

A giant panda is the central focus of the image, sitting on a mat made of bamboo stalks. The panda is looking directly at the camera with a calm expression. Its black and white fur is clearly visible. The background is slightly blurred, showing more of the bamboo environment.

# Бібліотека pandas для аналізу даних

Денис Лонтковський  
dlont88@gmail.com

## Зміст:

- Вступ та інсталяція бібліотеки
- Базові структури даних: ряд та фрейм
- Базові операції читання та запис
- Об'єднання, групування та зміна форми фреймів
- Практика

# Introduction to Python Pandas

- ▶ **Core Library Importance:** pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the [Python](#) programming language. Pandas is essential for efficient data manipulation and analysis in Python.
- ▶ **Wide Adoption:** Its popularity stems from community support, extensive documentation, and suitability for diverse data science tasks.
- ▶ **Versatile Functionality:** Pandas offers tools for data cleaning, transformation, and exploration.

# Installation of Pandas

- ▶ **Installing Pandas:** Install Pandas via pip using

```
pip install pandas
```

- ▶ **Specifying Versions:** To install a specific version, use

```
pip install pandas==2.2.1
```

# Pandas Data Structures: Series

- ▶ **Pandas Series:** A Pandas Series is a one-dimensional labeled array, facilitating index-based data access and manipulation.
- ▶ **Key Features:** Series include automatic alignment of data by labels and support for various data types efficiently.
- ▶ **Use Cases:** Commonly used for time series data, Series manage chronological values seamlessly for analysis purposes.

# Pandas Data Structures: DataFrame

- ▶ **DataFrame Structure:** A DataFrame is a two-dimensional labeled array that accommodates heterogeneous data types.
- ▶ **Data Handling Abilities:** It allows for complex data operations like filtering, aggregation, and reshaping.
- ▶ **Typical Use Cases:** Commonly utilized for storing and manipulating datasets in data analysis.

# Creating a Pandas DataFrame

- ▶ **Creating DataFrames from Lists:** Utilize `pd.DataFrame(list)` for seamless conversion of Python lists into Pandas DataFrames, enhancing data structure.
- ▶ **Building DataFrames from Dictionaries:** Employ `pd.DataFrame(dict)` to convert dictionaries into DataFrames, permitting labeled columns based on dictionary keys.
- ▶ **Forming DataFrames with NumPy Arrays:** Integrate `np.array` with `pd.DataFrame` for generating DataFrames from NumPy arrays.

pandas\_constructor.py

# Reading Data with Pandas

- ▶ **Reading CSV Files:** Use `pd.read_csv('file_path.csv')` to load data from CSV files, ensuring proper delimiter configuration.
- ▶ **Reading hd5 data:** Utilize `pd.read_hdf('file_name.h5')` for import of data from HDF5 (Hierarchical Data Format) files.

# Writing Data with Pandas

- ▶ **Writing CSV Files:** Utilize `df.to_csv('output.csv')` for exporting DataFrames as CSV files, ensuring data integrity and structure.



# Data Inspection and Exploration

- ▶ **Data Inspection Methods:** Utilize `head()`, `tail()`, `describe()`, and `info()` for insight into dataset structure and statistics.
- ▶ **Understanding Head and Tail:** `head()` previews initial entries, while `tail()` displays final entries.
- ▶ **Descriptive Statistics:** `describe()` generates summary statistics like mean, std, min, and max, to describe data distribution.

# Data Selection, Filtering, Indexing and Slicing

- ▶ **Selecting Columns and Rows:** Utilize bracket notation or dot notation for precise column and row selection within DataFrames.
- ▶ **Boolean Indexing:** Apply boolean conditions to filter DataFrames, allowing targeted data analysis based on specific criteria.
- ▶ **Chaining Selection Techniques:** Combine methods seamlessly by chaining selection techniques, enhancing flexibility and efficiency in data retrieval.
- ▶ **Indexing Techniques Overview:** Secondary indexing capabilities are essential for efficient data access and organization in Pandas DataFrames.
- ▶ **Using loc Function:** loc allows label-based indexing, enabling users to retrieve rows and columns using index labels effectively.
- ▶ **Using iloc Function:** iloc provides position-based indexing, facilitating precise access to data based on integer-location references.

# Handling Time Series Data

- ▶ **Datetime Indexing:** Utilize Pandas' datetime capabilities for indexing data, enhancing time series analysis through organized temporal data.
- ▶ **Resampling Techniques:** Pandas supports resampling methods, allowing users to manipulate time series frequencies efficiently and accurately.

# Merging and Joining DataFrames

- ▶ **Merging DataFrames:** Merge combines two DataFrames based on common fields.
  - ▶ `merge()`: Combine two `Series` or `DataFrame` objects with SQL-style joining
- ▶ **Joining Techniques:** Join methods facilitate relational merges similar to SQL operations.
  - ▶ `DataFrame.join()`: Merge multiple `DataFrame` objects along the columns
- ▶ **Concatenation Methods:** Concat stacks DataFrames vertically or horizontally.
  - ▶ Merge multiple `Series` or `DataFrame` objects along a shared index or column
  - ▶ `concat()` makes a full copy of the data, and iteratively reusing `concat()` can create unnecessary copies

# Data Cleaning in Pandas

- ▶ **Handling Missing Values:** Employ `fillna()` to substitute missing data, ensuring continuity and quality in datasets through imputation.
- ▶ **Removing Duplicates:** Utilize `drop_duplicates()` for elimination of duplicate records.

`pandas_na_duplicates.py`

# Data Transformation

- ▶ **Applying Functions:** Utilize `apply()` to execute functions across `DataFrame` rows or columns, enhancing data transformation flexibility.
- ▶ **Mapping Values:** `Map()` allows element-wise transformations within a `Series`, crucial for straightforward value conversions and replacements.
- ▶ **Lambda Functions:** Leverage lambda functions in conjunction with `apply()` for concise, custom transformations on `DataFrame` elements.

`pandas_transformation.py`

# GroupBy Functionality

- ▶ **GroupBy (split-apply-combine) Overview:** The GroupBy operation allows data aggregation through specification of key columns for analysis of subsets of data. This refers to a chain of three steps:
  - ▶ **Split** a frame into groups.
  - ▶ **Apply** some operations to each of those smaller tables.
  - ▶ **Combine** the results.
- ▶ **Aggregation Functions:** Utilize functions like `sum()` and `mean()` to compute aggregated statistics applicable across grouped data subsets.
- ▶ **Flexible Grouping:** GroupBy supports flexible operations, accommodating different aggregation strategies based on dataset characteristics.

# Electroweak symmetry breaking mechanism



# Higgs decay event shapes

- ▶ Read `higgs-boson/training/training.csv` into pandas dataframe
- ▶ Inspect dataframe using `describe()` and `head()`
- ▶ Plot several distributions
- ▶ [Higgs Boson Machine Learning Challenge | Kaggle](#)

# Example distribution of Higgs boson and background event shapes

