PPDS Week 5 Wednesday

Pandas continued

Data science with python

Karl-Emil Kjær Bilstrup

17.12.25

Python REPL Advertisement

Read, Evaluate, Print, and Loop

For exploration and to test assumptions

```
% python
>>> pd_series = pd.Series({'a':1.0, 'b':2.0, 'c':3.0})
>>> pd_series
a    1.0
b    2.0
c    3.0
dtype: float64
```

Creating a Series

```
import pandas as pd

# From a list with custom index
pd_series = pd.Series([1.0, 2.0, 3.0],
                      index=['a', 'b', 'c'])

# From a dictionary
pd_series = pd.Series({'a':1.0, 'b':2.0, 'c':3.0})

print(pd_series)
# Output:
# a    1.0
# b    2.0
# c    3.0
```

Creating a DataFrame

```
import pandas as pd
import numpy as np

# From a dictionary of lists
data = {
    'Name': ['Alice', 'Bob', 'Charlie'],
    'Age': [25, 30, 35],
    'City': ['NY', 'SF', 'LA']
}
df = pd.DataFrame(data)

print(df)
#      Name  Age City
# 0   Alice   25   NY
# 1     Bob   30   SF
# 2  Charlie  35   LA
```

Basic DataFrame Operations

```
# View first few rows
print(df.head())

# Get column names
print(df.columns)

# Access a column
print(df['Name'])

# Add a new column
df['Country'] = ['USA', 'USA', 'USA']

# Basic statistics for numeric columns
print(df.describe())
```

Reading and Writing Data (I/O)

Pandas makes it easy to work with different file formats:

	Read	Write
general	`read_table()`	
csv	`read_csv()`	`to_csv()`
html	`read_html()`	`to_html()`
json	`read_json()`	`to_json()`
excel	`read_excel()`	`to_excel()`
sql	`read_sql()`	`to_sql()`
...		

pandas - Dataframe - methods

Numpy functions apply to dataframes

```
np_array = np.arange(9).reshape((3,3))
df = pd.DataFrame(np_array, index=['a', 'b', 'c'],
                  columns=['col1', 'col2','col3'])

# stats
print(df.mean(axis=0), df.std(axis=0), df.median(axis=0))

# matrix multiplication
print(np.dot(df, df.T))
```

axis: `0` or `index` -> along index; `1` or `columns` -> along columns

Basic Pandas DataFrame Methods

Method	Description	Example
`head()`, `tail()`	View first/last rows	`df.head(3)`
`info()`	Get DataFrame info (types, memory)	`df.info()`
`describe()`	Summary statistics	`df.describe()`
`shape`	Get dimensions (rows, columns)	`df.shape`
`columns`	Get column names	`df.columns`
`dtypes`	Get data types of columns	`df.dtypes`
`value_counts()`	Count unique values	`df['column'].value_counts()`
`sort_values()`	Sort by values	`df.sort_values('column')`
`groupby()`	Group data	`df.groupby('column').mean()`
`dropna()`	Remove missing values	`df.dropna()`
`fillna()`	Fill missing values	`df.fillna(0)`
`reset_index()`	Reset index to default	`df.reset_index()`

Common Selection Methods

Operation	Description	Example
Select column	Get single column	`df['column']`
Select multiple columns	Get several columns	`df[['col1', 'col2']]`
Select by label	Get rows by label	`df.loc[row_label]`
Select by position	Get rows by position	`df.iloc[0:2]`
Boolean indexing	Filter with condition	`df[df['Age'] > 30]`
Set column	Add/modify column	`df['new_col'] = values`

Pandas provides specific support for Series of certain types, through so-called Accessors.

Data type	Accessor
String	str
Categorical	cat
Datetime	dt
Sparse	sparse

```
pd_series.str.upper()
```

pandas - Dataframe - selecting with boolean array

```
np_array = np.arange(9).reshape((3,3))
df = pd.DataFrame(np_array, index=['a', 'b', 'c'],
                  columns=['col1', 'col2','col3'])
# I want to take all values greater than 4
mask = (df > 4)
print(df[mask])
    col1  col2  col3
a   NaN   NaN   NaN
b   NaN   NaN   5.0
c   6.0   7.0   8.0
# I want to zero out all large values
df[mask] = 0
print(df)
col1  col2  col3
a     0     1     2
b     3     4     0
c     0     0     0
```

pandas - Dataframe - selecting with boolean array

You can also create masks for selecting whole rows or columns

Selecting rows:

```
np_array = np.arange(9).reshape((3,3))
df = pd.DataFrame(np_array, index=['a', 'b', 'c'],
                  columns=['col1', 'col2','col3'])

# take all rows whose first element is greater than 2
row_mask = df.iloc[:,0] > 2
print(df[row_mask])
# or print(df.loc[row_mask])
   col1  col2  col3
b     3     4     5
c     6     7     8
```

pandas - Dataframe - selecting with boolean array

Selecting columns:

```
np_array = np.arange(9).reshape((3,3))
df = pd.DataFrame(np_array, index=['a', 'b', 'c'],
                  columns=['col1', 'col2','col3'])

# take all columns whose elements sum to greater 10
col_mask = df.sum(axis=0) > 10
df.loc[:,col_mask]
col2  col3
a     1     2
b     4     5
c     7     8
```

Exercise 1

1. Download the `british-english` file from absalon or Github
2. Read it into a `DataFrame` . Are there any options you need to add to make it work? **Hint 1: Print the dataframe and see if you can find an issue. Hint 2: Check the Header parameter in the documentation. Hint 3: Continue to 3. and see if it helps you spot the issue.**
3. Figure out how to assign a different column name - e.g. 'words' **Hint: Start by figuring out how to print the column name, then assign a new name.**
4. Figure out how to select all rows starting with the letter A. **Hint 1: Use the string accessor on a column. Hint 2: See str startswith methods in documentation.**

pandas - Dataframe - adding columns

Add a new column by assigning to the relevant label

```
np_array = np.arange(9).reshape((3,3))
df = pd.DataFrame(np_array, index=['a', 'b', 'c'],
                  columns=['col1', 'col2','col3'])
# Adding a copy of col1 to col4
df.loc[:,'col4'] = [0,3,6]
# Using df.assign
df = df.assign(col4 = df.loc[:,'col1'])
# Using df.insert
df.insert(3, 'col4', df.loc[:,'col1'])
print(df)
   col1  col2  col3  col4
a     0     1     2     0
b     3     4     5     3
c     6     7     8     6
```

pandas - Dataframe - adding columns

If columns don't have the expected index, missing elements will be set to NaN

```
np_array = np.arange(9).reshape((3,3))
df = pd.DataFrame(np_array, index=['a', 'b', 'c'],
                  columns=['col1', 'col2','col3'])
# Adding a copy of col1 to col4
df.loc[:,'col4'] = df.loc['b':,'col1']
print(df)
   col1  col2  col3  col4
a     0     1     2   NaN
b     3     4     5     3
c     6     7     8     6
```

pandas - Dataframe - adding rows

Add a new row by assigning to the relevant label

```
np_array = np.arange(9).reshape((3,3))
df = pd.DataFrame(np_array, index=['a', 'b', 'c'],
                  columns=['col1', 'col2','col3'])
# Adding a copy of a to d
df.loc['d',:] = df.loc['a',:] # or [0,1,2]
# df.insert and df.assign only support adding columns
print(df)
   col1  col2  col3
a     0     1     2
b     3     4     5
c     6     7     8
d     0     1     2
```

pandas - Dataframe - merging

`pd.merge()` combines two dataframes based on a shared column label

```
df1 = pd.DataFrame({'id' : [1, 2, 3],
'gender' : ['male', 'female', 'female']})

df2 = pd.DataFrame({'id' : [1, 2, 3],
'name' : ['Bob', 'Alice', 'Anna']})

df = pd.merge(df1, df2, on='id')
print(df)

    id  gender   name
0   1    male    Bob
1   2  female  Alice
2   3  female   Anna
```

pandas - Dataframe - merging

You can also merge on the index labels

```
df1 = pd.DataFrame({'id' : [1, 2, 3],
'gender' : ['male', 'female', 'female']})
df2 = pd.DataFrame({'id' : [1, 2, 3],
'name' : ['Bob', 'Alice', 'Anna']})
df = pd.merge(df1, df2, left_index=True,right_index=True)
print(df)
```

```
    id_x  gender  id_y   name
0     1    male     1    Bob
1     2  female     2  Alice
2     3  female     3   Anna
```

pandas - Dataframe - concatenation

`pd.concat()` combines multiple dataframes along columns or rows, controlled by the argument `axis`

```
np_array = np.arange(6).reshape((2,3))
df1 = pd.DataFrame(np_array, index=['a', 'b'],
                   columns=['col1', 'col2','col3'])
df2 = pd.DataFrame(np_array, index=['d', 'e'],
                   columns=['col1', 'col2','col3'])

# Concatenation along index
df = pd.concat([df1,df2],axis='index') # axis: {0/'index', 1/'columns'}, default 0
print(df)
   col1  col2  col3
a     0     1     2
b     3     4     5
d     0     1     2
e     3     4     5
```

pandas - Dataframe - concatenation

`pd.concat()` combines multiple dataframes along columns or rows, controlled by the argument `axis`

```
np_array = np.arange(9).reshape((3,3))
df1 = pd.DataFrame(np_array, index=['a', 'b', 'c'],
                   columns=['col1', 'col2','col3'])
df2 = pd.DataFrame(np_array, index=['a', 'b', 'c'],
                   columns=['col4', 'col5','col6'])

# Concatenation along index
df = pd.concat([df1,df2],axis='columns') # axis: {0/'index', 1/'columns'}, default 0
print(df)
    col1  col2  col3  col4  col5  col6
a     0     1     2     0     1     2
b     3     4     5     3     4     5
c     6     7     8     6     7     8
```

pandas - Dataframe - groupby

`df.groupby()` splits the dataframe `df` into different groups

the keyword `by=` determines the grouping rule

```
df = pd.DataFrame({'Animal': ['Falcon', 'Falcon',
                              'Parrot', 'Parrot'],
                   'Max Speed': [380., 370., 24., 26.]})
# I want to calculate the average speed of both species
groups = df.groupby(['Animal'])
# Apply mean to every animal group
avg_speed_df = groups.mean()
print(avg_speed_df)

        Max Speed
Animal
Falcon      375.0
Parrot       25.0
```

pandas - DataFrame - groupby

You can loop over the returned variable to check what the groups are

```
df = pd.DataFrame({'Animal': ['Falcon', 'Falcon',
                              'Parrot', 'Parrot'],
                   'Type': ['Captive', 'Wild', 'Captive', 'Wild'],
                   'Max Speed': [380., 370., 24., 26.]})
# I want to calculate the average speed of both species
groups = df.groupby(['Animal','Type'])
# I want to check what the groups are
for name, value in groups:
    print(name,value)
('Falcon', 'Captive')       Animal  Type  Max Speed
                         0  Falcon  Captive   380.0
('Falcon', 'Wild')          Animal  Type  Max Speed
                         1  Falcon  Wild      370.0
# ... prints two more groups
```

pandas - DataFrame - sorting values

`df.sort_values(by=label)` sorts either by row or column values:

```
np_array = np.random.rand(3,3)
df = pd.DataFrame(np_array,
    index=['a', 'b', 'c'],
    columns=['col1', 'col2','col3'])

print(df)
print(df.sort_values(by='col2', ascending=False))
print(df.sort_values(by='a',axis='columns'))
```

pandas - Remarks

- Indexing and slicing - return type:
  - `[:,:]` -> dataframe
  - `[:,i]`, `[i,:]` -> series
  - `[i,j]` -> single element
- pandas functions/methods are not in-place by default.
  - Some have argument `inplace`, and you may do `df.method_name(...,inplace=True)` instead of `df = df.method_name(...)`
- keywords:
  - axis: `0` or `index` -> along index; `1` or `columns` -> along columns
  - types: `str`, `category`, `float`, `int`

Exercise 2

1. Download `british-english_a-e` and `american-english_a-e` file from absalon
2. Read them into two `DataFrame` variables `df1` and `df2`. Set the column name to `words` for both dataframes
3. Merge the two word lists to a dataframe `df`. **Hint: check the argument `how` in pd.merge() (look in the documentation or search the web)**

What is Data Science?

Data Science is the intersection of:

- Statistics & Programming
- Domain Knowledge

Key activities:

- Data Collection and CLEANING
- Analysis
- Visualization
- Interpretation

Data Science Process

Data Collection

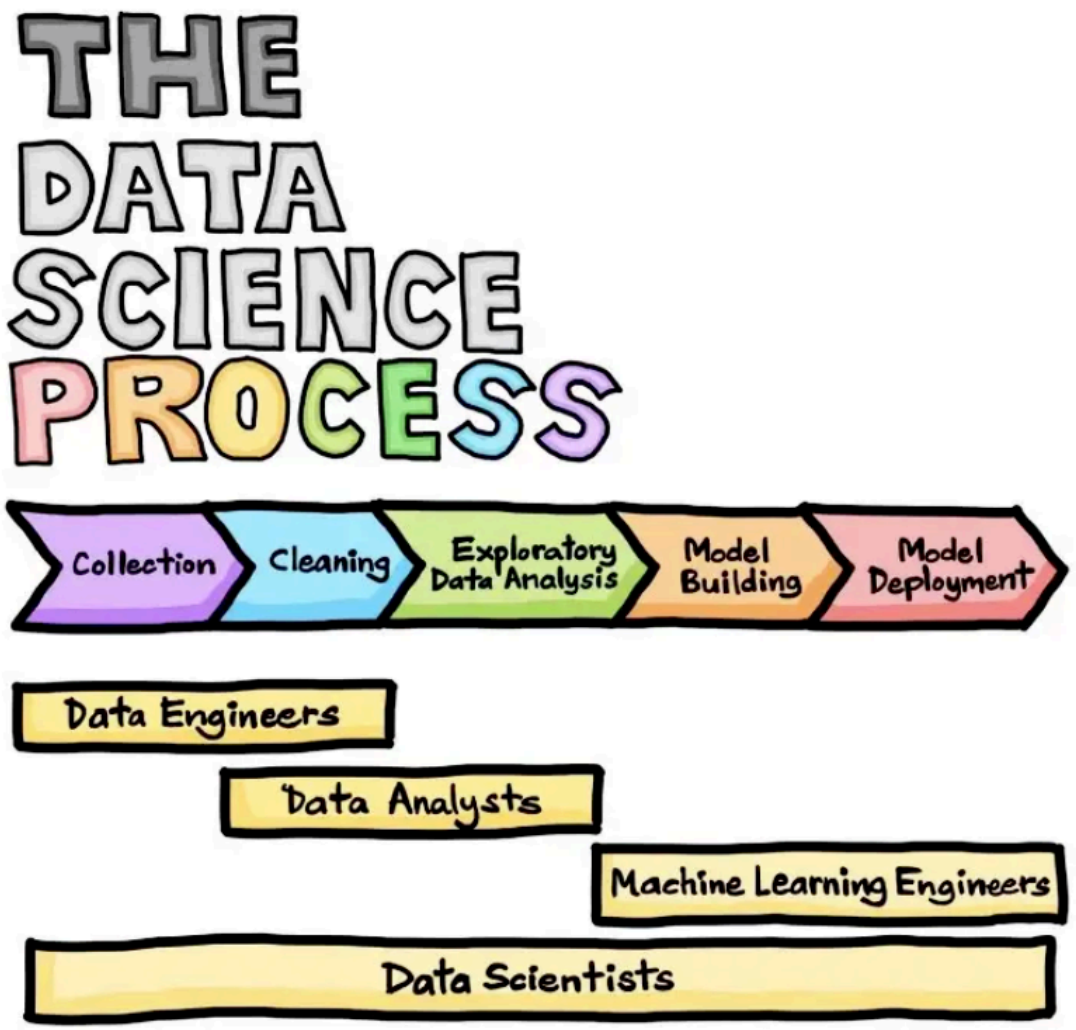
- Gather data that can help the analysis and manipulation

Cleaning

- Clean the raw data and convert it into the desired form format

Exploratory Data Analysis (EDA)

- Analyze the data, identify the relevant factors, suggests directions towards modeling



Why NumPy in Data Science?

- **Efficient Array Operations**
  - Fast computation on large datasets
  - Memory-efficient storage
  - Vectorized operations (no explicit loops needed)
- **Mathematical Operations**
  - Linear algebra operations
  - Statistical functions
  - Random number generation

NumPy Generating Random Data

```
np.random.rand(3,3)
array([[0.45537793, 0.8115471 , 0.74602629],
       [0.84160578, 0.36393521, 0.51246094],
       [0.34451085, 0.39302559, 0.88359236]])

np.random.standard_normal((3,3))
array([[-0.85249074, -0.40046098, -0.58350246],
       [ 3.24900664, -0.0125803 , -0.34059938],
       [ 1.4437066 ,  0.34075023, -2.40908765]])
```

NumPy Data Preprocessing

```
# Handle missing values
data = np.array([1, np.nan, 3, 4])
clean_data = np.nan_to_num(data) # replaces NaN with 0 or another defined value

# Normalize data
normalized = (data - np.mean(data)) / np.std(data)
```

NumPy Feature Engineering

Useful for machine learning

```
# Create polynomial features
x = np.array([1, 2, 3, 4])
poly_features = np.column_stack([x, x**2, x**3])

array([[ 1,  1,  1],
       [ 2,  4,  8],
       [ 3,  9, 27],
       [ 4, 16, 64]])
```

NumPy Statistical Analysis

```
# Basic statistics
mean = np.mean(data)
std = np.std(data)
...
```

Dataset Representation

- Rows: Samples/observations
- Columns: Features/variables

```
# Dataset with 3 samples, 4 features
dataset = np.array([
    [1.2, 2.3, 0.5, 1.0],  # Sample 1
    [2.1, 1.5, 0.8, 2.2],  # Sample 2
    [1.8, 1.9, 0.6, 1.5]   # Sample 3
])
```

NumPy image Processing

- Images as 2D/3D arrays

```
# Grayscale image: 2D array
image = np.array([
    [255, 128, 0],
    [128, 255, 128],
    [0, 128, 255]
])
```

Wrapping NumPy arrays in Pandas DataFrames or Series to get labeled data

You have seen it many time:

```
a = np.array([1, 2, 3])
df = pd.DataFrame({"values": a})
   values
0       1
1       2
2       3
```

How NumPy and Pandas Work Together

- **Pandas uses NumPy arrays internally for data storage**
- **Different roles**
  - NumPy: fast numerical engine
  - Pandas: structure, labels, and data handling
- **Typical workflow**
  i. Use Pandas to load, clean, and organize data
  ii. Use NumPy for fast numerical computations
  iii. Store results back in a DataFrame

NumPy vs Pandas

Comparison Table

Feature	NumPy	Pandas
Main purpose	Numerical computing	Data analysis & manipulation
Core structures	`ndarray`	`Series`, `DataFrame`
Data types	Single data type per array	Mixed data types supported
Indexing	Integer-based	Label-based (`loc`) and integer (`iloc`)
Typical use cases	Math, linear algebra, speed	Tables, CSVs, grouping, cleaning

Data science exercises 2 and 3

Soccer and protein structure

Worksheet on Github and Absalon

Old exam exercises

Still not confident with indexing and slicing arrays and DataFrames? Consider doing the Monday worksheet if you haven't already.