Tutorial on pandas

Working with *real-world data* and CSV files with pandas.

- Tutorial on pandas
  - Prerequisites
  - Reading, inspecting and writing data
  - Selecting columns
  - Selecting rows
  - Selecting rows and columns
  - Ploting with Pandas
  - Practice
  - Questions left as an exercise
  - References

Prerequisites

- PyCharm is installed
- pandas, numpy and matplotlib packages are installed

Reading, inspecting and writing data

For this tutorial, we will use a real-world dataset of Titanic passenger data.

1. Retrieve Titanic data and **copy/paste the data** into a file called `titanic.csv` in your current PyCharm Project.
2. **Create** a Python program called `analyze_titanic.py`. Import pandas

```
# analyze_titanic.py
import pandas as pd

# Your code...
```

3. **Read** the Titanic passenger data stored in the csv file with the **`read_csv`** function:

```
titanic = pd.read_csv("titanic.csv")
```

4. **Inspect** the type of the variable **`titanic`**:

```
print(type(titanic))
```

The **`read_csv`** function to read data stored as a csv file into a pandas **`DataFrame`**. A pandas DataFrame is one of the most powerful and widely used data structures in Python, particularly in data science, machine learning, and data analytics. This type is made to ease data cleaning, manipulation, alignment, grouping, and tabular analysis.

> Unlike Numpy **`arrays`**, **`DataFrames`** are strictly 2-dimensional.

5. Make sure to always have a check on the data after reading in the data. When displaying a DataFrame, the first and last 5 rows will be shown by default:

```
print(titanic)
```

6. **Inspect** the *first* 8 rows with the **`head`** function:

```
print(titanic.head(8))
```

7. **Inspect** the *last* 8 rows with the **`tail`** function:

```
print(titanic.tail(8))
```

8. **Check** how pandas interpreted each of the column *data types* by requesting the pandas **`dtypes`** attribute:

```
print(titanic.dtypes)
```

For each of the columns, the used data type is enlisted. The data types in this DataFrame are integers (**`int64`**), floats (**`float64`**) and strings (**`object`**).

9. **Save** the **`DataFrame`** to an Excel file (**`.xlsx`**) to share data with your colleagues:

```
titanic.to_excel("titanic.xlsx", sheet_name="passengers", index=False)
```

10. **Print** a summary of the dataset with the **`info`** function:

```
print(titanic.info())
```

We can see that:

- There are 891 entries, i.e. **891 rows**.
- Each row has a row **label** (aka the index) with values ranging from 0 to 890.
- The table has **12 columns**. Most columns have a value for each of the rows (all 891 values are non-null). Some columns do have missing values and less than 891 non-null values.
- The columns Name, Sex, Cabin and Embarked consist of textual data (**`strings`**, *aka* **`object`**). The other columns are numerical data, some of them are whole numbers (integer) and others are real numbers (float).

> The approximate amount of RAM used to hold the DataFrame is provided as well.

Selecting columns

1. **Select** the Age column to retrieve the age of the Titanic passengers:

```
ages = titanic["Age"]
print(ages)
```

> To select a single column, use square brackets **[]** with the column name of the column of interest.

Each column in a **DataFrame** is a **Series**. As a single column is selected, the returned object is a pandas **Series**. We can verify this by checking the type of the output:

```
print(type(ages))
```

2. **Select** the age and sex of the Titanic passengers:

```
age_sex = titanic[["Age", "Sex"]]
print(age_sex.head())
```

The returned data type is a **DataFrame**! The selection returned a **DataFrame** with 891 rows and 2 columns.

> Remember, a **DataFrame** is strictly 2-dimensional with both a row and column dimension.

Selecting rows

1. We are interested in the passengers older than 35 years:

```
above_35 = titanic[titanic["Age"] > 35]
print(above_35)
print(type(above_35))
```

> To select rows based on a conditional expression, use a condition inside the selection brackets [].

2. Let's have a look at the number of rows which satisfy the condition by checking the shape attribute of the resulting DataFrame **`above_35`**:

```
print(above_35.shape)
```

Selecting rows and columns

1. We are now interested in the *names* of the passengers older than 35 years:

```
adult_names = titanic.loc[titanic["Age"] > 35, "Name"]
```

In this case, a subset of both rows and columns is made in one go and just using selection brackets [] is not sufficient anymore. The **`loc`**/**`iloc`** operators are required in front of the selection brackets **`[]`**.

When using **`loc`**/**`iloc`**, the part *before* the comma is the *rows* you want, and the part *after* the comma is the *columns* you want to select.

> **`loc`** is label-based, while **`iloc`** is integer-position-based.

2. We can also select rows 10 till 25 and columns 3 to 5, by slicing the DataFrame:

```
titanic.iloc[9:25, 2:5]
```

Ploting with Pandas

Pandas has *built-in*, seamless integration with Matplotlib! **`pandas`** provides a convenient **`.plot()`** method *directly* on **`DataFrames`** and **`Series`** that acts as a *wrapper* around Matplotlib.

1. We need to use and import **`matplotlib`**. **Add** the following import statement:

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

2. We want to see the *repartition* of men and women on the Titanic. We need to count the occurrences of each category with the **`value_counts`** function:

```
sex = titanic[["Sex"]]
sex.value_counts().plot(kind="bar")

plt.title("Titanic Passengers by Sex")
plt.xlabel("Sex")
plt.ylabel("Count")
plt.show()
```

3. To display the bars as percentages instead of raw counts, normalize the data by dividing each value by the total and multiplying by 100:

```
(sex.value_counts(normalize=True)*100).plot(kind="bar")
```

4. To display the percentage breakdown specifically for *survivors*, we need to filter the **`DataFrame`** to keep *only* the passengers where **`Survived == 1`**:

```
survivors = titanic[titanic["Survived"] == 1]
survivor_percentage = survivors["Sex"].value_counts(normalize=True) * 100
survivor_percentage.plot(kind="bar", color=["#2ecc71", "#3498db"])
```

Did *gender* affect survival?

5. Let's plot the *age distribution* of all Titanic passengers using an *histogram*:

```
titanic["Age"].plot(kind="hist", bins=20, color="#3498db",
        edgecolor="white")
```

6. We would now like to see *the age distribution for survivors*. We can filter the DataFrame *first*, and then plot a histogram of their ages, just like we did before. We are going to plot *both* age distribution of all Titanic passengers and of survivors to be able to compare then easily:

```
# Plot all passengers in light gray/blue
titanic["Age"].plot(kind="hist", bins=20, alpha=0.5, label="All
        Passengers", color="gray", edgecolor="white")

# Plot survivors on top in green
titanic[titanic["Survived"] == 1]["Age"].plot(kind="hist", bins=20,
        alpha=0.7, label="Survivors", color="#2ecc71", edgecolor="white")

plt.title("Age Distribution: All Passengers vs. Survivors")
plt.xlabel("Age")
plt.ylabel("Count")
plt.legend()
```

According to the result, did *age* affect survival?

7. *Did 1st-class passengers have a higher survival rate than 3rd-class passengers?* Let's see…

First, we need to *group* passengers *by class* with the **`groupby`** function.

In pandas, **`groupby()`** is one of the most powerful and frequently used tools for data analysis. It allows you to **group your rows together based on the values in one or more columns** so that you can perform calculations **on each group independently**.

```
p_by_class = titanic.groupby("Pclass")
print(p_by_class.count()) # number of passengers by class
```

Let's select the **`Survival`** column and compute the proportion of survivors *per* class:

```
# mean() allows to compute the percentage
pclass_survival = titanic.groupby("Pclass")["Survived"].mean() * 100

pclass_survival.plot( kind="bar", edgecolor="black")

plt.title("Survival Rate by Passenger Class")
plt.xlabel("Passenger Class (1 = 1st, 2 = 2nd, 3 = 3rd)")
plt.ylabel("Survival Rate (%)")
plt.ylim(0, 100)
plt.grid(axis="y", linestyle="--", alpha=0.6)
plt.show()
```

So, did 1st-class passengers have a higher survival rate than 3rd-class passengers? …

Practice

Write Python instructions (directly in the console) with pandas to find the:

1. list of survivors
2. list of survivor's names only
3. number of survivors
4. number of survivors who were female
5. average age of survivors
6. median age of survivors

Questions left as an exercise

Use **pandas** to answer other interesting questions. For example:

- What was the survival rate of women in 3rd class *versus* women in 1st class?
- How much did passengers typically pay for their tickets? (median, average, distribution)

References

This tutorial is based on the following official pandas *getting started tutorials*:

- How do I read and write tabular data?
- How do I select a subset of a DataFrame?
- How do I create plots in pandas?