Digital Skills Training for Student
Central Computer Center, KUET

Machine Learning with Python (ML-1)

Session 9: Pandas DataFrame

Contents

Session 9: Pandas DataFrame

Pandas

Pandas is a Python library used for working with data sets.

It has functions for analyzing, cleaning, exploring, and manipulating data.

The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis" and was created by Wes McKinney in 2008.

Install it using this command:

```
C:\Users\Your Name>pip install pandas
```

```python
import pandas
import pandas as pd
print(pd.__version__)
```

https://pandas.pydata.org/
https://github.com/pandas-dev/pandas

Pandas Series

A Pandas Series is like a column in a table.

It is a one-dimensional array holding data of any type.

```
import pandas as pd
a = [1, 7, 2]
myvar = pd.Series(a)
print(myvar)
```

If nothing else is specified, the values are labeled with their index number.

```
print(myvar[0])
```

Pandas DataFrames

Data sets in Pandas are usually multi-dimensional tables, called DataFrames.

Series is like a column, a DataFrame is the whole table.

```
data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}
df = pd.DataFrame(data)

print(df)
```

Pandas DataFrames

Pandas use the **loc** attribute to return one or more specified row(s)

```
#refer to the row index:
print(df.loc[0])
print(df.loc[[0, 1]])
```

[This example returns a Pandas **Series.**]

[When using [ ], the result is a Pandas DataFrame]

```
df = pd.DataFrame(data, index = ["day1", "day2", "day3"])
print(df)

#refer to the named index:
print(df.loc["day2"])
```

Pandas Read CSV

A simple way to store big data sets is to use CSV files (comma separated files).

CSV files contains plain text and is a well know format that can be read by everyone including Pandas.

```
import pandas as pd
df = pd.read_csv('/content/drive/MyDrive/EDGE Training/Pandas/data.csv')
print(df.to_string())  [to print the entire DataFrame.]
print(df)              [will only return the first 5 rows, and the last 5 rows]
df
print(pd.options.display.max_rows)
pd.options.display.max_rows = 9999
print(df)
```

Pandas Read JSON

Big data sets are often stored, or extracted as JSON. [**JSON = Python Dictionary]**

JSON is plain text, but has the format of an object, and is well known in the world of programming, including Pandas.

```
import pandas as pd
df = pd.read_json('/content/drive/MyDrive/data.js’)
print(df.to_string())
print(df)
```

Selecting cols and rows from dataframe

```
Pulse = df["Pulse"]
print(Pulse)
```

[select specific columns from a DataFrame]

```
Pulse_MaxPulse = df[["Pulse", "Maxpulse"]]
print(Pulse_MaxPulse)

above_100 = df[df["Pulse"] > 100]
print(above_100)
```

[filter specific rows]

```
p_max = df.loc[df["Pulse"] > 35, "Maxpulse"]
print(p_max)
```

[specific rows and columns]

```
df.iloc[9:25, 2:5]
```

Selecting cols and rows from dataframe

REMEMBER

- When selecting subsets of data, square brackets [] are used.
- Inside these brackets, you can use a single column/row label, a list of column/row labels, a slice of labels, a conditional expression or a colon.
- Select specific rows and/or columns using loc when using the row and column names.
- Select specific rows and/or columns using iloc when using the positions in the table.
- You can assign new values to a selection based on loc/iloc.

DataFrame attributes and methods

- Attributes provide direct information about the DataFrame (its structure, size, or column names).
- Methods perform actions on the DataFrame, like modifying or analyzing the data.

Attribute	Method
• Stores metadata	• Performs operations or calculations
• No parentheses required	• Requires parentheses
• Cannot take arguments	• Can take arguments to customize behavior
• Often returns static info (e.g., columns, shape, dtypes)	• Often modifies, summarizes, or transforms the data
• Example: df.shape	• Example: df.drop()

Analyzing DataFrames

```
#Viewing the Data
print(df.head(10))          [Return the first n rows.]
print(df.head(-100))        [Returns all rows except the last |n| rows]

print(df.tail(5))           [Return the last n rows.]
print(df.tail(-5))          [Returns all rows except the first |n| rows]

print(df.info())
print(df.index)
print(df.index)
print(df.index)
df.select_dtypes(include=['float64'],exclude=['int64'])
```

Analyzing DataFrames

```
#Viewing the Data
print(df.values)
print(df.axes)
```
[Return a Numpy representation of the DataFrame.]

[Return a list representing the axes of the DataFrame.]

```
df.ndim
```
[Return 1 if Series. Otherwise return 2 if DataFrame.]

```
df.size
```
[Return an int representing the number of elements in this object.]

```
df.shape
```
[Return a tuple representing the dimensionality of the DataFrame.]

```
df.memory_usage(index=False, deep=True)
```
[Return the memory usage of each column in bytes.]

```
df.empty
```
[Indicator whether Series/DataFrame is empty.]

Analyzing DataFrames

```
df.astype(dtype, copy=None, errors='raise’)
```

[Cast a pandas object to a specified dtype]

Dtype : ***str, data type, Series or Mapping of column name -> data type***
Copy : ***bool, default True***
Errors : ***{‘raise’, ‘ignore’}, default ‘raise’***

```
df.dtypes
df.astype('int32').dtypes
df.astype({'Pulse': 'int32'}).dtypes
print(df.describe())
```

Analyzing DataFrames

```
df.convert_dtypes(infer_objects=True, convert_string=True,
convert_integer=True, convert_boolean=True,
convert_floating=True, dtype_backend='numpy_nullable’)
```

[Convert columns to the best possible dtypes using dtypes supporting pd.NA.]

infer_objects : *bool, default True*
convert_string : *bool, default True*
convert_integer : *bool, default True*
convert_Boolean : *bool, defaults True*
convert_floating : *bool, defaults True*
dtype_backend : *{‘numpy_nullable’, ‘pyarrow’}, default ‘numpy_nullable’*

Adding new columns

```
df["2xPulse"] = df["Pulse"] * 2
df_renamed = df.rename(
    columns={
        “name1": “name2",
        “col1": “col2",
    }
)
```

REMEMBER

- Create a new column by assigning the output to the DataFrame with a new column name in between the [].
- Operations are element-wise, no need to loop over rows.
- Use rename with a dictionary or function to rename row labels or column names.

Cleaning Data

Data cleaning means fixing bad data in your data set.
Bad data could be:

- Empty cells
- Data in wrong format
- Wrong data
- Duplicates

data.csv

- ➢ contains some empty cells ("Date" in row 22, and "Calories" in row 18 and 28).
- ➢ contains wrong format ("Date" in row 26).
- ➢ contains wrong data ("Duration" in row 7).
- ➢ contains duplicates (row 11 and 12).

Cleaning Data

Empty Cells

Empty cells can potentially give you a wrong result when you analyze data.

1. **Remove Rows**

- One way to deal with empty cells is to remove rows that contain empty cells.
- This is usually OK, since data sets can be very big, and removing a few rows will not have a big impact on the result.

```
new_df = df.dropna()
print(new_df.to_string())
```

[By default, the dropna() method returns a new DataFrame, and will not change the original]

```
df.dropna(inplace = True)
```

Cleaning Data

2. Replace Empty Values

- This way you do not have to delete entire rows just because of some empty cells.

`df.fillna(130)` [insert a *new* value instead.]

`df["Calories"].fillna(130)` [Replace Only For Specified Columns]

```
x = df["Calories"].mean()
df["Calories"].fillna(x, inplace = True)

x = df["Calories"].median()
df["Calories"].fillna(x, inplace = True)

x = df["Calories"].mode()[0]
df["Calories"].fillna(x, inplace = True)
```

Cleaning Data

Data of Wrong Format

- Cells with data of wrong format can make it difficult, or even impossible, to analyze data.

- To fix it, you have two options: remove the rows, or convert all cells in the the columns into the same format.

1. **Convert Into a Correct Format**

```
df['Date'] = pd.to_datetime(df['Date'])
print(df.to_string())
```

2. **Removing Rows**

```
df.dropna(subset=['Date'], inplace = True)
```

Cleaning Data

Wrong Data

"Wrong data" does not have to be "empty cells" or "wrong format", it can just be wrong, like if someone registered "199" instead of "1.99".

1. **Replacing Values**

```
df.loc[7, 'Duration'] = 45
```

   To replace wrong data for larger data sets you can create some rules

```
for x in df.index:
  if df.loc[x, "Duration"] > 120:
    df.loc[x, "Duration"] = 120
```

2. **Removing Rows**

```
for x in df.index:
  if df.loc[x, "Duration"] > 120:
    df.drop(x, inplace = True)
```

Cleaning Data

Duplicates

Duplicate rows are rows that have been registered more than one time.

```
print(df.duplicated())
```

Removing Duplicates

```
df.drop_duplicates(inplace = True)
```

Data Correlations

Finding Relationships

A great aspect of the Pandas module is the corr() method.

The corr() method calculates the relationship between each column in your data set.

`df.corr()` [The corr() method ignores "not numeric" columns.]

Perfect Correlation: 1.000000
Good Correlation: 0.922721
Bad Correlation: 0.009403

	Duration	Pulse	Maxpulse	Calories
Duration	1.000000	-0.155408	0.009403	0.922721
Pulse	-0.155408	1.000000	0.786535	0.025120
Maxpulse	0.009403	0.786535	1.000000	0.203814
Calories	0.922721	0.025120	0.203814	1.000000

Filtering a Dataframe

```
DataFrame.filter(items=None, like=None, regex=None,
axis=None)
```

[Subset the dataframe rows or columns according to the specified index labels.]

Items - ***list-like***
Like - ***str***
Regex - ***str (regular expression)***
Axis - ***{0 or 'index', 1 or 'columns', None}, default None***

pandas.DataFrame.groupby

```
DataFrame.groupby(by=None, axis=<no_default>, level=None, as_index=True, sort=True, group_keys=True, observed=<no_default>, dropna=True)
```

Group DataFrame using a mapper or by a Series of columns.

```
grouped_data = df.groupby('Gender')['Salary'].mean()print(grouped_data)
```

Apply a Function to a Column

```
# Define a function to increase salary by 10%
def increase_salary(salary):
    return salary * 1.10

# Apply the function to the 'Salary' column
df['Salary'] = df['Salary'].apply(increase_salary)
print(df.head())
```

Practice Problem

Load nba.csv and perform following tasks:

1. Display a summary of the dataset
2. Handle Missing Values
3. Filter Data Based on Conditions
4. Group the data by a categorical column and calculate the mean of another column.
5. Identify and remove duplicate rows from the dataset.
6. Sort the DataFrame based on one or more columns.
7. Create a new column based on existing data.
8. Apply a custom function to modify values in a column.
9. Merge two DataFrames.
10. Create a histogram of a numeric column (e.g., "Age") and a bar plot of categorical data (e.g., "Gender").

Practice Problem

Load nba.csv and perform following tasks:

1. Detect and remove outliers based on a numerical column.
2. Create a pivot table summarizing data.
3. Filter the data based on a date range (if time series data exists).
4. Analyze correlations between numerical columns.
5. Perform multi-level grouping and aggregation.
6. Normalize a numeric column using Min-Max Scaling.
7. Create a new column based on multiple conditions.
8. Count the number of unique values in a categorical column.
9. Fill missing values with group-specific averages.
10. Calculate a rolling average for a numeric column.
11. Resample data if the dataset contains time-series data.
12. Calculate the percentage change in a numeric column over time.

Thank You!