What is a tutorial on Pandas DataFrame and what are some examples?

Authored by

stats writer

June 24, 2024

A tutorial on Pandas DataFrame is a comprehensive guide that provides an in-depth understanding of the data structure and its functionalities in the Pandas library. It covers various methods and operations that can be performed on DataFrame objects, such as data manipulation, merging, filtering, and visualization. Examples of topics covered in a Pandas DataFrame tutorial include creating a DataFrame from various data sources, selecting and manipulating data, and performing statistical analysis. It also provides practical examples and exercises to help users better understand and apply the concepts discussed.

1. What is Pandas DataFrame?

A pandas DataFrame represents a two-dimensional dataset, characterized by labeled rows and columns, making it a versatile and immutable tabular structure. It comprises three essential components: the data itself, along with its rows and columns. Built upon the robust foundation of the NumPy library, pandas is implemented using languages such as Python, Cython, and C.

3. DataFrame Features

4. Create Pandas DataFrame

In this section of the tutorial, I will explain different ways to create pandas DataFrame with examples.

4.1 Create using Constructor

One of the easiest ways to create a DataFrame is by using its constructor.

```
# Create pandas DataFrame from List
import pandas as pd
technologies = ,
,
]
df=pd.DataFrame(technologies)
print(df)
```

Since we have not given labels to columns and indexes, DataFrame by default assigns incremental sequence numbers as labels to both rows and columns, these are called Index.

```
# Output
0 1 2
0 Spark 20000 30days
1 pandas 20000 40days
```

Assigning sequence numbers as column names can be confusing, as it becomes challenging to discern the content of each column. Therefore, it's advisable to assign meaningful names to columns that reflect the data they contain. To achieve this, utilize the 'column' parameter to label columns and the 'index' parameter to label rows in the DataFrame.

```
# Add Column & Row Labels to the DataFrame
column_names=
row_label=
df=pd.DataFrame(technologies,columns=column_names,index=row_label)
print(df)
```

Yields below output.

```
# Output
Courses Fee Duration
a Spark 20000 30days
b pandas 20000 40days
```

By default, pandas identify the data types from the data and assign's to the DataFrame. `df.dtypes` returns the data type of each column.

```
# types
df.dtypes
```

Output
Courses object
Fee int64
Duration object
dtype: object

You can also assign custom data types to columns.

```
# Set custom types to DataFrame
types={'Courses': str,'Fee':float,'Duration':str}
df=df.astype(types)
```

Another most used way to create DataFrame is from the dictionary.

```
# Create DataFrame from Dictionary
technologies = {
'Courses':,
'Fee' :,
'Duration':,
'Discount':
}
df = pd.DataFrame(technologies)
```

4.2 Create DataFrame From CSV File

In real-time we are often required to read TEXT, CSV, JSON files to create a DataFrame. In pandas, creating a DataFrame from CSV is pretty simple.

```
# Create DataFrame from CSV file
df = pd.read_csv('data_file.csv')
```

5. DataFrame Basic Operations

In order to explain some basic operations with pandas DataFrame let's create it with some data.

```
# Create DataFrame with None/Null to work with examples
import pandas as pd
import numpy as np
technologies = ({
'Courses':,
'Fee' :,
'Duration':,
'Discount':
})
row_labels=
df = pd.DataFrame(technologies, index=row_labels)
```

```
print(df)
```

Note that our data contains `np.nan`, `None`, and `empty` values. Note that every column in DataFrame is internally represented as pandas Series.

5.1 DataFrame properties

DataFrame has several properties, in this pandas DataFrame tutorial I will cover most used properties with examples.

Method/Property	Result	Description
`df.shape`	(8, 4)	Returns a shape of the pandas DataFrame (number of rows and columns) as a tuple. Number of rows and columns
`df.size`	32	Returns number of cells. It would be rows * columns.
`df.empty`	False	Return boolean. True when DF is empty.
`df.columns`	Index(, dtype='object')	Returns all column names as Series
`df.columns.values`		Returns column names from the header as a list in pandas.
`df.index`	Index(, dtype='object')	Returns Index of DataFrame
`df.index.values`		Returns Index as List.
`df.dtypes`	Courses object Fee float64 Duration object Discount int64 dtype: object	Returns Data types of columns
`df` `df]`	r0 22000.0 r1 25000.0 r2 23000.0 r3 24000.0 r4 NaN r5 25000.0 r6 25000.0 r7 22000.0 Name: Fee, dtype: float64	Pandas Select Columns by Name. Also, use to select multiple columns
`df2=df == 22000]`	Courses Fee Duration Discount r0 Spark 22000.0 30day 1000 r7 Python 22000.0 50days 1600	Filter DataFrame

Method/Property	Result	Description
`df2=df`	Courses Fee Duration Discount r6 Spark 25000.0 30day 1400 r7 Python 22000.0 50days 1600	Select Dataframe Rows by Index Select's Row from 6th Index
`df` `df.values`	40days	Get cell value (row x column) of DataFrame
`df = df - 500` `df`	r0 21500.0 r1 24500.0 r2 22500.0 r3 23500.0 r4 NaN r5 24500.0 r6 24500.0 r7 21500.0	Update DataFrame Column Substract 500 from 'Fee' Column
df = ''	Add new column with empty values	

6. Manipulate DataFrame

6.1 Describe DataFrame

`describe()` - describe function calculates count, mean, std, min, max, and different percentages of each numeric column of pandas DataFrame.

```
# Describe DataFrame for all numberic columns
df.describe()
```

Output
Fee Discount
count 7.000000 8.000000
mean 23714.285714 1537.500000
std 1380.131119 570.557372
min 22000.000000 1000.000000
25% 22500.000000 1150.000000
50% 24000.000000 1350.000000
75% 25000.000000 1775.000000
max 25000.000000 2500.000000

6.2 Filter Rows from DataFrame

`query()/apply()/loc` - These are used to query pandas DataFrame. you can also do operator chaining while filtering pandas rows.

```
# Using DataFrame.query()
df.query("Courses == 'Spark'",inplace=True)
df.query("Courses != 'Spark'")
df.query("Courses in ('Spark','PySpark')")
df.query("`Courses Fee` >= 23000 and `Courses Fee` <= 24000")
```

```
# Using DataFrame.loc
df.loc == value]
df.loc != 'Spark']
df.loc.isin(values)]
df.loc.isin(values)]
df.loc >= 1000) & (df <= 2000)]
df.loc >= 1200) & (df >= 23000 )]

# Using apply()
df.apply(lambda row: row.isin()])

# Other ways to filter
df == 'Spark']
df.str.contains("Spark")]
df.str.lower().str.contains("spark")]
df.str.startswith("P")]
```

6.3 Insert Rows & Columns to DataFrame

`insert()/assign()` - Adds a new column to the pandas DataFrame

By using assign() & insert() methods you can add one or multiple columns to the pandas DataFrame.

```
df = pd.DataFrame(technologies, index=row_labels)

# Adds new column 'TutorsAssigned' to DataFrame
tutors =
df2 = df.assign(TutorsAssigned=tutors)
```

```
# Add new column from existing column
df2=df.assign(Discount_Percent=lambda x: x.Fee * x.Discount / 100)

# Other way to add a column
df = tutors

# Add new column at the beginning
df.insert(0,'TutorsAssigned', tutors )
```

6.4 Rename DataFrame Columns

`rename()` - Renames pandas DataFrame columns

Pandas DataFrame.rename() method is used to change/replace columns (single & multiple columns), by index, and all columns of the DataFrame.

```
df = pd.DataFrame(technologies, index=row_labels)

# Assign new header by setting new column names.
df.columns=

# Change column name by index. This changes 3rd column
df.columns.values = "C"

# Rename Column Names using rename() method
df2 = df.rename({'a': 'A', 'b': 'B'}, axis=1)
df2 = df.rename({'a': 'A', 'b': 'B'}, axis='columns')
df2 = df.rename(columns={'a': 'A', 'b': 'B'})

# Rename columns inplace (self DataFrame)
df.rename(columns={'a': 'A', 'b': 'B'}, inplace = True)

# Rename using lambda function
df.rename(columns=lambda x: x, inplace=True)

# Rename with error. When x not present, it thorows error.
df.rename(columns = {'x':'X'}, errors = "raise")
```

6.5 Drop DataFrame Rows and Columns

`drop()` - drop method is used to drop rows and columns

Below are some examples. In order to understand better go through drop rows from panda DataFrame with examples. dropping rows doesn't complete without learning how to drop rows with/by condition

```
df = pd.DataFrame(technologies, index=row_labels)
```

# Drop rows by labels
df1 = df.drop()

# Delete Rows by position
df1=df.drop(df.index])

# Delete Rows by Index Range
df1=df.drop(df.index)

# When you have default indexs for rows
df1 = df.drop(0)
df1 = df.drop()
df1 = df.drop(range(0,2))

# Drop rows by checking conditions
df1 = df.loc >=1500]

# DataFrame slicing
df2=df # Returns rows from 4th row
df2=df # Removes first and last row
df2=df # Return rows between 2 and 4

Now let's see how to how to drop columns from pandas DataFrame with examples. In order to drop columns, you have to use either `axis=1` or `columns` param to drop() method.

```
df = pd.DataFrame(technologies, index=row_labels)
```

# Delete Column by Name
df2=df.drop(, axis = 1)

# Drop by using labels & axis
df2=df.drop(labels=, axis = 1)

Drop by using columns

df2=df.drop(columns=)

# Drop column by index
df2=df.drop(df.columns], axis = 1)

Other ways to drop columns
df.loc.columns, axis = 1, inplace=True)
df.drop(df.iloc, axis=1, inplace=True)

If you wanted to drop duplicate rows from pandas DataFrame use DataFrame.drop_duplicates()

7. Pandas Join, Merge, Concat to Combine DataFrames

In this section of the python pandas tutorial I will cover how to combine DataFrame using join(), merge(), and concat() methods. All these methods perform below join types. All these join methods works similarly to SQL joins.

Join Types	Supported By	Description
inner	join(), merge() and concat()	Performs Inner Join on pandas DataFrames
left	join(), merge()	Performs Left Join on pandas DataFrames
right	join(), merge()	Performs Right Join on pandas DataFrames
outer	join(), merge() and concat()	Performs Outer Join on pandas DataFrames
cross	merge()	Performs Cross Join on pandas DataFrames

Both pandas.merge() and DataFrame.merge() operate similarly, allowing the merging of two or more DataFrames. When performing a join based on columns, they disregard indexes. However, when joining on the index, the resulting DataFrame retains the indexes from the source DataFrames. In cases where no parameters are specified, the default behavior is to perform the join on all common columns.

```
# Quick Examples of pandas merge DataFrames
# pandas.merge()
df3=pd.merge(df1,df2)
```

# DataFrame.merge()
df3=df1.merge(df2)

Merge by column

df3=pd.merge(df1,df2, on='Courses')

# Merge on different colunn names
df3=pd.merge(df1,df2, left_on='Courses', right_on='Courses')

# Merge by Index
df3 = pd.merge(df1,df2,left_index=True,right_index=True)

# Merge by multiple columns
df3 = pd.merge(df3, df1, how='left', left_on=, right_on =)

# Merge by left join
df3=pd.merge(df1,df2, on='Courses', how='left')

# Merge by right join
df3=pd.merge(df1,df2, on='Courses', how='right')

# Merge by outer join
df3=pd.merge(df1,df2, on='Courses', how='outer')

Alternatively use join() for joining on the index. pandas.DataFrame.join() method is the most efficient way to join two pandas DataFrames on row index.

```
# pandas default join
df3=df1.join(df2, lsuffix="_left", rsuffix="_right")
```

# pandas Inner join DataFrames
df3=df1.join(df2, lsuffix="_left", rsuffix="_right", how='inner')

# pandas Right join DataFrames
df3=df1.join(df2, lsuffix="_left", rsuffix="_right", how='right')

# pandas outer join DataFrames
df3=df1.join(df2, lsuffix="_left", rsuffix="_right", how='outer')

# pandas join on columns
df3=df1.set_index('Courses').join(df2.set_index('Courses'), how='inner')

Similarly, pandas also support concatenate two pandas DataFrames using concat() method.

8. Iterate over Rows to perform an operation

Pandas DataFrame offers two methods, iterrows() and itertuples(), for iterating over each row. With iterrows(), you receive a tuple containing the index of the row and a Series representing its data. Conversely, itertuples() returns all DataFrame elements as an iterator, with each row represented as a tuple. Notably, itertuples() is quicker than iterrows() and maintains data types intact.

```
df = pd.DataFrame(technologies, index=row_labels)
```

# Iterate all rows using DataFrame.iterrows()
for index, row in df.iterrows():
print (index,row, row)

# Iterate all rows using DataFrame.itertuples()
for row in df.itertuples(index = True):
print (getattr(row,'Index'),getattr(row, "Fee"), getattr(row, "Courses"))

# Using DataFrame.index
for idx in df.index:
print(df, df)

9. Working with Null, np.NaN & Empty Values

Below are some of the articles that I have covered to handle None/NaN values in pandas DataFrame. It is very important to handle missing data in Pandas before you perform any analytics or run with machine learning algorithms.

10. Column Manipulations

One most used way to manipulate is by using pandas apply() function to DataFrame columns. If you are familiar with the lambda expressions, you can also use lambda expression with apply().

If you're new to the concept of lambda functions, they're essentially concise, anonymous functions in Python capable of handling any number of arguments and executing expressions. These expressions are particularly handy for creating functions on-the-fly without the need for formal definition using the lambda keyword.

DataFrame also provides several methods to manipulate data on columns.

11. Pandas Read & Write Excel

Use pandas DataFrame.to_excel() function to write a DataFrame to an excel sheet with extension .xlsx and use pandas.read_excel() function is used to read excel sheet with extension xlsx into pandas DataFrame

Read excel sheet Example

```
# Read Excel file
df = pd.read_excel('c:/apps/courses_schedule.xlsx')
print(df)
```

Write DataFrame to excel sheet

```
# Write DataFrame to Excel file
df.to_excel('Courses.xlsx')
```

Related Articles