallery is useful for inspiration, not exact reference.

Common Seaborn Plot Types

- Heatmaps
- Violin plots
- Box plots (including grouped box plots)
- Joint plots
- Hexbin-style statistical plots

Design Philosophy

Seaborn emphasizes clarity, aesthetics, and statistical meaning.

API Reference

- The **API Reference** is the most important part of the documentation.
- It lists:
  - Plotting functions
  - Required parameters
  - Optional styling arguments
- This is what we will rely on most while coding.

Why Learn Seaborn?

- Excellent integration with Pandas DataFrames
- Minimal code for high-quality statistical plots
- Industry-standard for exploratory data analysis (EDA)
- Complements Matplotlib rather than replacing it

Machine Learning with Python

- scikit-learn is the most popular machine learning library for Python.
- Provides many algorithms out of the box:
  - Regression
  - Classification
  - Clustering
  - Dimensionality reduction
- Consistent and easy-to-use interface.

Machine Learning Workflow (Review)

1. Collect data
2. Clean and format the data
3. Split data into training and testing sets
4. Train the model on training data
5. Test and evaluate the model
6. Tune parameters and iterate
7. Deploy the model

Estimator Concept in scikit-learn

Estimator

In scikit-learn, every machine learning algorithm is represented by an **estimator**.

- An estimator is a Python object
- It implements methods such as:
  - fit()
  - predict()
  - score()

Importing a Model

```
from sklearn.linear_model import LinearRegression
```

- LinearRegression is an estimator
- We import it from the appropriate model family

Instantiating the Model

```
model = LinearRegression()
```

- This creates a model object
- Model parameters have sensible default values
- Parameters can be customized if needed

Preparing the Data

- Features are stored in X
- Labels (targets) are stored in y
- Data must be split into:
  - Training set
  - Testing set

Train–Test Split

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2
)
```

- Automatically splits data
- Training data is used to fit the model
- Testing data is used to evaluate performance

Training the Model

- Training is done using the fit() method
- For supervised learning:
  - X_train = features
  - y_train = labels

Fitting the Model

```
model.fit(X_train, y_train)
```

- The model learns from training data

Making Predictions

- Once trained, the model can predict new values
- Uses the predict() method

Predicting on Test Data

```
y_pred = model.predict(X_test)
```

- y_pred contains predicted values
- These are compared against true labels

Evaluating the Model

- Evaluation depends on the problem type:
  - Regression
  - Classification
  - Clustering
- Many estimators implement a score() method

Supervised vs Unsupervised Learning

- **Supervised learning**
  - Uses labeled data
  - Methods: fit(), predict()
- **Unsupervised learning**
  - Uses unlabeled data
  - Methods: fit(), transform()

Common Estimator Methods

- fit(X, y) – train the model
- predict(X_new) – predict labels
- score(X, y) – evaluate performance
- transform(X) – transform data (unsupervised)
- fit_transform(X) – fit and transform in one step

Choosing an Algorithm

- scikit-learn provides an algorithm selection guide
- Helps decide between:
  - Regression vs classification
  - Supervised vs unsupervised

Key Question

Are you predicting a **quantity** or a **category**?

What's Next

- Start coding machine learning models in Python
- First algorithm: **Linear Regression**
- Apply the full scikit-learn workflow step by step