Pandas Basics

Slide 0 and 1.
Hi, hello and welcome to the first section of the 'Data Analysis With Pandas' course. In this pandas first section we learn about the basics of Pandas data structures. Pandas is a powerful data analysis package built on top of NumPy and provides an efficient implementation of a DataFrame. DataFrames are essentially multidimensional arrays with attached row and column labels, and often with heterogeneous types and/or missing data.

While pandas adopts many coding idioms from NumPy, the biggest difference is that pandas is designed for working with tabular or heterogeneous data. NumPy, by contrast, is best suited for working with homogeneous numerical array data. It also offers a convenient storage interface for labeled data, Pandas implements a number of powerful data operations familiar to users of both database frameworks and spreadsheet programs.

Pandas and in particular its Series and DataFrame objects, builds on the NumPy array structure and provides efficient access to "data munging" tasks that occupy much of a data scientist's time. Pandas objects can be thought of as enhanced versions of NumPy structured arrays in which the rows and columns are identified with labels rather than simple integer indices.

Slide 2.

Pandas offers a three fundamental data structures: the Series, DataFrame, and Index.
A Pandas Series is a one-dimensional array of indexed data.

Slide 3.

The essential difference is the presence of the index: while the NumPy array has an implicitly defined integer index used to access the values, the Pandas Series has an explicitly defined index associated with the values.

This explicit index definition gives the Series object additional capabilities. The Pandas Series can be constructed from the constructor.

pd.Series (data, index=index).

Data and index are the two important parameters to construct the series object. Where index is an optional argument and data can be one of many entities.

Slide 4.
The next fundamental structure in Pandas is the DataFrame. The DataFrame can be thought of either as a generalization of a NumPy array, or as a specialization of a Python dictionary.
Here the columns names means the names used to represent the columns and index names means the names used to represent the row index of the DataFrame object.

If a Series is an analog of a one-dimensional array with flexible indices, a DataFrame is an analog of a two-dimensional array with both flexible row indices and flexible column names.

Slide 5.

Just as we might think of a two-dimensional array as an ordered sequence of aligned one-dimensional columns, we can think of a DataFrame as a sequence of aligned Series objects. Here, by "aligned" we mean that they share the same index.

Slide 6.

Both the Series and DataFrame objects can have an explicit index that allow us to reference and modify data.
This Index object is an interesting structure in itself, and it can be thought of either as an immutable array or as an ordered set (technically a multiset, as Index objects may contain repeated values).
The Index object in many ways operates like an array. One difference between Index objects and NumPy arrays is that indices are immutable, that is, they cannot be modified via the normal means.
The Index object follows many of the conventions used by Python's built-in set data structure, so that unions, intersections, differences, and other combinations can be computed in a familiar way.

Slide 7.

Here I've highlighted the index of the DataFrame object to show that, the immutability of this explicit index object makes it safer to share indices between multiple DataFrames and arrays, without the potential for side effects from inadvertent index modification.

Data Indexing and Selection

Slide 0, 1.

Hi, hello and welcome to the section data indexing and selection.

Data indexing and selection means accessing and modifying values in Pandas Series and DataFrame objects. As per the basic nature, a Series object acts in many ways like a one dimensional NumPy array, and in many ways like a standard Python dictionary.

If we keep these two overlapping analogies in mind, it will help us to understand the patterns of data indexing and selection in the Series objects.

Slide 2.

When the Series acts as dictionary:

It provides a mapping from a collection of keys to a collection of values

When the Series acts as one-dimensional array:

It builds on the dictionary-like interface and provides array-style item selection via the same basic mechanisms as NumPy arrays—that is, slices, masking, and fancy indexing.

Slide 3.

Similar to like Series a DataFrame acts in many ways like a two-dimensional or structured array and in other ways like a dictionary of Series structures sharing the same index. These analogies help us to work with the DataFrame objects when we try to index and select the DataFrame values.

When the DataFrame acts as a Dictionary:

The individual Series that make up the columns of the DataFrame can be accessed via dictionary-style indexing of the column name. Equivalently, we can use attribute style access with column names that are strings.

When it acts as a two dimensional dictionary, we can do many familiar array-like observations on the DataFrame itself. For example, we can transpose the full DataFrame to swap rows and columns

Slide 4.

The indexing and selection of Pandas objects can be broadly categorized into two main groups;

Implicit and Explicit.

Slide 5 and 6.

[Slide's– explanation]

Slide 7.

One guiding principle of Python code is that "explicit is better than implicit". The implicit and explicit index notation's have two main differences;

[Slide – explanation]

Slide 8.

When the Pandas objects consist of both integer and non-integer and also explicit integer notation, the indexing, selection and slicing becomes a bit confusing task. Hence Pandas provides loc, iloc and ix indexers to index, select and slice the implicit and explicit indexers separately.

First, the loc attribute allows indexing and slicing that always references the explicit index. The iloc attribute allows indexing and slicing that always references the implicit Python-style index.

Essential Functionalities

Slide 0 and 1.

Hi, hello and welcome to the section essential functionalities used to preprocess the data.

The essential functionalities mean the fundamental mechanics of interacting with the data contained in a Series or DataFrame.

These are the essential functionalities used in Pandas Series or DataFrame objects.

Slide 2.

An important method on pandas objects is reindex, which means to create a new object with the data conformed to a new index.

Pandas provides ‘reindex’ to rearrange the data according to the new index, introducing missing values if any index values were not already present. For ordered data like time series, it may be desirable to do some interpolation or filling of values when reindexing. The method option allows us to do this, using a method such as ffill, which forward-fills the values while 'bfill' fills backward.

Slide 3.

With DataFrame, reindex can alter either the (row) index, columns, or both. When passed only a sequence, it reindexes the rows in the result. The columns can be reindexed with the columns keyword inside the ‘reindex’ method.

Slide 4.

Here is the summary of ‘reindex’ method arguments used in Pandas objects.

Slide 5.

Working with pandas objects indexed by integers is something that often trips up new users due to some differences with indexing semantics on built-in Python data structures like lists and tuples. In this case, pandas could “fall back” on integer indexing, but it’s difficult to do this in general without introducing a new alternative bugs for it. With the integer index there is potential for ambiguity.

Slide 6.

On the other hand, with a non-integer index, there is no potential for ambiguity.

Slide 7.

To keep things consistent, if we have an axis index containing integers, data selection will always be label-oriented. So we can use ‘loc’ for label based indexing and ‘iloc’ for integer based indexing.

Slide 8.

An important pandas feature for some applications is the behavior of arithmetic between objects with different indexes. When you are adding together objects, if any index pairs are not the same, the respective index in the result will be the union of the index pairs.

If the series objects used for arithmetic and data alignment have the same kind of index values with no missing values, then the resulting series has no change in the index values and no NaN values in the data operation.

Slide 9.

If the series objects used for arithmetic and data alignment have the different kind of index values with missing values. The arithmetic and the internal data alignment introduces missing values in the label locations that don’t overlap.

Slide 10.

If the DataFrame objects used for arithmetic and data alignment have the same kind of index values with no missing values, then the resulting DataFrame has no change in the index values and no NaN values in the data operation.

Slide 11.

If the DataFrame objects used for arithmetic and data alignment have the different kind of index values with missing values. The arithmetic and the internal data alignment introduces missing values in the label locations that don’t overlap and also returned DataFrame has index and columns which are the unions of the ones in each DataFrame.

Slide 12.

If we add DataFrame objects with no column or row labels in common, the result will contain all nulls.

Slide 13.

In arithmetic operations between differently indexed objects, if we want to fill with a special value, like 0 or any other value, when an axis label is found in one object but not the other; we can use ‘fill_values’ parameter in the arithmetic methods.

Slide 14.

The arithmetic operation between the Series and DataFrame objects is known as ‘broadcasting’.

Slide 15.

By default, arithmetic between DataFrame and Series matches the index of the Series on the DataFrame’s columns, broadcasting down the rows.

Slide 16.

If we want to instead broadcast over the columns, matching on the rows. We can pass (axis='index' or axis=0) which is the parameter used in the arithmetic methods by default it is set to operate along the rows.

Slide 17.

NumPy ufuncs (element-wise array methods) also work with pandas objects.

Slide 18.

Another frequent operation is applying a function on one-dimensional arrays to each column or row. DataFrame’s ‘apply’ method will do this job exactly.

By default the ‘apply’ method will invoke the function once per each column in frame. The result is a Series having the columns of frame as its index.

Slide 19.

If we pass axis='columns' to apply, the function will be invoked once per each row instead of column.

Slide 20.

The function passed to apply need not to return a scalar value; it can also return a Series with multiple values.

Slide 21.

We can also use Python functions which are used to perform the operation element-wise instead of either along the row or column axis.

Slide 22.

Sorting a dataset by some criterion is another important built-in operation. To sort lexicographically by row or column index, we can use the sort_index method, which returns a new, sorted object.

Slide 23.

Be default it returns the sorted object by row index but we can pass ‘axis = 1 or column’ to sort along the column index. The data is sorted in ascending order by

default, but can be sorted in descending order using the parameter ‘ascending = False’, by default it is set to True.

Slide 24.

In order to sort the objects by the values of the column of a DataFrame object or values of the series, we can use ‘sort_values’ method.

When sorting a DataFrame, we can use the data in one or more columns as the sort keys. To do so, we can pass one or more column names to the by option of sort_values method.

Slide 25.

Ranking assigns ranks from one through the number of valid data points in an array. The rank methods for Series and DataFrame are the place to look; by default rank breaks ties by assigning each group the mean rank. Ranks can also be assigned according to the order in which they’re observed in the data using the method option.

Slide 26.

While many Pandas functions (like reindex) require that the labels be unique, it’s not mandatory. Sometimes it is often necessary to have duplicate index labels at different positions in the Series or DataFrame objects. So we can use ‘is_unique’ method to check whether the Series or DataFrame object has the duplicate index labels or not.

Data Handling

Slide 0 and 1.

Hi, hello and welcome to the section data handling used to handle the text or tabular form data.

Accessing data is a necessary first step in data analysis with large and database type files. Pandas provides a number of functions to read the text data or tabular data as a DataFrame objects.

Slide 2

The data may be any one of these types that is data may be in the form of csv, excel, json etc. And the data source may be any one of these, that is local disk, data base or network source like WEB API.

Slide 3.

Pandas provides many functions to read and write different kinds of data either in the form of text or tabular form.

The most widely used and the popular methods are the 'read_csv' and 'read_table' methods though there exist many other methods.

Slide 4.

In order to view or print the raw contents of the file from local source like disk we can use 'type' method or 'cat' methods. Windows users can use type method and linux users can use cat method.

Slide 5.

In order to read the comma delimeted file into a dataframe object we can use read_csv method if we have a variable amount of whitespace we can use read_table instead of read_csv with the specified delimiter as the field of separator.

Slide 6.

The read_csv method offers different and highest number of parameters to read variety of csv files and other text files. Some of them are highlighted here.

Slide 7.

In order to read the few lines or few number of rows we can use nrows parameter and with the very large file we can use chunksize parameter to read the file in smaller pieces.

Slide 8.

To export or write the csv files to a another csv files or text form of files we can use 'to_csv' method provided by the pandas. The to_csv method offers a variety of parameters to control how we can export the csv files into text or another csv files for example we can use vertical bar instead of comma to separate the fields of the csv file or text file.

Slide 9.

In order to write to a csv file pandas will accept or invoke the Python's built in csv module. The Python csv module has csv.reader and csv.writer methods to read and write the csv files respectively.

Slide 10.

Other file handling methods are (read the slide). But we are mainly concentrating on the read_csv and read_table methods as they are widely used among these methods.

Let's see more about these read_csv and read_table methods in the practical session.

Data Preprocessing

Slide 0 ,1

Hi, hello and welcome to the section data preprocessing used to clean and preprocess the data.

As per the research on data analysis and data science, it is found that a significant amount of time is spent on data preparation: loading, cleaning, transforming, and rearranging. Such tasks are often reported to take up 80% or more of an analyst's time.

The major types of data preprocessing are, handling the missing values, data transformation and string manipulation.

Slide 2

Let's have a look at how to handle the missing values.

Slide 3

Missing data occurs commonly in many data analysis applications. One of the goals of pandas is to make working with missing data as painless as possible.

Pandas uses different ways to represent the missing values of different types. For numerical missing values pandas uses floating point representation NaN stands for Not a Number. This is also referred as the sentinel value for numeric missing data.

Pandas provides four important methods to handle the missing values, they are ***(read the slide)***

Slide 4

With DataFrame objects, things are a bit more complex. We may want to drop rows or columns that are all NA or only those containing any NAs. dropna by default drops any row containing a missing value.

Passing how='all' will only drop rows that are all NA. passing how = any will drop any row or column having null values. Passing how = all and axis=1 will drop columns that are all null. And if we pass thresh=n parameter with n as an integer value will drop only the n rows with null values.

Slide 5

Rather than filtering out missing data (and potentially discarding other data along with it), we can fill in the "holes" in many number of ways. Pandas provides a fillna method. Calling fillna method with a scalar value will replace the missing values with that scalar value. Calling fillna with a dict, we can use a different fill value for each column. fillna returns a new object, but you can modify the existing object in-place

using inplace=True parameter. We can also use our own standard aggregation methods to fill the missing values using fillna method say for ex: mean or min or max etc. fillna method uses many other parameters to control the missing values as per our requirement.

Slide 6

The data transformation is another class of important operation.

Slide 7

Some times we may find duplicate rows for many reasons. In such case we can use duplicated and drop_duplicates methods of pandas to handle the duplicate rows easily. The duplicated method will duplicated returns a boolean Series indicating whether each row is a duplicate (has been observed in a previous row) or not. drop_duplicates returns a DataFrame where the duplicated array is False. Both of these methods by default consider all of the columns; alternatively, you can specify any subset of them to detect duplicates.

Slide 8

For many datasets, we may wish to perform some transformation based on the values in an array, Series, or column in a DataFrame. The map method on a Series accepts a function or dict-like object containing a mapping elements for each data of the series or column of the dataframe object. Using map is a convenient way to perform element-wise transformations and other data cleaning related operations.

Slide 9

Filling in missing data with the fillna method is a special case of more general value replacement. Pandas provides a replace method which replaces one or more elements in the series or column of a dataframe objects with the specified value or values.

Slide 10

Similar to like values in series or dataframe object we can easily replace or rename the axis labels of series or dataframe object. The axis indexes also have a map method similar to like series to transform one form of axis labels to another form. Suppose if we want to create a transformed version of a dataset without modifying the original, we can use rename method. For ex: we can change the axis labels to a title type or all the characters of axis labels to an upper case type.

Slide 11

Continuous data is often discretized or otherwise separated into “bins” for analysis. For this, we can use cut and qcut methods of pandas. In cut method the data values are grouped based on the discrete values or bins. We can also pass our own bin

names by passing a list or array to the labels parameter of the cut method. On the other hand a closely related function, qcut, bins the data based on sample quantiles. Since qcut uses sample quantiles instead of discrete bins, by default we will obtain roughly equal-size bins.

Slide 12

Filtering or transforming outliers is largely a matter of applying array operations. We can use Boolean type dataframe to filter the dataframe object with some specific conditions.

Slide 13

Sometimes it is necessary to randomly select the rows or columns of a dataframe object. We can do so by randomly reordering a series or rows in the dataframe. This is reffered as permuting or random sampling. Calling the NumPy's permutation method with the length of the axis we want to permute produces an array of integers indicating the new ordering. To select a random subset without replacement, we can use the sample method on Series and DataFrame.

Slide 14

Another very important transformation for statistical modeling or machine learning applications is converting a categorical variable into a "dummy" or "indicator" matrix. If a column in a DataFrame has k distinct values, we would derive a matrix or DataFrame with k columns containing all 1's and 0's. pandas has a get_dummies function to do this operation. For statistical applications we can combine get_dummies with a discretization function like cut or qcut.

Slide 15

We know that Python is a popular raw data manipulation language in part due to its ease of use for string and text processing. Most text operations are made simple with the string object's built-in methods. For more complex pattern matching and text manipulations, regular expressions may be needed. pandas enables us to apply string and regular expressions concisely on whole arrays of data, additionally we can handle the annoying missing data. For example we can split the comma separated string using split method and with the help of join method we can easily join the split words using join method of Python.

Slide 16

Here is the list of some important string manipulation methods.

Slide 17

Regular expressions on the other hand provide a flexible way to search or match string patterns in text. Python's built-in re module is responsible for applying regular

expressions to strings. The re module functions fall into three categories: pattern matching, substitution and splitting.

Slide 18

Here is a list of some important methods used with the regular expressions.

Slide 19

We can also use these regular expression methods on the vectorized string operations.

Data Wrangling

Slide 0, 1

Hi, hello and welcome to the section data wrangling used to rearrange or reshape the data.

Data wrangling is the important step used to rearrange or reshape the data that is not already in usable form. Data wrangling can be broadly categorized into three important types: join, combine and reshape or rearrange.

Slide 2

Let's begin with the hierarchical indexing. The Hierarchical indexing is an important feature of pandas that enables us to have multiple (two or more) index levels on an axis. It provides a way for us to work with higher dimensional data in a lower dimensional form. It is also referred as the MultiIndex in general. Each multi-indexed object can be partially indexed with its label name. Hierarchical indexing plays an important role in reshaping data and group-based operations like forming a pivot table. In dataframe we can have hierarchical index on either of its axis.

Slide 3

In some specific times we will need to rearrange the order of the levels on an axis or sort the data by the values in one specific level. The swaplevel takes two level numbers or names and returns a new object with the levels interchanged. sort_index, on the other hand, sorts the data using only the values in a single level.

Slide 4

Many descriptive and summary statistics methods on DataFrame and Series accepts level to aggregate on a particular axis. For example we compute the sum of the dataframe objects using the level of the row index or column index.

Slide 5

We know that one or more of the dataframe columns can be used to form a pandas series. The set_index method will create a new hierarchically indexed object for the dataframe using one or more its columns as the index. reset_index, on the other hand, does the opposite of set_index and the hierarchically indexed levels will be moved into the columns.

Slide 6 (same as slide 5)

Slide 7

Data contained in pandas objects can be combined together in a number of ways.

Slide 8

pandas.merge connects rows in DataFrames based on one or more keys. This will be familiar to users of SQL or other relational databases, as it implements database join operations.

pandas.concat concatenates or “stacks” together objects along an axis.

The combine_first instance method enables splicing together overlapping data to fill in missing values in one object with values from another.

Slide 9

Merge or join operations combine datasets by linking rows using one or more keys. merge uses the overlapping column names as the keys by default. By default merge does an 'inner' join; the keys in the result are the intersection, or the common set found in both tables. Other possible options are 'left', 'right', and 'outer. The outer join takes the union of the keys, combining the effect of applying both left and right joins.

Slide 10

All the merge operations are broadly categorized into two main categories:

Many to one merge operation

and

Many to many merge operation.

Slide 11

If we find the merge index of the dataframe in its index, still we can merge using left_index=True or right_index=True parameters inside the merge method.

Slide 12

DataFrame has a convenient join instance for merging by index. It can also be used to combine together many DataFrame objects having the same or similar indexes but non-overlapping columns.

Slide 13

Another kind of data combination operation is referred to interchangeably as concatenation, binding, or stacking. Before working with the pandas concat method it is better to know whether it is able to give answers for the following questions.

They are:

If the objects are indexed differently on the other axes, should we combine the distinct elements in these axes or use only the shared values (the intersection)?

Do the concatenated chunks of data need to be identifiable in the resulting object?

Does the “concatenation axis” contain data that needs to be preserved?

The pandas concat method will try to address each of these concerns.

Slide 14

By default concat works along axis=0, producing another Series. If you pass axis=1, the result will instead be a DataFrame (axis=1 is the columns).

Slide 15

There is exist another data combination situation that can’t be expressed as either a merge or concatenation operation. With DataFrames, combine_first method will do the same thing column by column as the NumPy’s where function does.

Slide 16

Let’s see what is reshaping or pivoting the data.

There are a number of basic operations for rearranging tabular data. These are alternatively referred to as reshape or pivot operations.

Slide 17

Hierarchical indexing provides a consistent way to rearrange data in a DataFrame.

There are two primary actions:

stack

This “rotates” or pivots from the columns in the data to the rows

unstack

This pivots from the rows into the columns

Slide 18

Data Aggregation

Slide 0, 1

Hi, hello and welcome to the section groupby and data aggregation used to group and aggregate the data.

After loading, merging, and preparing a dataset, we may need to compute group statistics or possibly pivot tables for reporting or visualization purposes.

So categorizing a dataset and applying a function to each group, whether an aggregation or transformation is referred as groupby and aggregation operation.

Slide 2

The pandas groupby operation mechanism is based on the split-apply-combine formula.

Slide 3

Let's see how it works?

In the first stage of the process, data contained in a pandas object, whether a Series, DataFrame, or otherwise, is split into groups based on one or more keys that you provide. The splitting is performed on a particular axis of an object. That means, a DataFrame can be grouped on its rows (axis=0) or its columns (axis=1). Once this is done, a function is applied to each group, producing a new value. Finally, the results of all those function applications are combined into a result object.

Slide 4

The GroupBy object supports iteration, generating a sequence of 2-tuples containing the group name along with the chunk of data. To do that Pandas groupby method uses a Python's for loop syntax.

Slide 5

We can perform the groupby operation using any of the following ways:

That is by selecting a column or subset of columns of a dataframe object to groupby the dataframe based on the columns used. We can pass a dictionary as a key to groupby object. We can pass a list of elements with the length equal to the length of the dataframe row index labels to the groupby object to group the data. We can also pass a Pythons's functions as a group key for groupby operation. Or even we can use index level to group in the groupby operation.

Slide 6

Aggregations refer to any data transformation that produces scalar values from arrays. Pandas provides an aggregation or agg stands for aggregation to aggregate the data based on the grouped object.

Slide 7

These are some of the commonly used aggregation functions used to aggregate the data. We can also use our own aggregation functions if we don't find the aggregation functions in the built in aggregation functions list.

Slide 8

How to aggregate when the data file too large?

That is possible with the below steps. That's first select the particular columns using the groupby method. Then use the indexing on grouped object with the specific column again. Finally pass a single or list of aggregation functions to the aggregation method to aggregate the data.

Time Series Analysis

Slide 0, 1

Hi, hello and welcome, in this section we learn about the time series analysis.

Anything that is observed or measured at many points in time forms a time series.

Time series data is an important form of structured data in many different fields, such as finance, economics, ecology, neuroscience, and physics.

Slide 2

The time series can be marked and referred based on the application we are working with in it. Time series can be classified into four categories:

Time stamps: stands for specific instants in time.

Fixed periods: such as the month January 2020 or the full year 2020.

Intervals of time, indicated by a start and end timestamp. Periods can be thought of as special cases of intervals.

Experiment or elapsed time; each timestamp is a measure of time relative to a particular start time.

Slide 3

In Python the time series has many libraries related to date and time. here I'm highlighting the important libraries. They are:

datetime stores both the date and time down to the microsecond.

timedelta represents the temporal difference between two datetime objects.

Time, it stores time of day as hours, minutes, seconds and microseconds.

Date; it stores calendar data using the Gregorian calendar.

Calendar displays the dates and days of a calendar for a specific month.

And finally the days it displays the number of days in the timedelta object.

Slide 4

We can easily convert the string type of time series objects to a datetime objects using str and strftime methods passing a format specification.

Slide 5

Here is the list of Datetime format specifications (ISO C89 compatible).

Slide 6

Pandas provides to_datetime method to parse many different kinds of date representations. datetime objects also have a number of locale-specific formatting options for systems in other countries or languages. Some of them are listed here.

Slide 7

To handle the some common types of date formats pandas utilizes a third party package called dateutil. It is capable of handling most human intelligible date representations.

Slide 8

Generic time series in pandas are assumed to be irregular; that is, they have no fixed frequency. For many applications this is sufficient. But for some special applications this is not sufficient. pandas has a full suite of standard time series frequencies and tools for resampling, inferring frequencies, and generating fixed-frequency date ranges. pandas.date_range is responsible for generating a DatetimeIndex with an indicated length according to a particular frequency. By default, date_range generates daily timestamps. date_range by default preserves the time of the start or end timestamp. Frequencies in pandas are composed of a base frequency and a multiplier. Base frequencies are typically referred to by a string alias, like 'D' for daily or 'H' for hourly. “Shifting” refers to moving data backward and forward through time. Both Series and DataFrame have a shift method for doing naive shifts forward or backward, leaving the index unmodified.

Slide 9 and slide 10

Here is the list of some Base time series frequencies.

Slide 11

Working with time zones is generally considered as one of the most unpleasant parts of time series manipulation. As a result, many time series users choose to work with time series in coordinated universal time or UTC, which is the successor to Greenwich Mean Time and is the current international standard. In Python, time zone information comes from the third-party package that is pytz library.

Slide 12

Some of the other important topics of time series are periods and period arithmetic, period frequency conversion and converting timestamps to periods and back again to timestamps.

These are just the basic methods used in time series analysis; still we can find many methods to work with it.