Chapter 1 - Getting Started with Time Series Analysis

Time Series All Around Us

Whenever we are dealing with measurements collected throughout time at specified intervals, it becomes obvious that we are working with time series data and its specific challenges and complexities.

Time series data can take on different properties, by the nature of the data domain, thus requiring the practitioner to be familiar with the different methods and techniques available for analysis. These properties cluster time series data into different buckets or types depending on whether we are dealing with stationary, non-stationary, linear, non-linear, seasonal, non-seasonal, univariate, or multivariate time series data.

Time series data are prevalent in the financial world, but in reality, time series data is all around us existing in different domains and disciplines. Recent events have further simulated interest in learning time series analysis and prediction techniques, whether it is classic statistical methods or advanced machine learning algorithms, such as building COVID-19 prediction models, predicting presidential election results, analyzing stock, or cryptocurrency data.

Some examples of where time series data can be found by domain:

* Business: Marketing planning, Inventory management, Product Demand Planning, Resource Planning, Churn analysis
* Finance: Stock analysis, Budget forecasting, Sales forecasting, Volatility Modeling
* Government: Election forecast, Economic forecasting, Gross Domestic Product (GDP), Unemployment Rate, Population migration rate, Birth Rate
* Science: Weather forecasting, Earthquake prediction, Air quality, Species population growth
* Medical: Infectious disease transmission, Electrocardiogram monitoring (ECG or EKG), Healthcare cost prediction, Blood glucose monitoring, Hospital capacity
* Engineering: Predictive maintenance, Signal processing, Production decline analysis, Traffic volume forecasting
* Technology: Log Data, Web Traffic, Internet of Things (IoT), Server Utilization Demand
* Prevention: Credit card anomaly and fraud detection, Non-technical power loss detection, Crime rates

When working with time series data we usually have two goals:

1. Perform time series analysis (descriptive)
2. Build models through time series forecasting (predictive).

In time series analysis we strive to extract a better understanding and deeper intuition into the underlying phenomenon captured in our data using statistical methods. In time series forecasting, we aim to create a predictive model that extends from our data into the unforeseeable future and hence the term forecasting.

Time series data differs from the typical data used in machine learning in the classical sense due to the dependence on time, serial correlation, and dependence.

In this book, we cover a variety of recipes for both analysis and forecasting of time series data and pragmatic approach to handling the complex nature of time series data.

Time Series Analysis Environment Setup

As we dive into the various recipes provided in this book, it is important to create a Python virtual environment which provides an isolated Python environment in order to install all our dependencies without impacting other Python projects.

We will cover two methods to creating a Python virtual environment in this book. We will cover the use of conda or venv. If you are already using Anaconda <https://www.anaconda.com/products/individual> then Conda would be the preferred choice since it provides package dependency and environment management for Python (and supports other languages). Venv is a Python module that provides environment management that comes as part of the standard library in Python 3.3 or newer and requires no additional installation.

With these tools one can create multiple virtual environments for different Python projects that may require different Python versions (e.g. 2.7 or 3.8), or different Python packages and associated versions, or if experimenting with new packages to better understand how they work. This is a best practice taken by many developers and data science practitioners as to not over clutter the primary Python installation or creating version conflicts between packages.

Conda vs venv

If you have Anaconda or Miniconda installed, then using conda would be the preferred method to create a virtual environment. In this book, we will show different ways to install packages using conda install or pip install.

How to do it…

Using conda

Windows Users

On Windows OS you will need to use the Anaconda Prompt. You can access that by clicking on the Windows Icon, and start typing “Anaconda” to find matches. In the result pane, look for Anaconda Prompt, click that to open the Anaconda terminal.

1. Update conda to the current version

$ conda update conda

1. Create a new virtual environment named timeseries using Python 3.8

$ conda create -n timeseries python=3.8

Where -n is a shortcut for --name

1. Conda may identify additional packages that need to be downloaded and installed. You may be prompted to whether to proceed or not. Hit enter, or type y then enter, to proceed. You may get a different list of packages to download/install in comparison to the screen shot below.



Once you type y and hit enter, conda will proceed with the setup

Text

Description automatically generated

1. To activate the new virtual environment type

$ conda activate timeseries

1. To further validate the setup, you can run the following

$ conda info –envs

This will list all the conda environments that have been created and available. timeseries should be listed with an “\*” in front indicating it is the active environment.

1. To deactivate the environment

$ conda deactivate

1. To completely delete the environment and all the libraries installed

$ conda env remove -n timeseries

Where -n is a shortcut for --name

Using venv

1. Decide where to place the new virtual environment and specify the path, in this example I already navigated to Desktop and ran the command as shown here

$ python3 -m venv timeseries

In the code above we created a timeseries directory, which did not exist before, and all the supporting subdirectories, standard libraries, and other supporting files are all included.

Notice the use of python3 to indicate that we are creating a Python 3 virtual environment. The same would have been achieved with following code to create a Python 3.8 virtual environment.

$ python3.8 -m venv timeseries

If we navigate to Desktop\timeseries we can confirm the existence of the additional files and subfolders inside timeseries

Graphical user interface

Description automatically generated with low confidence

1. We can then activate our new environment

On Mac/Unix

$ source timeseries/bin/activate

We can confirm that the right Python env is activate

$ which python

And you should get a response that reflects the path to the Python interpreter

/Users/tarekatwan/Desktop/timeseries/bin/python

On Windows

$ timeseries/Scripts/activate.bat

1. To deactivate

$ deactivate

How it works…

Once we activate the virtual environment, we can check on the location of the Python interpreter to confirm that we are using the right one.

$ which python

This will show the path of our Python environment which contains all the necessary files including the Python interpreter.

The output of the statement above will show different outputs depending if it was a conda environment or venv environment

# when using conda

/Users/tarekatwan/opt/anaconda3/envs/timeseries/bin/python

# when using venv

/Users/tarekatwan/Desktop/timeseries/bin/python

Any additional packages or libraries that we install will be isolated from other environments and reside in the appropriate path and folder structures.

If we compare the folder structures of both venv and conda we can see similarities

Table

Description automatically generated

Conda timeseries environment

Graphical user interface, application

Description automatically generated

venv timeseries environment

Notice, that when using conda, all environments will default to the /envs/ location inside the anaconda3/ directory. When using venv, we actually had to specify where we plan to create the directory or project, thus giving us a little more flexibility on where we want to create our project.

The advantage of conda on the other hand, is that with conda we get two features: a package and dependency manager and a virtual environment manager. This means we can use the same conda command to create additional environments using conda create, and also install packages using conda install <package name> which we will see later on.

When using venv, keep in mind that venv is only a virtual environment manager, and we will still need to rely on pip as our package manager to install any new packages e.g. pip install <package name>

One of the added benefits of using conda is that we can create environments for other languages and not just Python. This includes Julia, R, Lua, Scala, Java, and more.

Pip vs Conda

Both Pip and Conda are very popular methods for managing packages in Python. It is very common to see these two options listed under the installation instructions of many popular Python libraries.

A picture containing graphical user interface

Description automatically generated

Example PyTorch installation page showing options for Conda and Pip.

There’s more…

Using a YAML file

We can also create a virtual environment from a YAML file. This option gives us great control in defining many aspects of the environment including all packages to be installed, in one step.

Here is an example of a YAML file that creates a conda environment labeled tscookbook

env.yml

name: tscookbook

channels:

- conda-forge

- defaults

dependencies:

- python=3.8

- pip

# Data Analysis

- statsmodels

- scipy

- pandas

- numpy

- pyqt

- tqdm

# Plotting

- matplotlib

- plotly

- seaborn

- pandas

- numpy

# Machine learning

- scikit-learn

# Jupyter Environment

- jupyter

- jupyter\_client

- jupyter\_console

- jupyter\_core

- jupyterlab

- notebook

- ipython

- jupyter\_contrib\_nbextensions

To create the virtual environment using the env.yml file we use conda env create -f

$ conda env create -f env.yml

once the process is completed, you can activate the environment

$ conda activate tscookbook

Using Docker

In addition to directly installing Anaconda on our local machine, there is another option to completely isolate the whole Python installation by using Docker. You can even create additional virtual environments inside the container.

There are three simple options using official images on Docker Hub:

* Using the official Docker Anaconda image

$ docker pull continuumio/anaconda3

* Using the official Python image

$ docker pull python

* Using one of the official Jupyter Docker Stacks. To read more about the available stacks please refer to their official documentation here <https://jupyter-docker-stacks.readthedocs.io/en/latest/using/selecting.html>

$ docker pull jupyter/datascience-notebook

See also

If you are using a machine that does not allow you to install any software or using an older machine with limited capacity or performance, then do not worry. There are still other options to get you moving and following through the recipes in this book.

Alternative options:

* Replit offers a free, in-browser IDE that supports more than 50+ languages including Python. All you need is to create an account and create your new replit space.
  + <https://replit.com/>
* Google Colab are hosted Python notebooks that already has some of the most popular data science packages preinstalled, including Pandas, Scikit-Learn, and Tensorflow. Colab allows you to install additional packages as well from within the notebook. A great feature with Colab, is that you get the option to configure your notebook to use a CPU, GPU, or a TPU all for free.
  + https://colab.research.google.com/

Installing Python Libraries

We will install all the necessary libraries and specific versions to be able to follow along with this book.

The easiest way to install many libraries at once is to use a requirements.txt file. A requirements.txt file has been provided in the GitHub repo <link> so that you can use to install all required libraries.

How to do it…

* Using Conda
  + Option 1: Create a new conda environment and install libraries in one step

$ conda create --name <env> -f requirements.txt

* + Option2: Install libraries in an existing conda environment

$ conda activate timeseries

$ conda install -f requirements.txt

* Using Venv and Pip
  + On a Mac/Linux: Activate the venv environment then install the packages

$ source timeseries/bin/activate

$ pip install -r requirements.txt

* + On a Windows: Activate the venv environment then install the packages

$ timeseries/Scripts/activate.bat

$ pip install -r requirements.txt

How it works…

There are two methods to create the requirements.txt which is generated from an existing virtual environment. This is useful when recreating environments to ensure reproducibility and consistency across your work. Say you worked on a project and installed specific libraries, and you want to ensure that when you share your code the other user has the same settings as yours. This is when generating the requirements.txt file can come in handy (or YAML file as show earlier) so the users can create their new virtual environment and install the same versions of the packages you have instructed in the requirements.txt.

With this scenario in mind, I am sharing my configurations to ensure we are using the same library versions for reproducibility and consistency when following along.

Both methods will export the list of packages already installed and their current versions. Example below of a requirements.txt file format:

pandas==1.2.3

numpy==1.19.2

statsmodels==0.12.2

scipy==1.6.2

* Pip freeze

pip freeze > requirements.txt

* Conda

conda list -e > requirements.txt

Generating Time Series Data

In the absence of a dataset to load, sometimes it is useful to be able to generate dummy time series data for the purpose of testing a new method, learning a new concept, or exploring a new script.

Getting Ready

In this recipe will use Pandas library to generate time series data and then explore ways to change the resample our time series data for different scenarios.

How to do it…

1. Use the date\_range function to generate a fixed frequency DateTimeIndex.

>>> dates = pd.date\_range(start='1/10/2021', periods=100, freq='H')

1. Generate dummy data. In our example we will generate some sales data

>>> sales = np.random.randint(100, 500, size=len(dates))

1. Generate a Pandas Series where the index is the dates generated

>>> ts = pd.Series(data=sales, index=dates, name='sales')

1. Validate the data generated

>>> ts.head()

2021-01-10 00:00:00 163

2021-01-10 01:00:00 120

2021-01-10 02:00:00 419

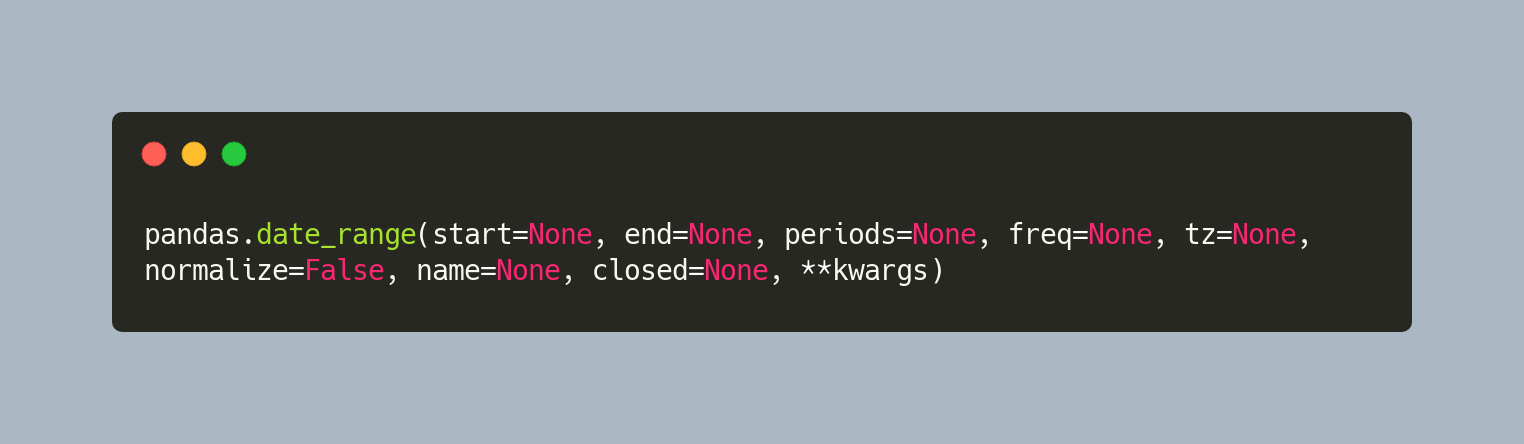
2021-01-10 03:00:00 419

2021-01-10 04:00:00 242

Freq: H, Name: sales, dtype: int64

How it works…

Using pandas.date\_range method to generate a fixed frequency DateTimeIndex. Here is the general syntax for date\_range:



If you just run pandas.date\_range() without any arguments you will get the error below

ValueError: Of the four parameters: start, end, periods, and freq, exactly three must be specified

The message is clear, at least three must be specified. In the code earlier we specified start, periods, and freq.

Common Frequency Strings

* D : Daily
* M : Monthly
* W : Weekly
* H : Hourly
* Q : Quarterly
* B : Business Day

Frequency Multiples

We can also specify multiples for any of the frequency strings such as ‘2D’ which reflect a 2 day increment while ‘3M’ reflects a 3 month increment

Example

>>> pd.date\_range(start='1/10/2021', periods=10, freq='2D')

Would produce

DatetimeIndex(['2021-01-10', '2021-01-12', '2021-01-14', '2021-01-16',

'2021-01-18', '2021-01-20', '2021-01-22', '2021-01-24',

'2021-01-26', '2021-01-28'],

dtype='datetime64[ns]', freq='2D')

Also note the four examples below with different start date formats would produce the same output.

>>> pd.date\_range(start='1/10/2021', periods=10, freq='D')

>>> pd.date\_range(start='1-10-2021', periods=10, freq='D')

>>> pd.date\_range(start='10-JAN-2021', periods=10, freq='D')

>>> pd.date\_range(start='2021-1-10', periods=10, freq='D')

There’s more…

Using Panda’s resample method, we can resample our time series from one frequency to another whether downsampling or upsampling. In this example, we will downsample the time series from an hourly (H) frequency to a daily (D) frequency.

1. The data generated earlier was hourly (H). We can downsample the data from hourly to daily frequency using resample. We will specify that we will take the last value.

>>> ts.resample('D').last()

2021-01-10 295

2021-01-11 381

2021-01-12 291

2021-01-13 340

2021-01-14 370

Freq: D, Name: sales, dtype: int64

1. Additionally, we can also generate a completely different dataset, similar to data representation we would expect from stock data using ohlc method, which gives us open, high, low, and close values.

>>> ts.resample('D').ohlc()

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **open** | **high** | **low** | **close** |
| **2021-01-10** | 163 | 498 | 120 | 295 |
| **2021-01-11** | 156 | 465 | 141 | 381 |
| **2021-01-12** | 137 | 475 | 117 | 291 |
| **2021-01-13** | 155 | 497 | 105 | 340 |
| **2021-01-14** | 426 | 426 | 359 | 370 |