Python Data Collection and Management for Public Policy Research

Day 9: Introduction to *pandas* (Part 2)

Blake Miller[†]

11 July, 2024

[†]Assistant Professor, Department of Methodology, London School of Economics and Political Science (E-mail: b.a.miller@lse.ac.uk)

DataFrame

DataFrame

A *DataFrame* is the 2D analogue of a *Series*: it is essentially a table of heterogeneous objects.

- **Index**: Holds the labels of the rows.
- **Columns**: Holds the labels of the columns.
- **Shape**: Describes the dimension of the table.

When you extract a column from a *DataFrame*, you get a proper *Series*, and you can operate on it using all the tools presented in the previous sections.

Further, most (but not all) of the operations that you can do on a *Series*, you can also do on an entire *DataFrame*.

Data Input and Output

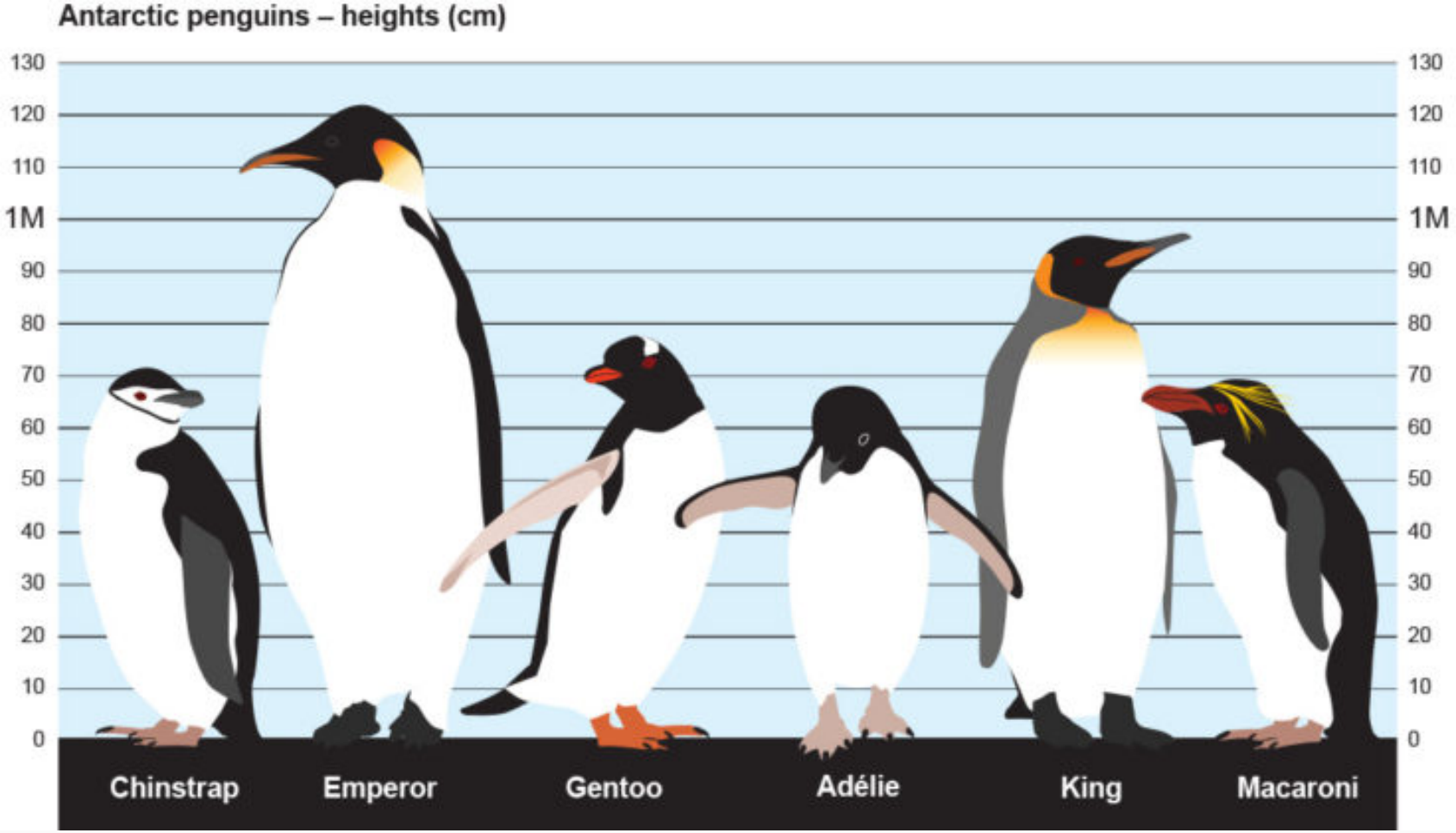
- Pandas can read in a *DataFrame* from various file types:

```
df_csv = pd.read_csv('data.csv')
df_excel = pd.read_excel('data.xlsx')
df_json = pd.read_json('data.json')
```

- Pandas can also export a *DataFrame* to various file types:

```
df.to_csv('data.csv')
df.to_excel('data.xlsx')
df.to_json('data.json')
```

Introduction to the Palmer Penguins Dataset



For more on the Palmer Penguins data,
see more info about the dataset and original paper here.

Introduction to the Palmer Penguins Dataset

The Palmer Penguins dataset consists of size measurements, clutch observations, and blood isotope ratios for three penguin species collected from three islands in the Palmer Archipelago, Antarctica.

- Species: Adélie, Chinstrap, and Gentoo
- Variables include bill length, bill depth, flipper length, body mass, and more.
- Used widely in data science for exploratory analysis and data visualization.

```
penguins = pd.read_csv('penguins.csv')
```

Creating a DataFrame from a Dictionary of Lists

A *DataFrame* can also be from a dictionary of column names and a a list of associated values.

```
>>> d = { "column1": [1., 2., 6., -1.], "column2": [0.,
    1., -2., 4.] }
>>> df = pd.DataFrame(d)
>>> print(df)
   column1  column2
0      1.0      0.0
1      2.0      1.0
2      6.0     -2.0
3     -1.0      4.0
>>> print(df.columns)
Index(['column1', 'column2'], dtype='object')
>>> print(df.index)
RangeIndex(start=0, stop=4, step=1)
>>> print(df.shape)
(4, 2)
```

Creating a DataFrame from a List of Dictionaries

A *DataFrame* can also be created from a dictionary of rows, with a mapping between each column name and the row's associated value.

```
>>> d = [{"a": 1, "b": 2}, {"a": 2, "c": 3}]
>>> df = pd.DataFrame(d)
>>> print(df)
   a    b    c
0  1  2.0  NaN
1  2  NaN  3.0
>>> print(df.columns)
Index(['a', 'b', 'c'], dtype='object')
>>> print(df.index)
RangeIndex(start=0, stop=2, step=1)
>>> print(df.shape)
(2, 3)
```

Getting the Size of a DataFrame

```
>>> # Using shape attribute
>>> n_row = penguins.shape[0]
>>> n_col = penguins.shape[1]
>>> print("No. rows:", n_row, ", No. columns:",
    n_col)
No. rows: 344 , No. columns: 9
>>> # Using len() function:
>>> n_row = len(penguins)
>>> cols = penguins.columns
>>> print(cols)
Index(['rowid', 'species', 'island', '
    bill_length_mm', ... dtype='object')
>>> n_col = len(cols)
>>> print("No. rows:", n_row, ", No. columns:",
    n_col)
No. rows: 344 , No. columns: 9
```

Viewing Data with `head()`

View the first three rows:

```
>>> print(penguins.head(3))
   rowid species  ...     sex  year
0      1  Adelie  ...    male  2007
1      2  Adelie  ...  female  2007
2      3  Adelie  ...  female  2007

[3 rows x 9 columns]
```

Viewing Data with `tail()`

View the last three rows:

```
>>> print(penguins.tail(3))
     rowid    species  ...     sex  year
341    342  Chinstrap  ...    male  2009
342    343  Chinstrap  ...    male  2009
343    344  Chinstrap  ...  female  2009

[3 rows x 9 columns]
```

Summary Statistics

```
>>> print(penguins[['bill_depth_mm', 'flipper_mm',
    'body_mass_g']].describe())
       bill_depth_mm  ...  body_mass_g
count     342.000000  ...   342.000000
mean       17.151170  ...  4201.754386
std         1.974793  ...   801.954536
min        13.100000  ...  2700.000000
25%        15.600000  ...  3550.000000
50%        17.300000  ...  4050.000000
75%        18.700000  ...  4750.000000
max        21.500000  ...  6300.000000

[8 rows x 3 columns]
```

Operations for Data Extraction

Here are the common operations to extract data from a DataFrame:

Operation	Syntax	Result
Select column	`df[col]`	Series
Select multiple columns	`df[[col1, col2]]`	DataFrame
Select row by integer location	`df.iloc[loc]`	Series
Slice rows	`df[5:10]`	DataFrame
Select rows by boolean vector	`df[bool_vec]`	DataFrame

Data Selection and Filtering

- Using `.loc` to select data by labels:

```
print(penguins.loc[0, 'species'])  # Example to access the first row's species
```

- Using `.iloc` to select data by positions:

```
print(penguins.iloc[0, :])  # Example to access the first row
```

Accessing Rows by Index

```
>>> print(penguins.iloc[10:15])
    rowid species  ...     sex  year
10     11  Adelie  ...     NaN  2007
11     12  Adelie  ...     NaN  2007
12     13  Adelie  ...  female  2007
13     14  Adelie  ...    male  2007
14     15  Adelie  ...    male  2007

[5 rows x 9 columns]
```

Accessing Rows by Condition

```
>>> penguins[penguins['body_mass_g'] > 4000].
    head()
     rowid species  ...   sex  year
7        8  Adelie  ...  male  2007
9       10  Adelie  ...   NaN  2007
14      15  Adelie  ...  male  2007
17      18  Adelie  ...  male  2007
19      20  Adelie  ...  male  2007

[5 rows x 9 columns]
```

Accessing Columns by Name

We can access columns with the `[]` notation.

```
>>> print(penguins['species'].head(3))
0    Adelie
1    Adelie
2    Adelie
Name: species, dtype: object
```

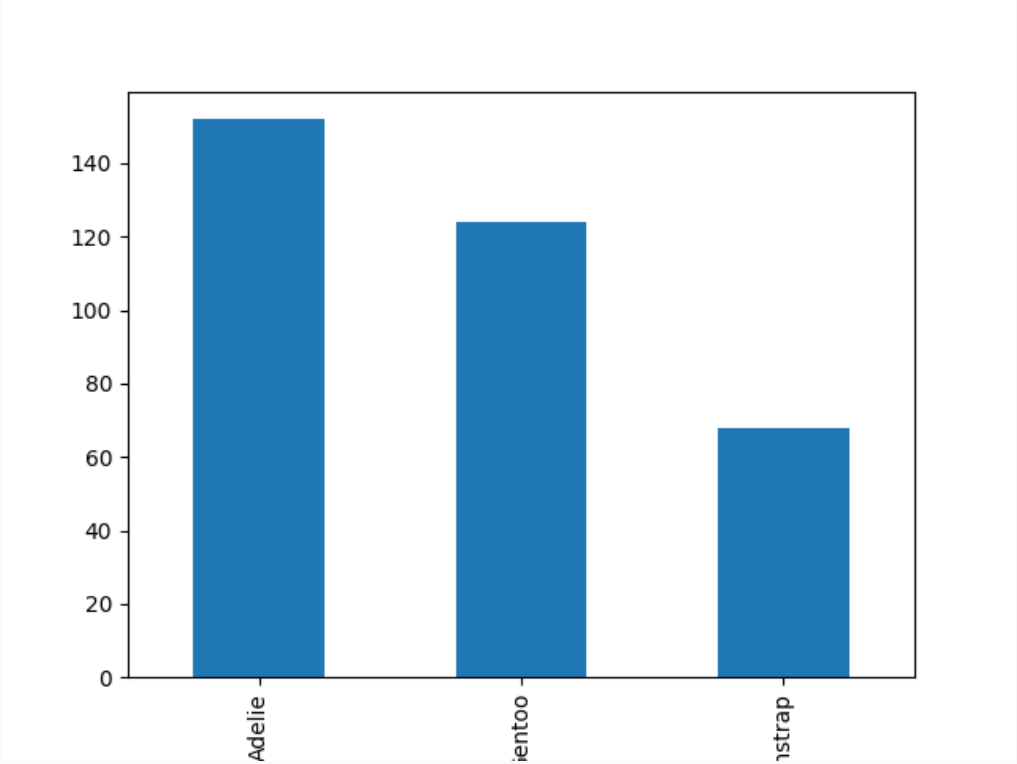
You can treat column names as attributes of the `DataFrame` if they conform to Python variable name conventions.

```
>>> print(penguins.species.head(3))
0    Adelie
1    Adelie
2    Adelie
Name: species, dtype: object
```

Accessing Multiple Columns

```
>>> subset = penguins[['species', 'island']]
>>> print(subset.head())
  species     island
0  Adelie  Torgersen
1  Adelie  Torgersen
2  Adelie  Torgersen
3  Adelie  Torgersen
4  Adelie  Torgersen
```

A Simple Bar Plot




```
import matplotlib.pyplot as plt
counts = penguins.species.value_counts()
counts.plot(kind='bar')
plt.show()
```

Data Cleaning

Handling Missing Values

- Check for missing values:

```
print(penguins.isnull().sum())
```

- Drop missing values:

```
penguins_complete = penguins.dropna()
```

- Fill missing values:

```
penguins_filled = penguins.fillna(0) #
    Example: Fill missing values with 0
```

Other Useful `Dataframe` Methods

- Remove duplicate rows:

```
>>> penguins_deduped = penguins.drop_duplicates()
```

- Rename columns:

```
>>> penguins_renamed = penguins.rename(columns={'
    species': 'penguin_species'})
>>> print(penguins_renamed.columns)
Index(['rowid', 'penguin_species', 'island', '
    bill_length_mm', 'bill_depth_mm',
       'flipper_length_mm', 'body_mass_g', 'sex', '
           year'],
      dtype='object')
```

Subsetting by `notnull()` and `isnull()`

- *`notnull()`*: True for non-null, False for null values.
- *`isnull()`*: True for null values, False for non-null values.

```
>>> non_null_bill = penguins[penguins.bill_depth_mm.
    notnull()]
>>> print(non_null_bill[['rowid', 'bill_depth_mm']].head
    (3))
   rowid  bill_depth_mm
0      1           18.7
1      2           17.4
2      3           18.0
>>>
>>> null_bill = penguins[penguins.bill_depth_mm.isnull()]
>>> print(null_bill[['rowid', 'bill_depth_mm']].head(3))
     rowid  bill_depth_mm
3        4            NaN
271    272            NaN
```

Data Transformation

Adding and Removing Columns

- Add a new column:

```
df['new_column'] = values
```

- Remove a column:

```
df.drop('column_name', axis=1)
```

Applying Functions

- Apply a function to a column:

```
df['column'].apply(function)
```

Mapping and Replacing Values

- Map values in a column:

```
df['column'].map(mapping_dict)
```

- Replace values in a column:

```
df['column'].replace(to_replace, value)
```

Applying to the Penguin Data

```
>>> def g_to_lbs(g):
...     return g/453.592
...
>>> penguins['body_mass_lb'] = penguins['
    body_mass_g'].apply(g_to_lbs)
>>> penguins[['body_mass_g', 'body_mass_lb']].
    head(3)
    body_mass_g  body_mass_lb
0        3750.0      8.267342
1        3800.0      8.377573
2        3250.0      7.165029
```

Applying to the Penguin Data

```
en_zh_map = {
    Adelie : '阿德利',
    Chinstrap : '帽带',
    Gentoo : '巴布亚'
}
scientific_name_map = {
    'Adelie' : 'Pygoscelis adeliae'
    'Gentoo' : 'Pygoscelis papua'
    'Chinstrap' : 'Pygoscelis antarctica'
}
penguins['species_zh'] = df['species'].map(
    en_zh_map)
penguins['species_sci'] = df['species'].map(
    scientific_name_map)
```

Aggregation and Grouping

Aggregation and Grouping

- Group data by one or more columns:

```
grouped = df.groupby('column')
```

- Aggregate grouped data with various functions:

```
grouped['column'].mean()
grouped['column'].sum()
grouped['column'].count()
```

Applying to the Penguin Data

```
>>> grouped = penguins.groupby('sex')
>>> grouped['flipper_length_mm'].mean()
sex
female    197.363636
male      204.505952
Name: flipper_length_mm, dtype: float64
>>> grouped['body_mass_g'].mean()
sex
female    3862.272727
male      4545.684524
Name: body_mass_g, dtype: float64
```

Merging and Joining DataFrames

Concatenating DataFrames

- Concatenate DataFrames along a particular axis:

```
adelie = pd.read_csv('penguins_adelie.csv')
gentoo = pd.read_csv('penguins_gentoo.csv')
chinstrap = pd.read_csv('penguins_chinstrap.csv')
penguin = pd.concat([adelie, gentoo, chinstrap], axis
    =0)
```

Merging DataFrames

- Merge DataFrames based on a key:

```
penguin_egg = pd.read_csv('penguin_egg.csv')
penguin_full = pd.merge(penguin, penguin_egg, on='
    rowid')
```