4.1 Engagement: Working with Pandas Part 1

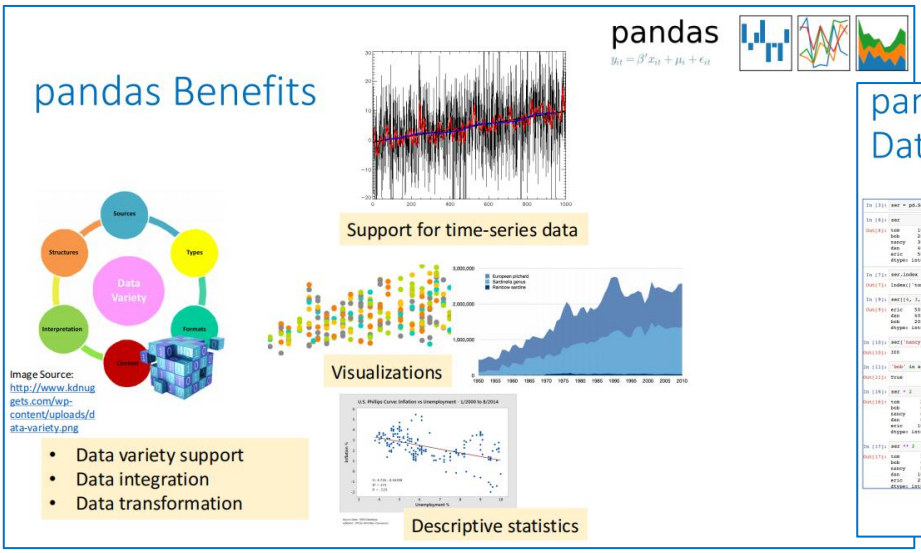
Why pandas

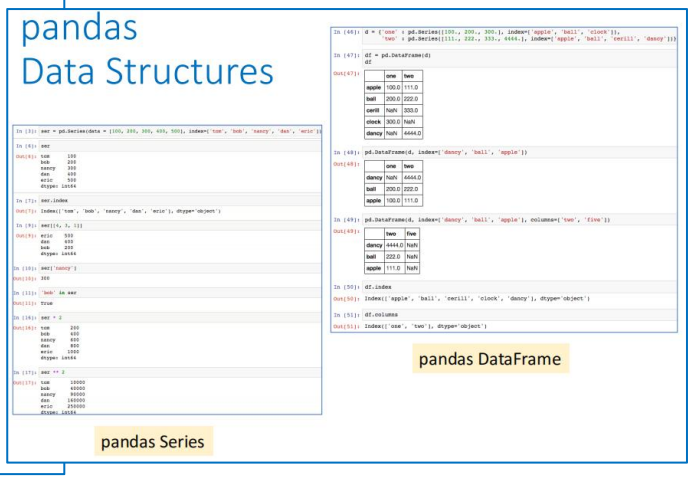
By the end of this video, you should be able to:

- Describe the value of Pandas to data science in Python
- Highlight the key data structures of Pandas
- Discuss the capabilities of Pandas that has resulted in its wide spread adoption as a tool for analytics

In this lesson, we'll be focused on the value of Pandas,the primary data analysis library in Python.

By the end of this video, you should be able to describe the value of pandas to data science in Python, highlight the key data structures of Pandas, and discuss the capabilities of Pandas that has resulted in its widespread adoption as a tool for analytics.





Pandas library provides a numberof data analysis-friendly features, which made it one of the most popular data science tools.

- Pandas builds up NumPy, so most of the NumPy advantages still hold true. However, it uniquely enables ingestion and manipulation of heterogeneous data types in an intuitive fashion. Pandas also enables combining large data sets using merge and join. And it provides a very efficient library for breaking data sets, transforming, and recombining.
- Another great feature Pandas provides is its visualizations.
- Plugged-in data has been simplified in-built functions that come with data frame. And descriptive statistics, by using simple function, is another good part of Pandas. This capability really simplifies the exploratory data analysis, as well as communication of results.
- Additionally, Pandas library handles time-series data effectively via native methods it provides to ingest, transform, and analyze time-series data.
- Other benefits to using Pandas are the ability to take advantage of native methods to handle missing data and data pivoting, easy data sorting, and description capabilities, fast generation of data plots, and Boolean indexing for fast image processing and other masking operations, just to name a few.

Pandas achieves this thanks to two data structures.Namely, pandas Series and pandas DataFrame.

总结:

pandas Series

- A 1-dimensional labeled array
- Supports many data types
- Axis labels → index
  - get and set values by index label
- Valid argument to most NumPy methods

pandas DataFrame

- A 2-dimensional labeled data structure
- A dictionary of Series objects
  - Columns can be of potentially different types
  - Optionally parameters for fine-tuning:
    - index (row labels)
    - columns (column labels)

Pandas provides many constructors to create DataFrames!

A series is one one-dimensional array-like object that provides us with many ways to index data. Series acts like an ndarray, but it supports many data types, integers, strings, floating point numbers, Python objects, et cetera, as a part of the array. It is a valid argument to most NumPy methods because of its similarities to arrays. The axis labels are collectively referred to as the index, and we can get and set values by these index labels. So a series is like a fit sized dictionary in this regard. But it's very flexible.

Although series is a flexible data structure,the data structure that gets used even more is pandas DataFrame. A DataFrame is a 2-D elastic data structure that supports heterogeneous data with labeled axis for rows and columns. Arithmetic operations can appear on both row and column labels. We can think of it as a container for series objects, where each row is a series.

Summary

Pandas supports all steps of DS pipeline

If you're looking for a functionalityto perform some data transformation, chances are Pandas already has it. It provides almost all major data-wrangling capabilities that data scientists need. It is actively supported by developer community and constantly increasing in functionality. We think Pandas will continue to play even a larger role in data science process in the coming decade. We have just reviewed why the Pandas library in Python is very useful and talked about the two data structures in it. Let's get started with our Pandas notebook to review these data structures.

Live Code: Why pandas

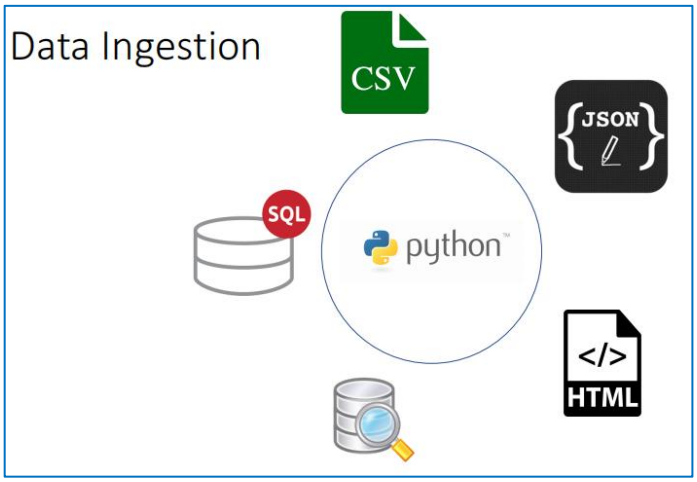
详见 notebook: “Introduction to Pandas.ipynb”

总结:

pandas: Data Ingestion

By the end of this video, you should be able to:

- Describe the efficient and easy to use methods that *pandas* provides for importing data into memory
- Identify functions such as '`read_csv`' for reading a CSV file into a DataFrame
- Discuss about other data sources that *pandas* can directly import from



In this lesson we'll be focusedon importing data into Python. By the end of this video you should be able to describe efficient and easy to use methods that pandas provides for importing data into memory. Identify functions such read dash csv for reading a csv, commas separated values, file into a Dataframe. And discuss about other data resources that pandas can directly import from.

One of the biggest advantages of using pandas isits ability to ingest data from a variety of sources in a variety of data types and formats. We can simply say that pandas has simplified the data ingestion for all of us. Let's look at a few of these data formats and functions that make it possible.

read_csv
- Input : Path to a Comma Separated File
- Output: Pandas DataFrame object containing contents of the file

read_json
- Input : Path to a JSON file or a valid JSON String
- Output: Pandas DataFrame or a Series object containing the contents

read_html
- Input : A URL or a file or a raw HTML String
- Output: A list of Pandas DataFrames

read_sql_query
- Input1 : SQL Query
- Input2 : Database connection
- Output: Pandas DataFrame object containing contents of the file

read_sql_table
- Input1 : Name of SQL table in database
- Input2 : Database connection
- Output : Pandas DataFrame object containing contents of the table

- One of the most popular data formatsis comma separated values, or shortly csv. Csv is a simple file format used to store tabular data such as a spreadsheet or a database. Files in the csv format can be ingested into Python as Dataframes using the pandas read csv function.
- JSON, or java script object notation, is a format for structuring data and it's commonly used for communication within web applications. Using the read JSON function in Python pandas we can ingest the structure and contents of a JSON file as a pandas Dataframe or a series data structure.
- Html, is a hyper text markup language, and it's a file format used as the basis of every webpage. The data in an html document gets stored as a list of pandas DataFrames using the read html function.
- Sequel, or SQL, stands for structured query language. SQL is used to communicate with a database using queries to insert, delete, and select data of interest. The read SQL query function in pandas provides us a way to subset and load data from a relational database to Python.
- Similarly, we can load a whole relational table using the pandas read SQL table function. Then it will simply show in tabular format, as a pandas DataFrame data structure.

Summary

- There are many other methods available in Pandas to ingest data:
  - Google Big Query
  - SAS files
  - Excel tables
  - Clipboard contents
  - Pickle files
  - http://pandas.pydata.org/pandas-docs/stable/api.html#input-output

As a summary, ingestion of datainto Python was not always easy. Pandas made it an intuitive process. And it has enabled data scientists with tools to manipulate the data ingested and key data structures to allow for a vast variety of these data formats.

We listed just a few of the source types that we can ingest into Python. But there are many more examples if you follow the link provided in the summary slide.

Live Code: Data Ingestion

详见 notebook: “Introduction to Pandas.ipynb”

总结:

Pandas: Descriptive Statistics

By the end of this video, you should be able to:

- Describe the capabilities of Pandas for performing statistical analysis on data
- Leverage frequently used functions such as describe()
- Explore other statistical functions in Pandas, which is constantly evolving

In this lecture, we'll be focused onsome of the useful functions in Pandas to generate descriptive data statistics.

By the end of this video, you should recognize the value of Pandas to data science and Python. Describe the capabilities of Pandas for performing statistical analysis on data, and leverage frequently used functions such as describe. We'll also explore other statistical functions in Pandas which is constantly evolving.

describe()

- Syntax : data_frame.describe()
- Output: Shows summary statistics of the dataframe

```
ratings['rating'].describe()

count    2.000026e+07
mean     3.525529e+00
std      1.051989e+00
min      5.000000e-01
25%      3.000000e+00
50%      3.500000e+00
75%      4.000000e+00
max      5.000000e+00
Name: rating, dtype: float64
```

corr()

- Syntax: data_frame.corr()
- Computes pairwise Pearson coefficient (ρ) of columns
- Other coefficients available: Kendall, Spearman

Covariance

$$\rho_{X,Y} = \frac{\mathrm{cov}(X,Y)}{\sigma_X \sigma_Y}$$

Standard deviation

Summary statistics are quantities that capturevarious characteristics of a data set and the values in it, with a single number or small set of numbers. Some basic summary statistics that you should compute for your data set are mean and standard deviation. Pandas does this automatically through the describe function. Looking at these measures will give you an idea of the nature of your data and they can tell you if there's something wrong with your data. For example, min and max values are out of the range of, that are out of the range of zero to five can point to a poor data set in our ratings database.

Correlation or 'corr', C O R R function for computingPearson coefficient, can be used to explore the dependencies between different variables and the data. Some other correlation coefficients are available, like Kendall and Spearman correlations and Pandas can offer support for those as well. As a side note, a negative correlation score means if X becomes larger, then Y becomes smaller. And positive correlation means that the two variables are correlated. We will use the corr() function as it is and wait until the next class on statistics to explore correlation measurements further.

func = min(), max(), mode(), median()

- The general syntax for calling these functions is
  - data_frame.func()
  - Frequently used optional parameter:
    - axis = 0 (rows) or 1 (columns)

mean()

- Syntax: data_frame.mean(axis={0 or 1})
  - Axis = 0 : Index
  - Axis = 1 : Columns
- Output: Series or DataFrame with the mean values

std()

- Syntax: data_frame.std(axis={0 or 1})
  - Axis = 0 : Index
  - Axis = 1 : Columns
- Output: Series or DataFrame with the Standard Deviation values
  - Normalized by N-1

Pandas also offers a number of statistical functionsyou can perform over the whole data frame, a part of the data frame, or individual columns. We refer to all these functions as 'func' on the slide, or F U N C. Just replace your favorite statistical operation with it, like max, min, mode, and median, and you'll find that function in Pandas.

In this slide, we provide you with some basic informationto refer to on inputs and outputs of mean and standard deviation, in addition to the ones we mentioned before.

总结:

any()

- Output: Returns whether ANY element is True
- Benefits:
  - Can detect if a cell matches a condition very quickly

all()

- Output: Returns whether ALL element is True
- Benefits:
  - Can detect if a column or row matches a condition very quickly

Summary

- Some other functions that are worth exploring:
  - Count()
  - Clip()
  - Rank()
  - Round()
  - http://pandas.pydata.org/pandas-docs/stable/api.html#api-dataframe-stats

Pandas also provides capabilities for checkinga condition over a whole data frame or columns with it. Any and all functions, when applied to the resulting objects from the comparison, respectively tell us if any of the comparison results are true or if all of them are true.

As a summary, Pandas provides anextensive array of functions for performing statistical analysis, although we scratched the surface by reliving a few here, I recommend that you spend some time exploring other functions given at the link we provide on this slide. Now let's spend some time in our notebook to review what we discussed.

Live Code: Descriptive Statistics

详见 notebook: “Introduction to Pandas.ipynb”

总结:

4.2 Engagement: Working with Pandas Part 2

pandas: Data Cleaning

By the end of this video, you should be able to:

- Explain why there is need to clean data
- Describe data cleaning as an activity
- Leverage key methods pandas provides for data cleaning

We will now reviewsome of the important and most used data cleaning functions in Pandas.

By the end of this video you should be able to explain why there's a need to clean data, describe data cleaning as an activity, and leverage key methods Pandas provides for data cleaning.

Real-world data is messy!

- Missing values
- Outliers in the data
- Invalid data (e.g. negative values for age)
- NaN value (np.nan)
- None value

Handling Data Quality Issues

- Replace the value
- Fill gaps forward / backward
- Drop fields
- Interpolation

As we mentioned before during our data science overview,real-world data is messy. It can have problems related to missing values, outliers in the data, an invalid data, for instance, negative values for age, and in Python we can also have records within DataFrames as NaN or none values. Since we get the data downstream, we usually have no or little control over how the data is collected. So, we have the data we get and we have to address quality issues by detecting and correcting them. All of these are related to making the data really ready for analysis in the end.

So, how do we clean?Here are some approaches in Pandas that we can take advantage of to address these data quality issues. We can replace invalid or NaN values with more appropriate values. For invalid values again or g aps we can also try to fill in the data instead of removing them. Here, a best estimate for a reasonable value can be used as a replacement. There can be different techniques to find the best estimate and often domain knowledge is required to understand what would be that best estimate. For example, for a missing age value of an employee, a reasonable value can be estimated based on the employee's length of employment. An interpolation of the data values can also be applied to generate estimations of those missing values. Based on how the exploratory and statistical analysis of the dataset goes, we can also think of dropping some of the fields and values that are not important to the task. Outliers, for example, might be dropped depending on the situation.

df.replace()

	0	1
0	-0.349596	-2.017159
1	9999.000000	9999.000000
2	9999.000000	9999.000000
3	0.113889	0.616122
4	0.014707	-1.731660
5	9999.000000	9999.000000
6	1.233087	0.720138
7	9999.000000	9999.000000
8	9999.000000	9999.000000
9	9999.000000	9999.000000

9999.0000

```
df=df.replace(9999.0, 0)
df
```

	0	1
0	-0.349596	-2.017159
1	0.000000	0.000000
2	0.000000	0.000000
3	0.113889	0.616122
4	0.014707	-1.731660
5	0.000000	0.000000
6	1.233087	0.720138
7	0.000000	0.000000
8	0.000000	0.000000
9	0.000000	0.000000

0.0000

Fill missing data gaps forward and backward

df

	0	1
0	0.061038	1.339673
1	NaN	NaN
2	1.578293	0.637435
3	NaN	NaN
4	NaN	NaN
5	NaN	NaN
6	NaN	NaN
7	NaN	NaN
8	NaN	NaN
9	-1.145787	0.052887

```
df.fillna(method='ffill')
```

	0	1
0	0.061038	1.339673
1	0.061038	1.339673
2	1.578293	0.637435
3	1.578293	0.637435
4	1.578293	0.637435
5	1.578293	0.637435
6	1.578293	0.637435
7	1.578293	0.637435
8	1.578293	0.637435
9	-1.145787	0.052887

```
df.fillna(method='backfill')
```

	0	1
0	0.061038	1.339673
1	1.578293	0.637435
2	1.578293	0.637435
3	-1.145787	0.052887
4	-1.145787	0.052887
5	-1.145787	0.052887
6	-1.145787	0.052887
7	-1.145787	0.052887
8	-1.145787	0.052887
9	-1.145787	0.052887

http://pandas.pydata.org/pandas-docs/stable/missing_data.html

总结:

Let's quickly look at some of the functions in Pandasbefore we review them in the Notebook. Using the replace function we can globally change values in a DataFrame. The screenshot shown here shows how we can replace every 9999 with a zero instead.

Fillna method will replace missing values with the last known value forward and backward, meaning going up and going down in the column. If you look at the values in the first DataFrame shown here, the row index one, and column one, you will notice that the forward fill replaces the NaN value in that column with the value of row zero, column one in the forward fill case. If you look at the second DataFrame, row one, column zero will no longer be NaN, it's gonna be the value stored in row zero, column zero. In the backward fill, the same row gets replaced by the value in row two, column one. So, it's going upwards in the DataFrame.

Drop fields using dropna()

df

	0	1	2
0	NaN	NaN	-0.335410
1	NaN	NaN	0.685743
2	0.077005	0.085073	0.565144
3	0.394961	-1.829587	0.494039
4	1.486227	-0.480726	-0.127278
5	NaN	NaN	-0.047668
6	NaN	NaN	-1.504804
7	-0.530518	-0.881817	-2.687352
8	0.825376	1.042468	-0.311527
9	0.097617	1.373572	-0.682435

df.dropna(axis=0)

	0	1	2
2	0.077005	0.085073	0.565144
3	0.394961	-1.829587	0.494039
4	1.486227	-0.480726	-0.127278
7	-0.530518	-0.881817	-2.687352
8	0.825376	1.042468	-0.311527
9	0.097617	1.373572	-0.682435

df.dropna(axis=1)

	2
0	-0.335410
1	0.685743
2	0.565144
3	0.494039
4	-0.127278
5	-0.047668
6	-1.504804
7	-2.687352
8	-0.311527
9	-0.682435

Drop fields using dropna() – axis=0

df

	0	1	2
0	NaN	NaN	-0.335410
1	NaN	NaN	0.685743
2	0.077005	0.085073	0.565144
3	0.394961	-1.829587	0.494039
4	1.486227	-0.480726	-0.127278
5	NaN	NaN	-0.047668
6	NaN	NaN	-1.504804
7	-0.530518	-0.881817	-2.687352
8	0.825376	1.042468	-0.311527
9	0.097617	1.373572	-0.682435

df.dropna(axis=0)

	0	1	2
2	0.077005	0.085073	0.565144
3	0.394961	-1.829587	0.494039
4	1.486227	-0.480726	-0.127278
7	-0.530518	-0.881817	-2.687352
8	0.825376	1.042468	-0.311527
9	0.097617	1.373572	-0.682435

Drop fields using dropna() -- axis=1

df

	0	1	2
0	NaN	NaN	-0.335410
1	NaN	NaN	0.685743
2	0.077005	0.085073	0.565144
3	0.394961	-1.829587	0.494039
4	1.486227	-0.480726	-0.127278
5	NaN	NaN	-0.047668
6	NaN	NaN	-1.504804
7	-0.530518	-0.881817	-2.687352
8	0.825376	1.042468	-0.311527
9	0.097617	1.373572	-0.682435

df.dropna(axis=1)

	2
0	-0.335410
1	0.685743
2	0.565144
3	0.494039
4	-0.127278
5	-0.047668
6	-1.504804
7	-2.687352
8	-0.311527
9	-0.682435

Another function that we'll use a lotis called dropna to drop the fields with missing values. So, dropna function will drop any row or column with a missing value in the DataFrame. With the axis zero option, which is also the default for dropna, any rows with missing values will be eliminated or it's going to be taken out of the DataFrame. So, if you look at here, row zero, one, five and six are no longer in the DataFrame after executing the dropna function with axis zero. With the axis one option, any columns with missing values will be eliminated. In this case, suppose column zero and one has NaN values in them, we are left with only column two in the resulting DataFrame.

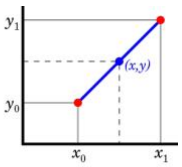
Perform linear interpolation

df

	0	1	2
0	-0.260156	-1.666998	-0.492616
1	NaN	NaN	0.396817
2	-1.263953	-0.562550	-1.670459
3	-0.765183	-0.619429	0.316981
4	-0.165719	-0.678431	0.485722
5	-1.243191	0.494006	0.145171
6	0.373786	-0.769120	-0.929956
7	NaN	NaN	-2.251816
8	-0.536697	1.228345	-1.040728
9	0.254220	-0.021794	1.268333

df.interpolate()

	0	1	2
0	-0.260156	-1.666998	-0.492616
1	-0.762055	-1.114774	0.396817
2	-1.263953	-0.562550	-1.670459
3	-0.765183	-0.619429	0.316981
4	-0.165719	-0.678431	0.485722
5	-1.243191	0.494006	0.145171
6	0.373786	-0.769120	-0.929956
7	-0.081455	0.229613	-2.251816
8	-0.536697	1.228345	-1.040728
9	0.254220	-0.021794	1.268333



Summary

- There are many other ways to transform missing data:
  - Using 'polynomial' interpolation
  - Using Regular Expressions for replacement
- More : http://pandas.pydata.org/pandas-docs/version/0.15.2/missing_data.html#numeric-replacement

You can also interpolate valuesin both series and DataFrame objects. The default for interpolate function is a linear interpolation, meaning the method tries to fit the values to occur over line using linear polynomials, like the point XY in the graphic on this slide. There other interpolation methods to pick from as well but we'll leave it at linear interpolation in this introduction class.

As a summary,there are many ways to deal with missing data and Pandas provides many easy ways to tackle them. We only scratch the surface here but please follow the links provided in this video for a starting point on further methods.

Live Code: Data Cleaning

详见 notebook: "Introduction to Pandas.ipynb"

总结:

Pandas: Data Visualization

By the end of this video, you should be able to:

- Identify key plotting functions of Pandas
- Recognize the ease of utilization of native Pandas methods (for e.g. with DataFrames)

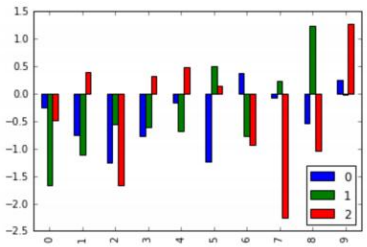
Now we will review the plotting functions in pandas.Note that we will come back to more advanced plotting, in pandas, over the next couple of weeks, after our in-depth data visualization lesson. So this is only a simple introduction.

By the end of this video you should be able to identify key plotting functions of pandas, and recognize the ease of utilization of native pandas methods with data frames.

df.plot.bar()

df

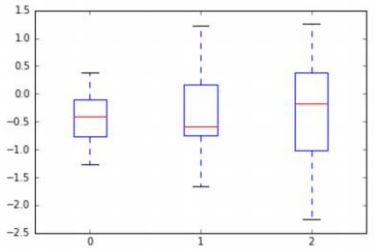
	0	1	2
0	-0.260156	-1.666998	-0.492616
1	-0.762055	-1.114774	0.396817
2	-1.263953	-0.562550	-1.670459
3	-0.765183	-0.619429	0.316981
4	-0.165719	-0.678431	0.485722
5	-1.243191	0.494006	0.145171
6	0.373786	-0.769120	-0.929956
7	-0.081455	0.229613	-2.251816
8	-0.536697	1.228345	-1.040728
9	0.254220	-0.021794	1.268333



df.plot.box()

df

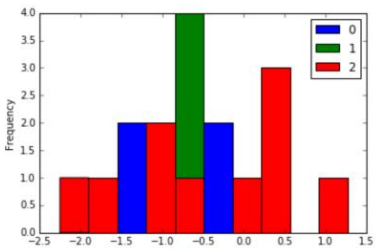
	0	1	2
0	-0.260156	-1.666998	-0.492616
1	-0.762055	-1.114774	0.396817
2	-1.263953	-0.562550	-1.670459
3	-0.765183	-0.619429	0.316981
4	-0.165719	-0.678431	0.485722
5	-1.243191	0.494006	0.145171
6	0.373786	-0.769120	-0.929956
7	-0.081455	0.229613	-2.251816
8	-0.536697	1.228345	-1.040728
9	0.254220	-0.021794	1.268333



df.plot.hist()

df

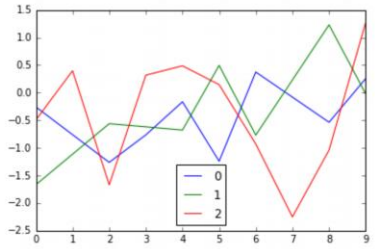
	0	1	2
0	-0.260156	-1.666998	-0.492616
1	-0.762055	-1.114774	0.396817
2	-1.263953	-0.562550	-1.670459
3	-0.765183	-0.619429	0.316981
4	-0.165719	-0.678431	0.485722
5	-1.243191	0.494006	0.145171
6	0.373786	-0.769120	-0.929956
7	-0.081455	0.229613	-2.251816
8	-0.536697	1.228345	-1.040728
9	0.254220	-0.021794	1.268333



df.plot()

df

	0	1	2
0	-0.260156	-1.666998	-0.492616
1	-0.762055	-1.114774	0.396817
2	-1.263953	-0.562550	-1.670459
3	-0.765183	-0.619429	0.316981
4	-0.165719	-0.678431	0.485722
5	-1.243191	0.494006	0.145171
6	0.373786	-0.769120	-0.929956
7	-0.081455	0.229613	-2.251816
8	-0.536697	1.228345	-1.040728
9	0.254220	-0.021794	1.268333



Let us use this data frame as an example to providefor many plot functions of data frames in this video. The plot package offers nice visualizations of bar charts, where each column is represented by a different color, and turned into a bar that goes until the value in that column. Another plot option, box plots, generated by the box function, is a good way of showing data distribution. So each box will have minimum and maximum, and medium for columns, if you look at this graph. Histograms, another type of graph, show the distribution of data, and it can show skewness, or unusual dispersion between data values. Via proper use of the hist function, H-I-S-T, we can generate histograms, not just for one column, but multiple variables in the data, just like we see in this graph. Additionally using the plot function we can create quick line graphs of our data sets. Here we see each column in our data frame represented by a different line, and those points connected by straight lines.

Summary

Explore here: http://pandas.pydata.org/pandas-docs/stable/api.html#api-dataframe-plotting

`DataFrame.plot`([x, y, kind, ax, ...])	DataFrame plotting accessor and method
`DataFrame.plot.area`([x, y])	Area plot
`DataFrame.plot.bar`([x, y])	Vertical bar plot
`DataFrame.plot.barh`([x, y])	Horizontal bar plot
`DataFrame.plot.box`([by])	Boxplot
`DataFrame.plot.density`(**kwds)	Kernel Density Estimate plot
`DataFrame.plot.hexbin`(x, y[, C, ...])	Hexbin plot
`DataFrame.plot.hist`([by, bins])	Histogram
`DataFrame.plot.kde`(**kwds)	Kernel Density Estimate plot
`DataFrame.plot.line`([x, y])	Line plot
`DataFrame.plot.pie`([y])	Pie chart
`DataFrame.plot.scatter`(x, y[, s, c])	Scatter plot
`DataFrame.boxplot`([column, by, ax, ...])	Make a box plot from DataFrame col columns or
`DataFrame.hist`(data[, column, by, grid, ...])	Draw histogram of the DataFrame's

In summary, pandas provides a diverse setof colorable methods for plotting. And they're gonna come a lot on our example notebooks, as we go through this class. It's fun to explore these plots and visually look at your data in different forms. Sometimes you will start seeing things that you don't normally see in the data set, by looking at the data. Then you visualize that you'll be able to see a lot more. So we recommend following the link here, and spending some more time on these functions, and exploring and having fun with these plots.

Live Code: Data Visualization

详见 notebook: “Introduction to Pandas.ipynb”

总结:

pandas: Frequent Data Operations

By the end of this video, you should be able to:

- Handpick data (rows or columns) in a DataFrame using Pandas methods
- Add/ Delete rows or columns in a DataFrame
- Perform aggregation operations / group by

In this videowe'll review the most frequent data operations for subsetting, filtering, insertion, deletion, and aggregation of data. Having such efficient data operations speeds up all algorithms that use these operations and they are used a lot. Let's figure out how to leverage these in Pandas.

By the end of this video you should be able to handpick data, both rows or columns, in a DataFrame using Pandas methods, add or delete rows or columns in a DataFrame and perform aggregation operations like groupby in DataFrames.

```
df
```

	sensor1	sensor2	sensor3
0	-0.260156	-1.666998	-0.492616
1	-0.762055	-1.114774	0.396817
2	-1.263953	-0.562550	-1.670459
3	-0.765183	-0.619429	0.316981
4	-0.165719	-0.678431	0.485722
5	-1.243191	0.494006	0.145171
6	0.373786	-0.769120	-0.929956
7	-0.081455	0.229613	-2.251816
8	-0.536697	1.228345	-1.040728
9	0.254220	-0.021794	1.268333

Slice Out Columns

```
df['sensor1']
```

```
0   -0.260156
1   -0.762055
2   -1.263953
3   -0.765183
4   -0.165719
5   -1.243191
6    0.373786
7   -0.081455
8   -0.536697
9    0.254220
Name: sensor1, dtype: float64
```

Filter Out Rows

```
#Select rows where sensor2 is positive
df[df['sensor2'] > 0]
```

	sensor1	sensor2	sensor3
5	-1.243191	0.494006	0.145171
7	-0.081455	0.229613	-2.251816
8	-0.536697	1.228345	-1.040728

In all of the coming slideswe will use a toy DataFrame that is present in each slide with the name df. This is the first DataFrame that we see in this slide. You can slice out the column of your choice simply by providing its name, like we see here for sensor. We've reviewed this one earlier as well.
Filtering out rows based on a condition is a very common task. Here, we do df sensor greater than zero to find out all the rows that have sensor2 in sensor2 that are greater than zero.

Insert New Columns

```
df['sensor4']=df['sensor3']**2
```

```
df
```

	sensor1	sensor2	sensor3	sensor4
0	-0.260156	-1.666998	-0.492616	0.242671
1	-0.762055	-1.114774	0.396817	0.157464
2	-1.263953	-0.562550	-1.670459	2.790434
3	-0.765183	-0.619429	0.316981	0.100477
4	-0.165719	-0.678431	0.485722	0.235925
5	-1.243191	0.494006	0.145171	0.021075
6	0.373786	-0.769120	-0.929956	0.864819
7	-0.081455	0.229613	-2.251816	5.070676
8	-0.536697	1.228345	-1.040728	1.083115
9	0.254220	-0.021794	1.268333	1.608669

Add a New Row

```
df.loc[10] = [11,22,33,44]
```

```
df
```

	sensor1	sensor2	sensor3	sensor4
0	-0.260156	-1.666998	-0.492616	0.242671
1	-0.762055	-1.114774	0.396817	0.157464
2	-1.263953	-0.562550	-1.670459	2.790434
3	-0.765183	-0.619429	0.316981	0.100477
4	-0.165719	-0.678431	0.485722	0.235925
5	-1.243191	0.494006	0.145171	0.021075
6	0.373786	-0.769120	-0.929956	0.864819
7	-0.081455	0.229613	-2.251816	5.070676
8	-0.536697	1.228345	-1.040728	1.083115
9	0.254220	-0.021794	1.268333	1.608669
10	11.000000	22.000000	33.000000	44.000000

Adding a new columnis performed by using the name of the new column you want on the left-hand side and providing the value on the right-hand side. In this example, we create a new column called sensor4 by squaring the values of column and sensor3. You can provide data on right-hand side in a variety of formats. I would encourage you to try adding any column using values in a list, array and see how this works for yourself.

Here,in this slide, we show you how you can leverage the .loc or location function to specify exactly which row you want to add to your new data. Notice, the right-hand side is a list of values containing exactly the same number of values as the number of columns.

总结:

Delete a Row

```
df.drop(df.index[[5]])
```

	sensor1	sensor2	sensor3	sensor4
0	-0.260156	-1.666998	-0.492616	0.242671
1	-0.762055	-1.114774	0.396817	0.157464
2	-1.263953	-0.562550	-1.670459	2.790434
3	-0.765183	-0.619429	0.316981	0.100477
4	-0.165719	-0.678431	0.485722	0.235925
6	0.373786	-0.769120	-0.929956	0.864819
7	-0.081455	0.229613	-2.251816	5.070676
8	-0.536697	1.228345	-1.040728	1.083115
9	0.254220	-0.021794	1.268333	1.608669

Delete a Column

```
del df['sensor1']
```

df

	sensor2	sensor3	sensor4
0	-1.666998	-0.492616	0.242671
1	-1.114774	0.396817	0.157464
2	-0.562550	-1.670459	2.790434
3	-0.619429	0.316981	0.100477
4	-0.678431	0.485722	0.235925
5	0.494006	0.145171	0.021075
6	-0.769120	-0.929956	0.864819
7	0.229613	-2.251816	5.070676
8	1.228345	-1.040728	1.083115
9	-0.021794	1.268333	1.608669

Group By and Aggregate

df

	student_id	physics	chemistry	biology
0	2	198	92	108
1	2	111	134	122
2	2	37	174	25
3	4	121	128	63
4	4	191	97	178
5	4	102	157	182
6	12	70	76	181
7	12	101	62	128
8	12	26	51	56
9	100	148	78	159

```
df.groupby('student_id').mean()
```

	physics	chemistry	biology
student_id			
2	115.333333	133.333333	85.000000
4	138.000000	127.333333	141.000000
12	65.666667	63.000000	121.666667
100	148.000000	78.000000	159.000000

Drop functions, as you've seen before,let's you delete a row from a DataFrame. You can use df.index to specify one or more rows to drop. Notice how the right DataFrame on the right is smaller and has the fifth row missing.

As we've seen before,del function intuitively lets you delete a column simply using its name. Sometimes there are columns that are not relevant to your specific analysis like we had for the timestamps in our ratings database, so that data is the data you don't want to consider in your analysis. You can simply get rid of it using the del function.

Now, groupby is a very useful methodthat let's you get combined statistics about the DataFrame. Here we show you how to perform groupby using a student ID and extract mean scores for each subject. So, if the student took the same subject more than a couple of times, we can group them by and take the average of the performance for the student two. If you look at two, it shows up now as average score for that student ID two for each subject.

In summary, Pandas has a vast array of efficient methods to allow you to play with your dataset. We have only discussed a short subset here but we encourage you to explore at the link here to find out new ways. You will soon realize with some experience that connecting these simple operations can result in large and complex pipelines of analytics that transform raw data into something that we can analyze and get meaningful insights from.

Summary

We saw a subset of transformation, more to explore here :

http://pandas.pydata.org/pandas-docs/stable/api.html

Live Code: Frequent Data Operations

详见 notebook: “Introduction to Pandas.ipynb”

总结:

4.3 Engagement: Working with Pandas Part 3

pandas: Merging DataFrames

By the end of this video, you should be able to:

- Explain that data is usually distributed across different locations and tables
- Combine data from distinct DataFrames to obtain the big picture
- Distinguish among different ways to combine data sets

Example Dataframes

left

	_key1	_key2	city	user_name
0	K0	z0	city_0	user_0
1	K1	z1	city_1	user_1
2	K2	z2	city_2	user_2
3	K3	z3	city_3	user_3

left

right

	_key1	_key2	hire_date	profession
0	K0	z0	h_0	p_0
1	K1	z1	h_1	p_1
2	K2	z2	h_2	p_2
3	K3	z3	h_3	p_3

right

Often, when working with dataframes,we need to work with data from multiple frames. A common practice is to merge data we want from two frames into a single frame, and execute operations on the new frame. If you're familiar with database management systems, this is very similar to a join operation. In this video, we will review how to do this in pandas.
By the end of this video, you should be able to explain that data is usually distributed across different locations and tables. Combine data from distinct DataFrames to obtain a big picture. And distinguish among different ways to combine data sets.

For this video we will use the two DataFramesgiven here as examples. The left DataFrame has two key columns, key one and key two, and additional two columns for city and user name. The right DataFrame has the same two key columns, and additional two columns for hire date and profession.

pandas.concat() and .append()

-- Stack DataFrames

```
pd.concat([left, left])
```

	_key1	_key2	city	user_name
0	K0	z0	city_0	user_0
1	K1	z1	city_1	user_1
2	K2	z2	city_2	user_2
3	K3	z3	city_3	user_3
0	K0	z0	city_0	user_0
1	K1	z1	city_1	user_1
2	K2	z2	city_2	user_2
3	K3	z3	city_3	user_3

```
pd.concat([left, right])
```

	_key1	_key2	city	hire_date	profession	user_name
0	K0	z0	city_0	NaN	NaN	user_0
1	K1	z1	city_1	NaN	NaN	user_1
2	K2	z2	city_2	NaN	NaN	user_2
3	K3	z3	city_3	NaN	NaN	user_3
0	K0	z0	NaN	h_0	p_0	NaN
1	K1	z1	NaN	h_1	p_1	NaN
2	K2	z2	NaN	h_2	p_2	NaN
3	K3	z3	NaN	h_3	p_3	NaN

```
left.append(right)
```

	_key1	_key2	city	hire_date	profession	user_name
0	K0	z0	city_0	NaN	NaN	user_0
1	K1	z1	city_1	NaN	NaN	user_1
2	K2	z2	city_2	NaN	NaN	user_2
3	K3	z3	city_3	NaN	NaN	user_3
0	K0	z0	NaN	h_0	p_0	NaN
1	K1	z1	NaN	h_1	p_1	NaN
2	K2	z2	NaN	h_2	p_2	NaN
3	K3	z3	NaN	h_3	p_3	NaN

The concat function in pandas can be usedto stack DataFrames and create a new DataFrame out of them. Here we see the DataFrame called left being concatenated with itself. The index for the resulting DataFrame will have row indexes from the original tables preserved.

If the two DataFrames given to the concat function have columns that are separate, the resulting DataFrame will have the columns from both frames represented. In that case, some of the cells for the columns that didn't exist in the original DataFrames will end up having NaN or missing values as they will be missing in the first DataFrame we merged into this larger DataFrame.

An alternative to concat is append.We can also use the append function to append the DataFrame to any other DataFrame. It behaves similarly to the concat function, but it is a function of the DataFrame itself. So we'll say left append, and then give it another DataFrame. Like our first use of concat, we've got those many empty cells again here.

总结:

Inner join using pandas.concat()

```
pd.concat([left, right], axis=1, join='inner')
```

	_key1	_key2	city	user_name	_key1	_key2	hire_date	profession
0	K0	z0	city_0	user_0	K0	z0	h_0	p_0
1	K1	z1	city_1	user_1	K1	z1	h_1	p_1
2	K2	z2	city_2	user_2	K2	z2	h_2	p_2
3	K3	z3	city_3	user_3	K3	z3	h_3	p_3

Inner join using pandas.merge()

```
pd.merge(left, right, how='inner')
```

	_key1	_key2	city	user_name	hire_date	profession
0	K0	z0	city_0	user_0	h_0	p_0
1	K1	z1	city_1	user_1	h_1	p_1
2	K2	z2	city_2	user_2	h_2	p_2
3	K3	z3	city_3	user_3	h_3	p_3

Instead of having extra rows with missing numbers,we could also try an inner join. Inner join is a useful operation for merging data, as it combines the column values of two DataFrames, into a new DataFrame, just like we see here. Notice the way to do this is to specify to concat that the join type is inner. In the previous slide, the concatenated DataFrames were stacked vertically. Here, they are placed next to each other horizontally, using the indices zero, one, two, and three. In the horizontal stacking unfortunately, this isn't the perfect merge for our data either, as the key columns have been duplicated when they were merged into the new DataFrame separately.

The operation which will give us a true combinationof these two frames is called merge. The benefit of using the merge operation, is that it can eliminate the duplicate columns between the DataFrames it joins. So it's behaving very much like concat using inner join, it just trips out those duplicate columns we had. Although all these methods have utility depending on the situation, I find myself using merge fairly often, as I'm often trying to combine data from multiple different sources, which all share the same keys.

In summary, data in the form of tables benefitsfrom both clean and simple joining and merging operations, similar to relational database operations. Pandas provides support for these database-like operations through it's native DataFrame functions, make it an easier to integrate data from a variety of sources for analytics on the merge datasets.

Summary

More adventure:

http://pandas.pydata.org/pandas-docs/stable/merging.html#database-style-dataframe-joining-merging

Live Code: Merging DataFrames

详见 notebook: “Introduction to Pandas.ipynb”

总结:

pandas: Frequent String Operations

By the end of this video, you should be able to:

- Describe what operations the string methods can perform
- Navigate your way to find the right string method for you
- Perform basic string operations in Pandas

String is a commonly used data typebecause data science often involves studying text data. In this video, we will review just a few of the many useful string operations in Pandas. For those of you who worked through the Python basics a few weeks ago, you'll recognize that many of these methods are quite similar to the methods from Python's strings.

By the end of this video, you should be able to describe what operations the string methods can perform, navigate your way to find the right string method for you, and perform basic string operations in Pandas.

str.split()

df

	_key1	_key2	city	user_name	hire_date	profession
0	K0	z0	city_0	user_0	h_0	p_0
1	K1	z1	city_1	user_1	h_1	p_1
2	K2	z2	city_2	user_2	h_2	p_2
3	K3	z3	city_3	user_3	h_3	p_3

```
df['city'].str.split('_')

0    [city, 0]
1    [city, 1]
2    [city, 2]
3    [city, 3]
dtype: object
```

str.contains()

df

	_key1	_key2	city	user_name	hire_date	profession
0	K0	z0	city_0	user_0	h_0	p_0
1	K1	z1	city_1	user_1	h_1	p_1
2	K2	z2	city_2	user_2	h_2	p_2
3	K3	z3	city_3	user_3	h_3	p_3

```
df['city'].str.contains('2')

0    False
1    False
2     True
3    False
Name: city, dtype: bool
```

str.replace()

df

	_key1	_key2	city	user_name	hire_date	profession
0	K0	z0	city_0	user_0	h_0	p_0
1	K1	z1	city_1	user_1	h_1	p_1
2	K2	z2	city_2	user_2	h_2	p_2
3	K3	z3	city_3	user_3	h_3	p_3

```
df['city'].str.replace('_', '##')

0    city##0
1    city##1
2    city##2
3    city##3
Name: city, dtype: object
```

One of the most important string operations is split.It helps with separating data into pieces around a delimiter character. For example, the city column in the df dataframe here is split around the underscore character, returning an object with arrays of all the pieces for each row in the city column. To be clear, the city field now contains an array of strings rather than just the string. For the split function itself, notice that city_0 has been turned into two strings, city and zero. The same has been applied for each string in the city column.

The function contains, provides a simple way to check if a string has a given character in it. We already used this in our live video coding sessions before, but let's look at this example when we check if any of the values in the city column contains two as a character. We see that only index two has it. So, the result is a boolean series with only one true value for that index two.

Using the replace operation,we can replace a substring with another one. For example, we are replacing the underscore character with two hashtags here.

str.extract()

df

	_key1	_key2	city	user_name
0	K0	z0	city 0	user_0
1	K1	z1	city 1	user_1
2	K2	z2	city 2	user_2
3	K3	z3	city 3	user_3

```
# Extract words in the strings
df['city'].str.extract('([a-z]\w{0,})')

0    city
1    city
2    city
3    city
Name: city, dtype: object
```

```
# Extract single digit in the strings
df['city'].str.extract('(\d)')

0    0
1    1
2    2
3    3
Name: city, dtype: object
```

Summary

Explore more :

http://pandas.pydata.org/pandas-docs/stable/text.html#text-string-methods

The extract function will return the first matchfor a regular expression it finds. The top example here is a regular expression to extract words in a string and a bottom example is how we can extract the strings. In general, extract, can be a quick way to get new features and values. For example, you can use it to build a numeric feature out of text data like in the lower example.

In summary, string operations can be very handy in data cleaning. Please use the link provided here to get started on exploring more string operations to make yourself familiar with the rest of them.

Live Code: Frequent String Operations

详见 notebook: “Introduction to Pandas.ipynb”

总结:

pandas: Parsing Timestamps

By the end of this video, you should be able to:

- Explain what Unix time / POSIX time / epoch time is
- Describe data types for datetime
- Select rows based on time stamps
- Sort tables in chronological order

Unix time / POSIX time / epoch time

- Number of seconds elapsed since
  - 00:00:00
  - Coordinated Universal Time (UTC),
  - Thursday, 1 January 1970
- Prominent in UNIX like systems
- Parsing Timestamp: We have to read POSIX time and understand what the exact time stamp was

Working with timestamps can be difficult dueto a number of different time data formats and resolutions. In this video, we will go over some of the time data formats, and structures and operations.
By the end of it, you should be able to explain what Unix time, POSIX time, and epoch time is, describe data types for datetime, select rows based on timestamps, and sort tables in chronological order.
Unix time tracks the progress of timeby counting the number of seconds since a specific time instant, which is the start of the year 1970 as per UTC time zone. Notice that this is an integer. We need to find a way to convert this to a readable date and time.

Data Types for Timestamps

- **Generic data type:** `datetime64[ns]`
- **Convert** `int64` timestamp to `<M8[ns]` or `>M8[ns]` on your machine

```
tags.dtypes
userId          int64
movieId         int64
tag            object
timestamp       int64
dtype: object
```

→ `dtype('<M8[ns]')`

Convert Timestamp to Python Format

`to_datetime()`

```
tags['parsed_time'] = pd.to_datetime(tags['timestamp'], unit='s')
```

```
tags.head(2)
```

	userId	movieId	tag	timestamp	parsed_time
0	18	4141	Mark Waters	1240597180	2009-04-24 18:19:40
1	65	208	dark hero	1368150078	2013-05-10 01:41:18

Datetime64 [Ns] is a general data type for datetime.This general data type maps to specific data types called M8, or, before M8, or after M8, depending on your machine. Our big task is to convert int64, which was that original instant since 1970 UTC time, into either one of the above datetime formats so Python renders it in a human-readable format.

It's kind of confusing, maybe, seeing all these data types, but we can quickly convert a timestamp to Python format using, for example, to_datetime function here. This function simply will convert the input to datetime format, to a form that Python understands. And in this case, we'll put that into that parsed_time column which you see is pretty human-readable as well. The unit argument is most important here as it tells the function what the unit of the input is. In this example, the input column is timestamp from the dataframe called tags, and the unit of the input is declared to be seconds. The output is stored in a new column that is named parsed_time.

Select Rows Based on Timestamps

```
greater_than_t = tags['parsed_time'] > '2015-02-01'
```

```
selected_rows = tags[greater_than_t]
```

Sort Tables in Chronological Order

```
tags.sort_values(by='parsed_time', ascending=True)[:10]
```

	userId	movieId	tag	timestamp	parsed_time
333932	100371	2788	monty python	1135429210	2005-12-24 13:00:10
333927	100371	1732	coen brothers	1135429236	2005-12-24 13:00:36
333924	100371	1206	stanley kubrick	1135429248	2005-12-24 13:00:48
333923	100371	1193	jack nicholson	1135429371	2005-12-24 13:02:51
333939	100371	5004	peter sellers	1135429399	2005-12-24 13:03:19
333922	100371	47	morgan freema	1135429412	2005-12-24 13:03:32
333921	100371	47	brad pitt	1135429412	2005-12-24 13:03:32
333936	100371	4011	brad pitt	1135429431	2005-12-24 13:03:51
333937	100371	4011	guy ritchie	1135429431	2005-12-24 13:03:51
333920	100371	32	bruce willis	1135429442	2005-12-24 13:04:02

Once time is converted to Python format,you can use it to create filters. The filter will select only the rows that will match your criteria. For example, once we build a boolean filter called greater_than_t, and in this case, make that timestamp to be after 2015, February 1, or 02-01, we can use this filter to select only the cells in the dataframe that are true for this condition. We can also leverage the timestamp to sort data in ascending or descending order. The sort_values function in Panda's dataframes provides number of options to sort, one of which is by parsed time. So we are getting the parsed_time column in date format that we had before, and sorting time series data using the sort values function that's already a function of the dataframe. Sorting time series data like this can help in improved and effective visualizations because then you can provide assorted data to your visualization. So we can say, in essence Panda's ability to sort data based on time stamps can give us a big picture, almost instantly. You can even look at this dataset or dataframe and see the progression over time by looking at, you know, from above, to the lower rows in the data frame.

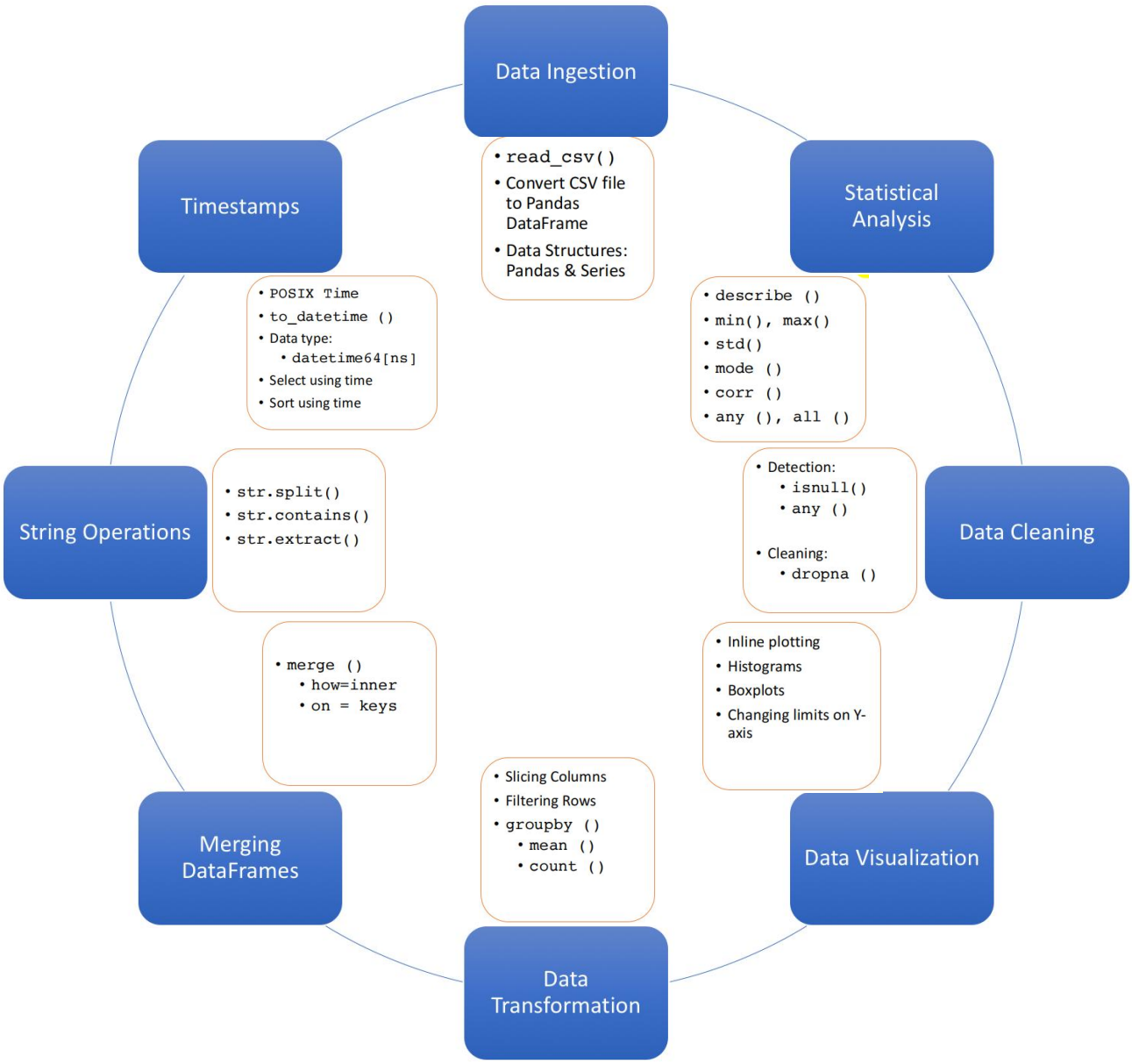
总结：

Summary

- POSIX / Unix time can be hard to read for users
- Converting to Python datetime format gives practical ways to:
  - Select data based on human readable time stamps
  - Create conditions using understandable time stamps

In summary, POSIX, or Unix time,can be hard to read for users because it's a large integer, but converting it to Python datetime format gives us practical ways to select the data and make it human-readable in terms of timestamps, and create conditions on it using understandable time stamps from a human-readability point of view.

pandas: Summary of Movie Rating Notebook



Summary

- A typical data ingestion and transformation cycle
- Movies notebook as a representative example

总结: