TRAINING ON PANDAS

INTRODUCTION

Pandas is a Python package that provides fast, flexible, and expressive data structures designed to make working with "relational" or "labeled" data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real world data analysis in Python. Additionally, it has the broader goal of becoming the most powerful and flexible open source data analysis / manipulation tool available in any language. It is already well on its way towards this goal.

Here are just a few of the things that pandas does well:

- Easy handling of missing data (represented as NaN, NA, or NaT) in floating point as well as non-floating point data
- Size mutability: columns can be inserted and deleted from DataFrame and higher dimensional objects
- Automatic and explicit data alignment: objects can be explicitly aligned to a set of labels, or the user can simply ignore the labels and let Series, DataFrame, etc. automatically align the data for you in computations
- Powerful, flexible group by functionality to perform split-apply-combine operations on data sets, for both aggregating and transforming data
- Make it easy to convert ragged, differently-indexed data in other Python and NumPy data structures into DataFrame objects
- Intelligent label-based slicing, fancy indexing, and subsetting of large data sets
- Intuitive merging and joining data sets

Installing and Importing Pandas

If you already have python installed, than use

- conda install pandas

Or

- pip install pandas

Import Pandas

- Import pandas as pd

Core components of pandas: Series and DataFrames

The primary two components of pandas are the Series and DataFrame.

A Series is essentially a column, and a DataFrame is a multi-dimensional table made up of a collection of Series.

Apple
3
2
0
1

+

Oranges
0
3
7
2

=

Apple	Oranges
3	0
2	3
0	7
1	2

DataFrames and Series are quite similar in that many operations that you can do with one you can do with the other, such as filling in null values and calculating the mean.

Creating DataFrame

There are many ways to create a DataFrame from scratch, but a great option is to just use a simple dictionary.

Let's say we have a fruit stand that sells apples and oranges. We want to have a column for each fruit and a row for each customer purchase. To organize this as a dictionary for pandas we could do something like:

```
data = {
    'apples': [3, 2, 0, 1],
    'oranges': [0, 3, 7, 2]
}
```

And then pass it to the pandas DataFrame constructor:

```
purchases = pd.DataFrame(data)
print(purchases)
```

	apples	oranges
0	3	0
1	2	3
2	0	7
3	1	2

How did that work?

Each (key, value) item in data corresponds to a column in the resulting DataFrame.

The Index of this DataFrame was given to us on creation as the numbers 0-3, but we could also create our own when we initialize the DataFrame.

Let's have customer names as our index:

- purchases = pd.DataFrame(data, index=['June', 'Robert', 'Lily', 'David'])
- print(purchases)

	apples	oranges
June	3	0
Robert	2	3
Lily	0	7
David	1	2

So now we could locate a customer's order by using their name:

- purchases.loc['June']

OUT:

apples 3

oranges 0

Name: June, dtype: int64

Iterating a DataFrame

- iteritems()

Iterates over each column as key, value pair with label as key and column value as a Series object, to iterate over the (key,value) pairs.

- iterrows()

iterrows() returns the iterator yielding each index value along with a series containing the data in each row, iterate over the rows as (index,series) pairs.

- itertuples()

itertuples() method will return an iterator yielding a named tuple for each row in the DataFrame. The first element of the tuple will be the row's corresponding index value, while the remaining values are the row values, iterate over the rows as namedtuples.

Reading data from CSVs

With CSV files all you need is a single line to load in the data:

df = pd.read_excel('filename.xlsx')

df

	Unnamed: 0	apples	oranges
0	June	3	0
1	Robert	2	3
2	Lily	0	7
3	David	1	2

CSVs don't have indexes like our DataFrames, so all we need to do is just

designate the index_col when reading:

df = pd.read_excel(filename.xlsx ', index_col=0)

df

	apples	oranges
June	3	0
Robert	2	3
Lily	0	7
David	1	2

Some Important Functions

1. .head() - It outputs the first five rows of your DataFrame by default, but we could also pass a number as well.
2. .tail() - It outputs the last five rows of your DataFrame by default, but we could also pass a number as well.
3. .info() – It should be one of the very first commands you run after loading your data. It provides the essential details about your dataset, such as the number of rows and columns, the number of non-null values, what type of data is in each column, and how much memory your DataFrame is using.
4. .shape - It has no parentheses and is a simple tuple of format (rows, columns). You'll be going to .shape a lot when cleaning and transforming data. For example, you might filter some rows based on some criteria and then want to know quickly how many rows were removed.
5. drop_duplicates() - method will also return a copy of your DataFrame, but this time with duplicates removed. Another important argument for drop_duplicates() is keep, which has three possible options:
   a) first: (default) Drop duplicates except for the first occurrence.
   b) last: Drop duplicates except for the last occurrence.
   c) False: Drop all duplicates.

Functions & Description

1	count()	Number of non-null observations
2	sum()	Sum of values
3	mean()	Mean of Values
4	median()	Median of Values
5	mode()	Mode of values
6	std()	Standard Deviation of the Values
7	min()	Minimum Value
8	max()	Maximum Value
9	abs()	Absolute Value
10	prod()	Product of Values
11	cumsum()	Cumulative Sum
12	cumprod()	Cumulative Product

Check for Missing Values

To make detecting missing values easier (and across different array dtypes), Pandas provides the isnull() and notnull() functions, which are also methods on Series and DataFrame objects.

```
Import numpy as np

Import pandas as pd


df = pd.DataFrame(np.random.randn(5, 3), index=['a', 'c', 'e', 'f','h'],columns=['one', 'two', 'three'])

df = df.reindex(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'])

print df['one'].isnull()


Print(df.fillna(0))
```

```
a  False
b  True
c  False
d  True
e  False
f  False
g  True
h  False
Name: one, dtype: bool
```

Group By Function

Any groupby operation involves one of the following operations on the original object. They are −

1. Splitting the Object
2. Applying a function
3. Combining the results

In many situations, we split the data into sets and we apply some functionality on each subset. In the apply functionality, we can perform the following operations −

1. Aggregation − computing a summary statistic
2. Transformation − perform some group-specific operation
3. Filtration − discarding the data with some condition

Let us now create a DataFrame object and perform all the operations on it −

Merging/Joining

Pandas provides a single function, merge, as the entry point for all standard database join operations between DataFrame objects −

- pd.merge(left, right, how='inner', on=None, left_on=None, right_on=None, left_index=False, right_index=False, sort=True)

- left − A DataFrame object.
- right − Another DataFrame object.
- on − Columns (names) to join on. Must be found in both the left and right DataFrame objects.
- left_on − Columns from the left DataFrame to use as keys. Can either be column names or arrays with length equal to the length of the DataFrame.
- right_on − Columns from the right DataFrame to use as keys.
- left_index − If True, use the index (row labels) from the left DataFrame as its join key(s).
- right_index − Same usage as left_index for the right DataFrame.
- how − One of 'left', 'right', 'outer', 'inner'. Defaults to inner. Each method has been described below.
- sort − Sort the result DataFrame by the join keys in lexicographical order. Defaults to True, setting to False will improve the performance substantially in many cases.

Merge Using 'how' Argument

The how argument to merge specifies how to determine which keys are to be included in the resulting table.

If a key combination does not appear in either the left or the right tables, the values in the joined table will be NA.

Here is a summary of the how options and their SQL equivalent names −

Merge Method	SQL Equivalent	Description
left	LEFT OUTER JOIN	Use keys from left object
right	RIGHT OUTER JOIN	Use keys from right object
outer	FULL OUTER JOIN	Use union of keys
inner	INNER JOIN	Use intersection of keys

Concatenation

Pandas provides various facilities for easily combining together Series, DataFrame, and Panel objects.

- pd.concat(objs,axis=0,join='outer',join_axes=None, ignore_index=False)

- objs – This is a sequence or mapping of Series, DataFrame, or Panel objects.
- axis – {0, 1, ...}, default 0. This is the axis to concatenate along.
- join – {‘inner’, ‘outer’}, default ‘outer’. How to handle indexes on other axis. Outer for union and inner for intersection.
- ignore_index – boolean, default False. If True, do not use the index values on the concatenation axis. The resulting axis will be labeled 0, ..., n - 1.
- join_axes – This is the list of Index objects. Specific indexes to use for the other (n-1) axes instead of performing inner/outer set logic.

Concatenating Using append -

A useful shortcut to concat are the append instance methods on Series and DataFrame. These methods actually predated concat. They concatenate along axis=0

- df1.append(df2)

Thank You!