Pandas and Numpy

Sergio Paniego Blanco
@sergiopaniego
sergiopaniegoblanco@gmail.com
https://sergiopaniego.github.io/

Pandas and numpy

- Introduction to numpy
- Data manipulation and analysis with Pandas.
- Mathematical operations and array manipulation with numpy
- Integration of pandas and numpy in python projects.

Intro to numpy

- Python library used for numerical computing.
- Provides:
  - Support for large, multi-dimensional array and matrices
  - A collection of mathematical functions to operate on these arrays effectively.
- Especially useful in data science, machine learning, and scientific computing.
- Allows for fast and memory-efficient operations on large datasets, which would be slower and more memory-intensive using Python's native data structures.
- Cheat Sheet

Intro to numpy

- Key features:
  - **Array Operations:** Provides a powerful N-dimensional array object (ndarray) for storing data.
  - **Mathematical Functions:** Enables a variety of mathematical operations, such as linear algebra, statistical calculations, and Fourier transforms.
  - **Broadcasting:** Allows operations between arrays of different shapes, which is useful for vectorized computations.
  - **Integration with Other Libraries:** Works seamlessly with libraries like **pandas**, scikit-learn, and TensorFlow, making it a foundational tool in the Python scientific ecosystem.

Intro to numpy

- Advantages to Python lists:
  - **Performance (speed and memory efficiency).** Faster especially when performing mathematical operations on large datasets. Numpy arrays consume less memory than Python lists.
  - **Vectorized operations.** Mathematical operations can be applied on entire arrays without needing explicit loops.
  - **Broadcasting** allows operations between arrays of different shapes, which can simplify code and reduce the need for loops.
  - **Wide range of mathematical and statistical functions.**
  - **Multi-dimensional arrays** (N-dimensional).

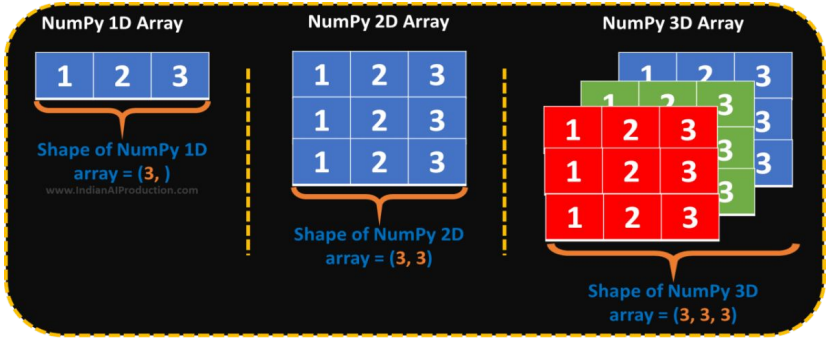
Intro to numpy

- Let’s see a comparison example.

```
pip install numpy
```

Numpy array creation

- Array creation
  - `np.array(list, dtype=data_type)`
  - `list` : list of elements
  - `data_type`: int, float…
- Other options:
  - `np.zeros(shape)`
  - `np.ones(shape)`
  - `np.arange([start], stop[, step])`
  - `np.linspace(start, stop, num=50)`



Numpy array creation

Numpy basic operations and functions

- Vectorized arithmetic operations.
- Mathematical and statistical functions:
  - `np.sum()`
  - `np.mean()`
  - `np.std()`
  - `np.min()`
  - `np.max()`

Numpy basic operations and functions

Numpy broadcasting

- **Broadcasting** allows operations between arrays of different shapes, which can simplify code and reduce the need for loops.

Numpy broadcasting

Numpy array manipulation

- **Indexing and slicing**
  - Selecting elements, rows and columns.
- **Reshaping with `reshape`.**
  - Changing the array shape without altering its data.
- **Concatenating arrays**
  - Merging arrays with `np.concatenate`.

Numpy array manipulation

ACTIVITY

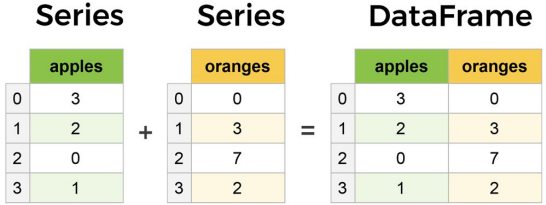
- **Exercise 1:** Create an array of 20 simulated temperature values and perform the following operations:
  - Calculate the mean, standard deviation, minimum, and maximum.
  - Add 2 degrees to each temperature value.
  - Reshape the array to a 4x5 matrix.
- **Exercise 2:** Given a quarterly sales array for two products, calculate the annual total and the quarterly average.
- **Exercise 3:** Use `np.where` to create a mask identifying values above the mean in the temperature array.

Data manipulation and analysis with Pandas

- Powerful and widely-used open-source data analysis and manipulation library for Python.
- Provides data structures and functions designed to make working with structured data both easy and efficient.

Data structures in Pandas

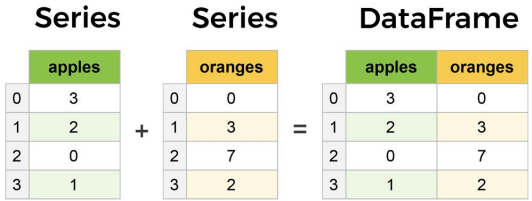
- Data structures
  - Series: one-dimensional labeled array that can hold any data type (integers, floats, strings, etc.)
  - DataFrame: two-dimensional labeled data structure (spreadsheet or SQL table). Consists of rows and columns. Can hold different data types in different columns.
- Data manipulation: wide range of functions.
  - Data cleaning
  - Data aggregation
  - Merging and joining.



- Data analysis: easily perform exploratory data analysis (EDA). `describe()` or `info()`.
- Input/Output: supports a variety of file formats (CSV, Excel, Json, SQL databases…).
- Time series: built-in support for time series data.
- Built on top of numpy

Data structures in Pandas

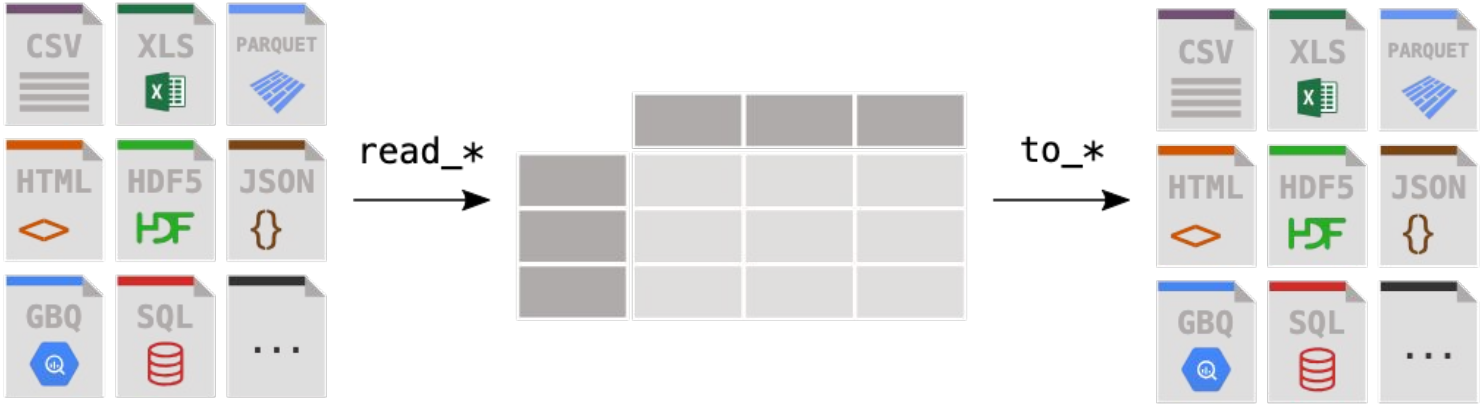
- Creating a Pandas Series:
  - `pd.Series(data)`: this function is used to create a Series from various data types such as lists, dictionaries, or NumPy arrays.
  - `pd.Series(data, index)`: allows you to specify a custom index for the Series.
- Creating a Pandas DataFrame:
  - `pd.DataFrame(data)`: this function is used to create a DataFrame from various data types such as lists, dictionaries, or NumPy arrays.
  - `pd.DataFrame(data, index)`: allows you to specify a custom index for the DataFrame.
  - `pd.DataFrame(data, columns)`: allows you to specify the order of the columns.



Data structures in Pandas

Importing and exporting data

- Reading and writing files:
  - Pandas allows reading and writing data from CSV, Excel, or JSON.



Importing and exporting data

Data manipulation

- Selection and filtering:
  - We can select rows and columns using .loc[] and .iloc[].
- Data cleaning:
  - The data may have missing values, duplicates, or outliers.
  - Pandas provides methods like .dropna(), .fillna(), or .duplicated()
- Creating new columns:
  - Pandas provides functionality for adding new data.

Data manipulation

Aggregation operations

- Statistical summaries
  - Pandas provides methods like `.describe()`, `.sum()`, `.mean()` to generate statistical summaries.
- Data grouping:
  - Pandas provides grouping functionality using `.groupby()`

Aggregation operations

ACTIVITY

- Load the Iris dataset https://raw.githubusercontent.com/mwaskom/seaborn-data/master/iris.csv
- Explore the data
  - Display the first 5 rows of the dataset.
  - Display the summary statistics.
  - Check for any missing values in the dataset.
- Data Manipulation: Create a new column that represents the ratio of `sepal_length` to `sepal_width`.
- Group and Aggregate: Group the dataset by species and calculate the average `sepal_length`, `sepal_width`, `petal_length`, and `petal_width`.
- Summarize Findings: Display the grouped results and interpret the average measurements by species.

Advanced operations with numpy and Pandas

- In this module:
  - Advances indexing techniques, conditional selection, and combining DataFrames from different sources.

Advanced operations in numpy

- Advanced Indexing and conditional selection:
  - Boolean indexing: allows you to filter numpy arrays based on conditions.
  - Fancy indexing: allow you to select specific elements from the array using integer indices.
- Using `np.where`.

Advanced operations in numpy

Advanced operations in Pandas

- Applying functions: apply functions to data structures in Pandas
  - `apply()`: apply a function along the axis of a DataFrame (rows or columns) or to a Series.
  - `map()`: primarily used with Series. It allows you to apply a function element-wise to the Series or to replace values based on a mapping
- Combining DataFrames:
  - `merge()`: used to combine two DataFrames based on common columns or indices, similar to SQL joins (inner, outer, left, right)
  - `join()`: used to join on indices, allowing for a simpler syntax when dealing with DataFrames with a common index.
  - `concat()`: used to concatenate multiple DataFrames along a particular axis (row-wise or column-wise).

Advanced operations in Pandas

ACTIVITY

Use the `apply()` function to perform transformations on a DataFrame, and then combine multiple DataFrames to generate insights on customer orders.

- Load the Data (`costumers.csv` and `orders.csv`)
- User the `apply()` function to create a new column in the `orders` DataFrame that applies a custom function to determine if the order amount is above a certain threshold (e.g., $30). The function will return "High" if it is above and "Low" if it is not.
- Calculating Total Spent: Now, let's calculate the total amount spent by each customer using `groupby()`.
- Combining DataFrames: Merge the `customers` DataFrame with the `total_spent` DataFrame using the `merge()` function to associate each customer with their total spending.
- Data Cleaning: Check for any customers who haven't made any orders (total_spent is NaN) and fill these values with 0.
- Display the Result: Print the resulting DataFrame, `customer_summary`, to see the total spending of each customer along with their order amount categories.

Integrating Pandas and numpy in Python projects

- Let’s how we can integrate Pandas and numpy in a Python project.
  - Integration in Django
    - Data preprocessing with Pandas and numpy
  - Data processing in Django Views

Integrating Pandas and numpy in Python projects

- Let’s how we can integrate Pandas and numpy in a Python

```
django-admin startproject sales_analysis
cd sales_analysis
python manage.py startapp sales
```

```
INSTALLED_APPS = [ ... 'sales', ]
```

Integrating Pandas and numpy in Python projects

- Define models

```
# models.py
from django.db import models

class SalesData(models.Model):
    product_name = models.CharField(max_length=255)
    sales_amount = models.FloatField()
    date = models.DateField()

class SalesReport(models.Model):
    month = models.CharField(max_length=20)
    total_sales = models.FloatField()
    top_product = models.CharField(max_length=255)
```

```
python manage.py makemigrations
python manage.py migrate
```

Integrating Pandas and numpy in Python projects

- Define form

```
# forms.py
from django import forms

class UploadFileForm(forms.Form):
   file = forms.FileField()
```

Integrating Pandas and numpy in Python projects

- Define views.py

```
import pandas as pd
from django.shortcuts import render
from .models import SalesData, SalesReport
from .forms import UploadFileForm
from django.db.models import Sum

def load_sales_data(request):
        if request.method == 'POST':
                form = UploadFileForm(request.POST, request.FILES)
                if form.is_valid():
                        file = request.FILES['file']
                        df = pd.read_csv(file)
                        # Example preprocessing
                        df['date'] = pd.to_datetime(df['date'])
                        df['sales_amount'] = df['sales_amount'].astype(float)
                        # Save to the database
                        for _, row in df.iterrows():
                                SalesData.objects.create(
                                        product_name=row['product_name'],
                                        sales_amount=row['sales_amount'],
                                        date=row['date']
                                )
                        return render(request, 'upload_success.html')
        else:
                form = UploadFileForm()
                return render(request, 'upload.html', {'form': form})

def generate_report(request):
        # Aggregate sales data by month
        monthly_sales = (
                SalesData.objects.values('date__year', 'date__month')
                .annotate(total_sales=Sum('sales_amount'))
                .order_by('date__year', 'date__month')
        )
        # Get top-selling products
        top_products = (
                SalesData.objects.values('product_name')
                .annotate(total_sales=Sum('sales_amount'))
                .order_by('-total_sales')[:5]
        )

        context = { 'monthly_sales': monthly_sales, 'top_products': top_products, }
        return render(request, 'report.html', context)
```

Integrating Pandas and numpy in Python projects

- Define templates

```
# upload.html
<h1>Upload Sales Data</h1>
<form method="post" enctype="multipart/form-data">
     {% csrf_token %}
     {{ form.as_p }}
     <button type="submit">Upload</button>
</form>
```

```
# upload_sucess.html
<h1>Data Uploaded Successfully!</h1>
<a href="{% url 'generate_report' %}">Generate Sales Report</a>
```

Integrating Pandas and numpy in Python projects

- Define templates

```
# report.html
<h2>Monthly Sales Report</h2>
<table>
        <tr>
                <th>Year</th>
                <th>Month</th>
                <th>Total Sales</th>
        </tr>
        {% for sale in monthly_sales %}
        <tr>
                <td>{{ sale.date__year }}</td>
                <td>{{ sale.date__month }}</td>
                <td>{{ sale.total_sales }}</td>
        </tr>
        {% empty %}
        <tr><td colspan="3">No sales data available.</td></tr>
        {% endfor %}
</table>
<h2>Top-Selling Products</h2>
<table>
        <tr>
                <th>Product Name</th>
                <th>Total Sales</th>
        </tr>
        {% for product in top_products %}
        <tr>
                <td>{{ product.product_name }}</td>
                <td>{{ product.total_sales }}</td>
        </tr>
        {% empty %}
        <tr><td colspan="2">No product data available.</td></tr>
        {% endfor %}
</table>
```

Integrating Pandas and numpy in Python projects

- Configure urls

```
#sales/urls.py
from django.urls import path
from .views import load_sales_data, generate_report

urlpatterns = [
     path('upload/', load_sales_data, name='load_sales_data'),
     path('report/', generate_report, name='generate_report'),
]
```

```
#sales_analysis/urls.py
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
     path('admin/', admin.site.urls),
     path('sales/', include('sales.urls')),
]
```

Integrating Pandas and numpy in Python projects

- Let's run the server

```
python manage,py runserver
```

- Go to http://127.0.0.1:8000/sales/upload/

Advanced operations in Pandas

ACTIVITY

- Extend the existing sales report by calculating and displaying the total sales per quarter.
- Instructions:
  - Calculate Quarterly Sales in the `generate_report` View:
    - Update the view to calculate quarterly sales using Django's `Sum` aggregation.
    - Group the data by quarter, using the date of each sale to determine the quarter.
  - Update the Template:
    - Add a new table to `generate_report.html` that displays the quarterly sales data.

Next steps


TensorFlow


K
Keras


matplotlib


NumPy


pandas


seaborn


scikit
learn