Pandas Basics

27.09.2019

Python Notes

Intro

- Pandas is one of the most popular Python libraries for Data Science and Analytics.
- It’s like the “SQL of Python.”
- Because pandas helps you to manage two-dimensional data tables in Python.
- We’ll note the most important (that is, the most often used) things that you have to know .

Why Pandas?

- NumPy is used for matrixes. Pandas is used for dataframes.
- Pandas is FAST and EFFICIENT library for dataframes
- Allows easy switching between different data file types
- Easy handle of missing data
- Efficient reshaping, slicing and indexing

Pandas Data Structures

- There are two types of data structures in pandas:

- **Series:** is a one dimensional data structure (*"a one dimensional ndarray"*) that can store values — and for every value it holds a unique index, too.
- **Dataframe:** is a two (or more) dimensional data structure – basically a table with rows and columns. The columns have names and the rows have indexes.

Building a Dataframe

- We generally import the data to build dataframes, but for the sake of clarity we will build it here.
- Arrays get lists, Dataframes get dictionaries

Example: Create a dataframe by using a dictionary.

```
import pandas as pd


#Every key has a 6 element lists as a value.
dictionary = {"NAME":["ali","veli","kenan","hilal","ayse","evren"],
              "AGE" : [15,16,17,33,45,66],
              "SALARY":[100,150,240,350,110,220]}


#Create a dataframe. Each key becomes a column name or feature. List values alines under its feautre.
dataFrame1 = pd.DataFrame(dictionary)

dataFrame1
```

Out :

```
NAME  AGE  SALARY
0   ali   15     100
1  veli   16     150
2 kenan   17     240
3 hilal   33     350
4  ayse   45     110
5 evren   66     220
```

Basic Methods

Example: Basic dataframe methods

```
#return first 5 rows of the dataFrame
head = dataFrame1.head()

#return first 3 rows of the dataFrame
head = dataFrame1.head(3)

#return last 3 rows of the dataFrame
tail = dataFrame1.tail()

#return features. I can reach them by indexing dataFrame1.columns[0]
dataFrame1.columns

#returns general info of the dF
dataFrame1.info()

#returns datatypes for each column. Each column can store only 1 datatype.
dataFrame1.dtypes

#return some info about numeric features
dataFrame1.describe()
```

Indexing and Slicing

Example:

```
#return the entire NAME COLUMN as a series.
dataFrame1["NAME"]
dataFrame1.NAME #alternative way
```

Out :

```
0      ali
1     veli
2    kenan
3    hilal
4     ayse
5    evren
```

```
#Use loc in order to index as matrixes.

#return all rows and 'AGE' column.
dataFrame1.loc[:,"AGE"]

#return 0th, 1th,2th rows and 'SALARY' column.
dataFrame1.loc[0:2,"SALARY"]
```

```
# 'NAME' to 'SALARY' columns.
dataFrame1.loc[:3,"NAME":"SALARY"]

# until 'NAME'
dataFrame1.loc[:,:"NAME"]

# 'SALARY' and 'NAME'
dataFrame1.loc[:3,["SALARY","NAME"]]

# reverse rows
dataFrame1.loc[::-1,:]

# For integer locations use iloc:
dataFrame1.iloc[:,0:2]
```

Adding a New Feature

Example: Add new feature to the dF

```
# Define a feature and assign a list to it.
dataFrame1["NewFeature"] = [-1,-2,-3,-4,-5,-6]
```

Out :

```
    NAME  AGE  SALARY  NewFeature
0    ali   15     100          -1
1   veli   16     150          -2
2  kenan   17     240          -3
3  hilal   33     350          -4
4   ayse   45     110          -5
5  evren   66     220          -6
```

Filtering the DataFrame

- I might wanna filter the rows which has salary > 200 or age < 20 or maybe both.

Example:

```
# Define a filter.
filter200 = dataFrame1.SALARY > 200 # First it gets the SALARY column then returns true or false for each element. type--> Series
```

Out :

```
0    False
1    False
2     True
3     True
4    False
5     True
```

```
# Now filter the dF.
filtered_dataFrame = dataFrame1[filter200] # filtered dF. All columns, only true rows.
```

Out :

```
    NAME   AGE   SALARY  NewFeature  NewFeature2
2  kenan    17     240          -3           -3
3  hilal    33     350          -4           -4
5  evren    66     220          -6           -6
```

```
# I might have done the same thing in a single line as well:
filtered_dataFrame1 = dataFrame1[dataFrame1.SALARY > 200]

# If I want to find rows with salary>200 and age<20. I can combine filters:
filter20 = dataFrame1.AGE < 20
combinedFiltered_dataFrame = dataFrame1[filter200 & filter20]
```

Dropping and Concetanating DataFrames

Example:

```
# Let's add a new feature
dataFrame1["NewColumn"] = [-11,-22,-33,-44,-55,-66]

# Let's drop this new feautre. axis=1 means column drop. inplace=True means change parent dF.
dataFrame1.drop(["NewFeature"],axis=1,inplace=True)
```

```
# Let's concatenate head and tail dFs
data1 = dataFrame1.head()
data2 = dataFrame1.tail()

# Horizontal
dataConcat = pd.concat([data1,data2],axis=0)


# Vertical
dataConcat = pd.concat([data1,data2],axis=1)
```

List Comprehension

- We generally use it to preprocess our data and make it ready for upcoming processing stages
- There are many different applications for LIST COMPREHENSION. You can check it on google.
- It is basically an efficient way of of building new lists by iterating through another list.

Example: Create a new column and label each row as "HighSalary" if salary > averageSalary:

```
# First find the mean for SALARY
averageSalary = dataFrame1.SALARY.mean()

# Add a new column. Assign a list generated by list comprehension.
dataFrame1 ["SalaryLevel"] = ["HighSalary" if each > averageSalary  else "LowSalary" for each in dataFrame1.SALARY]
```

Example: Change uppercase columns to lowercase columns by list comprehension:

```
# get every element in columns then turn them into lowercase strings and build a new list and finally assign the list.
dataFrame1.columns =[each.lower() for each in dataFrame1.columns]
```

Transforming DataFrames

```
Example: Let's create a new feature and put age^2

# Use list comprehension
dataFrame1["Age^2"] = [each**2 for each in dataFrame1.AGE]

#I can to the same operation by apply() method.
def square(age):
    return age**2

#Build a new feature. Assign created series to the new feature.
dataFrame1["Age^2"] = dataFrame1.AGE.apply(square)
```

More blogs