DI Martin Uray
martin.uray@fh-salzburg.ac.at

Applied AI with Python
Assignment Sheet 3: Visualizing Continuous and Time Series Data

Version:
2024-11-19

To prepare for the next session of the course, students are expected to complete the following exercises. At the beginning of the session, a handout will be distributed containing a list of exercises. Students will be required to tick off their individual achievements as evidence of their work. During the session, students will be randomly selected from those present and will be asked to present (defend) their solutions. A brief interview will follow, during which the student's solution will be discussed. If any doubts arise regarding a student's presentation that they are unable to explain, no credit will be awarded for that exercise in the final grading. To ensure a collective learning experience, the instructor will also review and discuss the presented work with the plenum.

1 Continuous Data

In this part, we are going to work with a popular wine data set containing measurements for different attributes of wines. This makes it a perfect example to learn on how to handle continuous data. The data set itself is available to you via scikit-learn.

Ex. 16 Data Exploration and Cleaning: For your first assignment, you are asked to complete the following steps after loading the `winedata.csv` using pandas:

(a) Try to get an idea of what exactly is stored in the data. Check the size of the dataset (columns and rows) and see if you can find out more about the columns and its values.

(b) Call the describe() method of your `DataFrame`. What do you see?

(c) Check for missing values (NaNs), calculate their occurrences and treat missing values correctly according to one of the methods shown to you in the lecture. Be prepared to defend your choice!

Ex. 17 Apply and Visualize Normalization: One technique often used in data preprocessing, especially when dealing with features in differing value ranges, is normalization. Through normalization, differences in value ranges for multiple columns in a data set can be compensated by (e.g.) transforming each feature in such a way that its mean is 0 with standard deviation 1. While this sounds difficult, it is actually pretty easy to do, and we'll use scikit-learn's StandardScaler for it. So in short:

(a) Use seaborn's boxplot() function **before** applying normalization on your cleaned data set.

(b) Transform the data by correctly applying the `StandardScaler`.

(c) Redraw the `boxplot` and be prepared to talk about the differences you can make out!

Ex. 18 Apply and Visualize Dimensionality Reduction: Trying to figure out similarities between the data points (wines) in our data set is a challenging task. As it stands, we have thirteen features describing each wine. If we want to use figures to visualize the relationships between the different wines, we need a way to reduce the dimensionality of the dataset to two dimensions. There are many different ways to do this. You may, for example, simply select two features from the thirteen by hand to achieve this. This approach has some limitations though, as you lose all the information present in the other eleven columns.

Other techniques are PCA and TSNE, both well known and state-of-the-art methods to implement dimensionality reduction. For now, knowing how to apply these techniques on a data set is enough. However, if you want to know a bit more about how these techniques work, check the scikit-learn documentation. Complete the following steps to finish this assignment:

(a) Instantiate an object of your preferred method for dimensionality reduction and don't forget to parametrize this object: make sure to set the **number of target dimensions to 2**.

(b) Visualize the resulting dimensions in a scatter plot.

(c) Use the dataset's labels to add color to your scatter plot indicating the vineyard each data point was produced in. Do not forget to add a **legend** to your plot.

(d) Be prepared to explain what you see in our next session!

(e) Be prepared to explain the dimension reduction method used!

2 Time Series Data

In this part, you will work with electricity data from the Open Power System Data (OPSD). The data set includes daily totals of power consumption and production (solar and wind) in gigawatt hours (GWh) from 2006–2017 for Germany. The dataset can be found using this link.
You can find more information about the data set as well as an in-depth tutorial on how to work with the data set in a blog by Jennifer Walker: Time Series Analysis with Pandas (Tutorial). Also have a look at Working with Time Series in Python (Tutorial) if you're interested in further reading on the topic.

Getting Started

Pandas offers the `TimeStamp` class as well as the `to_datetime()` function to handle temporal data. Dive into working with time series in pandas by using the example data set containing energy data of Germany. To begin, complete the following steps:

1. Read the `opsd_german_daily.csv` into a `DataFrame` and call its `head()` and `describe()` methods.
2. Check the column's datatype. With what you just learned about `TimeStamps`, can you identify the steps you need to take to make the `Date` column usable in time series plots?

Important

Before you start with the actual assignments, please ensure that you set the `index` of your `DataFrame` to the `Date` column. Also, add new columns containing the year, month (January–December) and weekday (Monday–Sunday) for each row in the dataframe. The quickest way to do this correctly is by relying on the methods provided by pandas' `DateTimeIndex`. Also, check out time-based indexing to get to know a quick way to select the data you need!

Ex. 19 Germany's Daily Energy Consumption 2006–2017: Use an appropriate plotting function to plot a time series of Germany's daily energy `Consumption from 2006--2017`. The plots need to be self-contained and be tweaked to show all features idependently.

Ex. 20 Subplots Showing Daily Consumption and Solar-, Wind-Generation Next, generate **one** figure with **three subplots** containing time series for `Wind`, `Solar` and energy `Consumption`. Ensure that you properly represent the different magnitudes of renewable energy generation and power consumption by **scaling the y-axis accordingly**!

Ex. 21 Can you Spot Patterns? Select a single year from the dataset using pandas' time-based indexing for further examination. Plot this year's `Consumption` in a time series plot. Be prepared to comment on what you see next session!
Next, find a way to plot the distribution of the monthly energy `Consumption` in Germany from 2006–2017. What do you see?

Ex. 22 Calculate and Visualize Monthly Consumption and Generation: First, resample the data to change the resolution of our dataset to reflect the monthly sums of energy `Consumption` and generation (split into `Wind` and `Solar`). To do so fairly easily, make yourself familiar with `df.resample(...)`.
Finally, create **a single time series plot** containing monthly sums of `Consumption`, `Wind` and `Solar` from 2006–2017. Try out different colors and styles (e.g. area plots, line plots, ...) within this single plot and make sure to **add a legend**!

Ex. 23 Use Your Own Data! Go to Wiki Pageviews and download a dataset of the page views of a Wikipedia page of your choice.

(a) Enter the page name(s) you want to find out more about.
(b) Click into the Dates field and choose "All time", then download the `.csv` file and load it into your notebook.
(c) Change the index to make use of time-based indexing (`DateTimeIndex`).
(d) Inspect the dataset (missing values?)
(e) Check for patterns in your data (e.g. selecting just one year/month, grouping monthly/weekly, ...)
(f) Finally, show **at least one plot** with a weekly aggregate of numbers.