DATA1002 Week 7 Tutorial

Monday 15/09/25

Tutorial Outline

- Content revision (Modules), Python Demo
- Content revision (Functions), Menti
- Work on Assignment 1

THE UNIVERSITY OF SYDNEY

Tutor: *Tommy Lu*

Housekeeping

1st hour we'll be revising content.
2nd hour we'll be working on the Assignment.

Let me know if you're not in a group!

Note on engagement.

Group Project Stage 1

Due: 17:00 pm on Sunday at the end of week 8 (Sep 28th)

Content Revision

Modules

CSV Module

- A module for reading, and writing, in csv and other similar physical formats
- Deals with complexities like fields that themselves contain a comma

```
import csv

first_row = True
csv_reader = csv.reader(open('file.csv')) # loop over each row from
the CSV reader for row in csv_reader:
if first_row:
    field_names = row
    # field_names is a list containing fields from the header row
    first_row = False
else:
    # row is a list containing the fields from one line
    # access the fields by index eg row[0] is first field
```

Use of CSV module (1)

```
import csv
with open('file.csv') as f:
    reader = csv.reader(f)
    for row in reader:
        print(row)
```

The with statement avoids situations where errors occur in execution
It closes files properly when block is exited

Use of CSV module (2)

```
import csv
with open('file.csv') as csvfile:
    csv_reader = csv.reader(csvfile,
        delimiter=';', quotechar='|')
    for row in csv_reader:
```

Adjust configuration for different field separator, field quoting, etc

- This utilizes Python concept of optional keyword arguments
- These come after whatever positional arguments are passed, and must be named exactly as documented
- But some can be missing, in which case called function uses default value for that argument

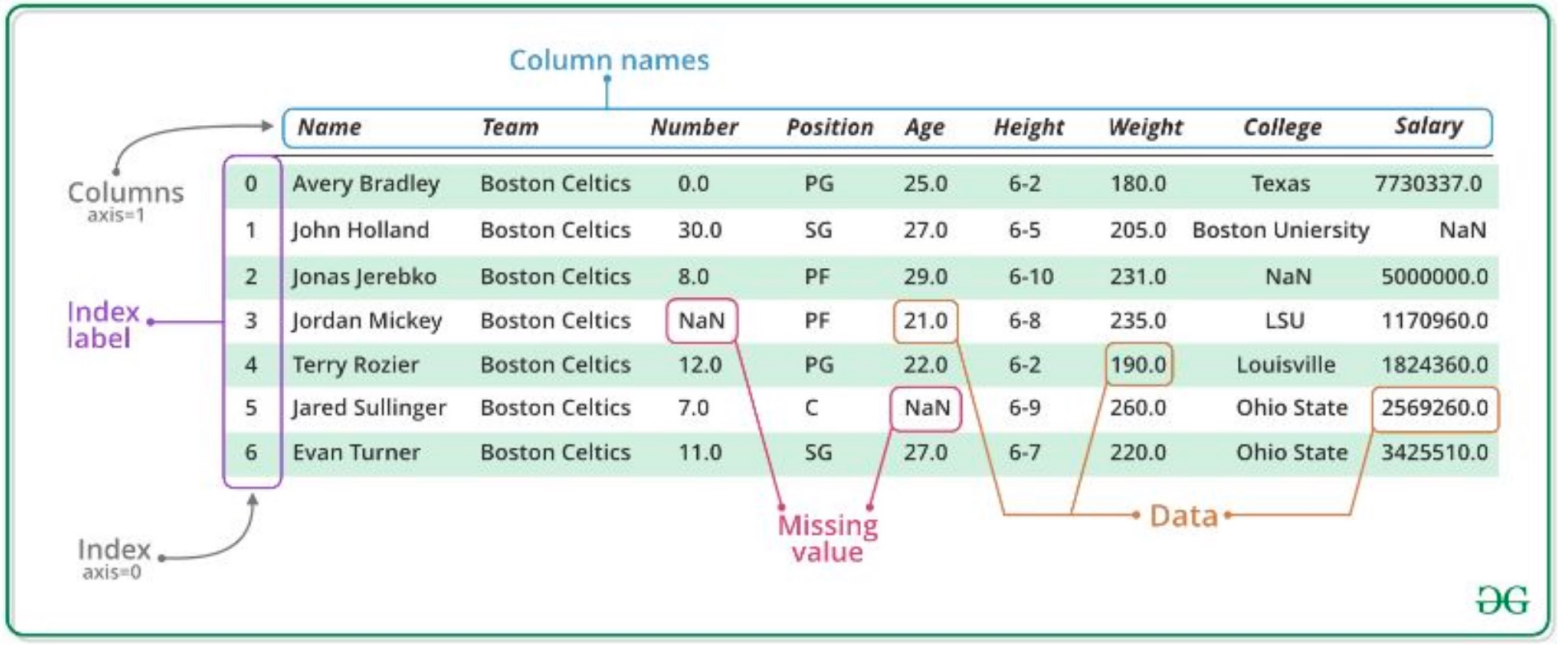
Pandas!!

- An external library which is widely used for coding data science activities in Python
- This offers a huge number of features in many different ways to do the same thing
- Allows you to:
  - Avoid explicit loops
  - Easily do calculations in one line

Dataframe

- Pandas typically stores dataset in an object called a dataframe
  - Think of it as a rectangular array, where rows and columns are named
  - Like a dictionary of dictionaries where the inner dictionaries all use the same keyset, and each inner dictionary has all its values of the same type

Dataframe

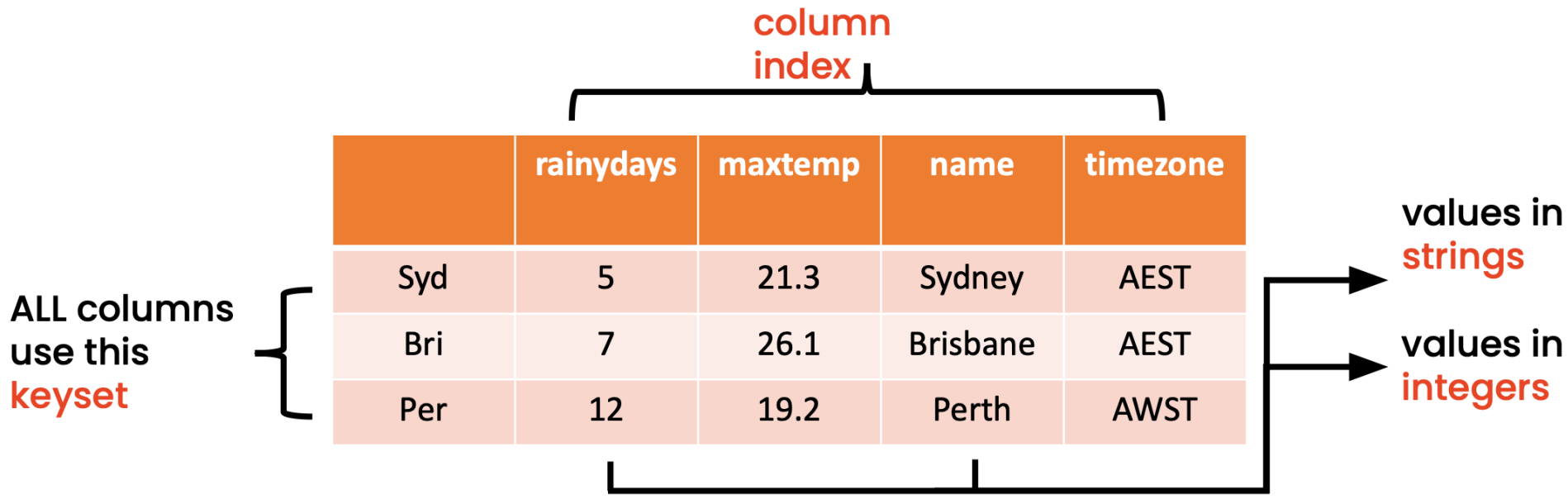


	Name	Team	Number	Position	Age	Height	Weight	College	Salary
0	Avery Bradley	Boston Celtics	0.0	PG	25.0	6-2	180.0	Texas	7730337.0
1	John Holland	Boston Celtics	30.0	SG	27.0	6-5	205.0	Boston Uniersity	NaN
2	Jonas Jerebko	Boston Celtics	8.0	PF	29.0	6-10	231.0	NaN	5000000.0
3	Jordan Mickey	Boston Celtics	NaN	PF	21.0	6-8	235.0	LSU	1170960.0
4	Terry Rozier	Boston Celtics	12.0	PG	22.0	6-2	190.0	Louisville	1824360.0
5	Jared Sullinger	Boston Celtics	7.0	C	NaN	6-9	260.0	Ohio State	2569260.0
6	Evan Turner	Boston Celtics	11.0	SG	27.0	6-7	220.0	Ohio State	3425510.0

Create a Dataframe

```
import pandas as pd
dict = {
"raindays": {"Syd":5, "Bri": 7, "Per":12},
"maxtemp": {"Syd" :21.3,"Bri": 26.1, "Per":
19.2},
"name": {"Syd": "Sydney","Bri": "Brisbane",
"Per": "Perth"},
"timezone": {"Syd": "AEST","Bri": "AEST",
"Per":"AWST"}
}
df = pd.DataFrame(dict)
```

Create a Dataframe




```
dict = {
"raindays": {"Syd":5, "Bri": 7, "Per":12},
"maxtemp": {"Syd" :21.3,"Bri": 26.1, "Per": 19.2},
"name": {"Syd": "Sydney","Bri": "Brisbane", "Per": "Perth"},
"timezone": {"Syd": "AEST","Bri": "AEST", "Per":"AWST"}
}
```

Pandas Terminology

- We refer to either rows or columns as an axis
- The keys or labels used for access are called the index for that axis
  - row index defaults to 0, 1, 2 etc if not explicitly set
  - `list(df.index)` gives a list of row index labels from DataFrame `df`
  - `list(df.columns)` gives a list of column index labels from DataFrame `df`
- The data type of a single column or row is Series
  - all values are of the same type; with many operations available

Accessing an Entry

- Use the value of a single entry from the dataframe, with dictionary-like syntax

```
dataframe[col_name][row_name]
```

  - E.g., `df["raindays"]["Syd"]`

- Also, change an entry by "assignment" syntax

```
dataframe[col_name][row_name] = new_value
```

Filtering

- Boolean comparison
  - e.g., `df["raindays"] < 10` has `True` for row index where the column called raindays has a number less than 10, False elsewhere
  - e.g., `df["raindays"]<df["sunnydays"]` is a Boolean Series that compares two series with the same index structure

- `df[filter]` is a dataframe containing only the rows from df for which the corresponding filter entry is True
  - E.g., `(df[df["raindays"]<10]) ["name"])` gives names from rows in which raindays is less than 10

Data Cleaning

- `df.isnull()` - returns True where df has a missing value
- `df.dropna()` - returns the dataframe without the rows with any missing value
- `df.fillna(x)` - replace missing values in dataframe with `x`
  - If `x` is a dictionary whose keys are column names, the replacement value in each column is chosen appropriately

Duplicates

- `df[column].is_unique()` - gives a Boolean to indicate is there are duplicate values anywhere in that column
- `df.duplicated()` - returns a series of Booleans with the same rowindex as dataframe
  - Use keyword argument `keep` to mark last or all appearances of duplicate rows
- `df.drop_duplicates()` - removes duplicate rows from df except for the first appearance of each

Pivoting Datasets

- You are provided with a csv called *student_scores.csv*.
- This data is provided to you in wide format
- By looking up the Pandas function '**melt**', convert the data into long format

- **EXTENSION:** figure out how to '**unmelt**' your data frame after pivoting it - pivot()

```
Original DataFrame:
      Name  Math  English  Science
0    Ayman    85       92       78
1   Biling    70       88       90
2  Charlie    95       85       85
```

```
Long-format DataFrame:
      Name  Subject  Score
0    Ayman     Math     85
1   Biling     Math     70
2  Charlie     Math     95
3    Ayman  English     92
4   Biling  English     88
5  Charlie  English     85
6    Ayman  Science     78
7   Biling  Science     90
8  Charlie  Science     85
```

Definition and Usage

The `melt()` method reshapes the DataFrame into a long table with one row for each each column.

Syntax

```
dataframe.melt(id_vars, value_vars, var_name, value_name, col_level, ignore_index)
```

Parameters

The `id_vars`, `value_vars`, `var_name`, `value_name`, `col_level`, `ignore_index` parameters are keyword arguments.

Parameter	Value	Description
id_vars	*Tuple* *List* *Array*	Optional, specifies the column, or columns, to use as identifiers
value_vars	*Tuple* *List* *Array*	Optional, specifies columns to unpivot.
var_name	String	Optional, specifies the label of the 'variable' column, default 'variable'
col_level	*Number* *String*	Optional, for MultiIndex DataFrames, specifies the level to melt
ignore_index	`True` `False`	Optional, default True. Specifies whether to ignore the original index or not

Python Exercise

Trying all the methods

Upskilling in Pandas

- You are provided with two csv files called *stocks.csv* and *stocks2.csv*
- Try and apply all the following methods to one, or where necessary, both csv files

- Upload a file format: `read_csv`, `read_excel`, etc.
- Access by row (`.loc`), column (`df['col']`), or entry.
- Filter
- Handle missing values (`dropna`, `fillna`)
- Remove duplicates
- Summarize (`mean`, `min`)
- Group (`groupby`)
- Reshape (`pivot`, `melt`)
- Combine (`concat`, `merge`).

Content Revision

Functions & Errors

Function

- Once functions are defined, this can be called from other parts of the code
- Provides a way to re-use code, rather than repeating it
  - Better for readability and maintainability
- Allows functional abstraction
  - Can have arguments (parameters) that can be different for various calls
- A function can return a value and be used within expressions, or it can be used just as a single statement doing some complicated work

Examples

- def greet(name):
    print("Hello,", name)

What would greet("abc") return?

- def add2(x, y):
    z = x + y
    return z

What would add2(3, 5) return?

Handling Errors

Example

In the example below, we have a program that initialises `x` and `y`, calculates `x / y`, prints the result then ends with a finishing message. However, the statement `result = x / y` will raise an exception.

PYTHON

```python
x = 1
y = 0
print(f'Dividing {x} by {y}!')
result = x / y # this statement may raise an exception
print('Finished program!')
```

Try & Except

When we use a `try` and `except`, you're often only concerned about a small section of code to handle the exception. It's good practise to not have your entire program inside a `try` and `except`, but rather only the code that needs to be in it. It improves the code's readability and makes it easier to identify what part of the code you're handling exceptions for.

Run PYTHON

```python
x = 1
y = 0
print(f'Dividing {x} by {y}!')
try:
    result = x / y
except ZeroDivisionError:
    print('Wait... we cannot divide by zero!')
print('Finished program!')
```

Printing that the program is finished can go after the `try` and `except` block. With this structure, it's clearer what code we are accounting for the `ZeroDivisionError` exception.

Try & Except

When we use a `try` and `except`, you're often only concerned about a small section of code to handle the exception. It's good practise to not have your entire program inside a `try` and `except`, but rather only the code that needs to be in it. It improves the code's readability and makes it easier to identify what part of the code you're handling exceptions for.

Run PYTHON

```
1 x = 1
2 y = 0
3 print(f'Dividing {x} by {y}!')
4 try:
5     result = x / y
6 except ZeroDivisionError:
7     print('Wait... we cannot divide by zero!')
8 print('Finished program!')
```

Printing that the program is finished can go after the `try` and `except` block. With this structure, it's clearer what code we are accounting for the `ZeroDivisionError` exception.

Common Error Types

KeyError

ValueError

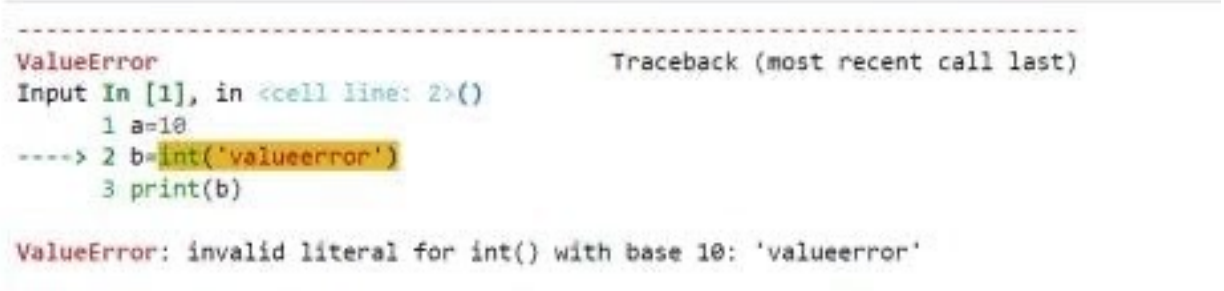
IndexError

TypeError

Common Error Types

KeyError

ValueError



IndexError

TypeError

Common Error Types

KeyError

ValueError

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [1], in <cell line: 2>()
      1 a=10
----> 2 b=int('valueerror')
      3 print(b)

ValueError: invalid literal for int() with base 10: 'valueerror'
```

IndexError

TypeError

```
5 + '4'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [3], in <module>
----> 1 5 + '4'

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Common Error Types

ValueError

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [1], in <cell line: 2>()
      1 a=10
----> 2 b=int('valueerror')
      3 print(b)

ValueError: invalid literal for int() with base 10: 'valueerror'
```

TypeError

```
5 + '4'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [3], in <module>
----> 1 5 + '4'

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

KeyError

```
vehicle = {
  "model": "RE Hunter 350",
  "color": "Black",
}

print(vehicle["speed"])
```

KeyError: 'speed'

IndexError

Common Error Types

ValueError

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [1], in <cell line: 2>()
      1 a=10
----> 2 b=int('valueerror')
      3 print(b)

ValueError: invalid literal for int() with base 10: 'valueerror'
```

TypeError

```
5 + '4'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [3], in <module>
----> 1 5 + '4'

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

KeyError

```
vehicle = {
  "model": "RE Hunter 350",
  "color": "Black",
}

print(vehicle["speed"])
```

KeyError: 'speed'

IndexError

main.py

```
1  # index    0       1       2       3       4
2  color = ['red', 'green', 'blue', 'black', 'white']
3
4  print(color[5])
```

```
Traceback (most recent call last):
  File "main.py", line 4, in <module>
    print(color[5])
IndexError: list index out of range


...Program finished with exit code 1
Press ENTER to exit console.
```

Menti

Pandas & Functions/Errors

Menti

Go to

www.menti.com

Enter the code

2650 9838

Or use QR code

Let’s Take a Short Break!

Lab Activities

Working on Assignment 1

Activity

By Week 7, your group should begin focusing on producing summary statistics for each dataset and starting the report writing process. This is a key component of Assignment 1 Stage 1 for DATA1002 and Assignment 1 Stage 2 for DATA1902.

For DATA1002, your task involves deciding what information should be summarized and explaining why that information is important for your analysis. You may choose to summarize multiple attributes within the dataset if they provide meaningful insights or help identify patterns or trends relevant to your research question.

Exam-Style Questions

Question 1:

How can modular programming practices, such as using functions and libraries, improve the scalability and maintainability of data science projects?

Provide examples.

Exam-Style Questions

Modular programming practices, such as using **functions and libraries**, improve the **scalability and maintainability** of data science projects by promoting code reusability and organisation. `Functions` allow for **encapsulating specific tasks**, making the codebase easier to understand and modify.

For example: a function to clean a dataset can be reused across multiple projects, ensuring consistency and reducing redundancy.

Libraries like `pandas` and `numpy` provide tested and optimised implementations of common tasks, further enhancing scalability. By breaking down complex projects into smaller, manageable modules, data scientists can update or replace individual components without affecting the entire system. This modular approach not only streamlines development but also **facilitates collaboration**, as different team members can **work on separate modules concurrently**.

Exam-Style Questions

Question 2 [DATA1002]:

Discuss the challenges and solutions for handling missing values and duplicates in a dataset. How do Python libraries like pandas facilitate these tasks in a data science project?

Exam-Style Questions

Handling missing values and duplicates is a common challenge in data science projects. Missing values can **distort analysis and lead to incorrect conclusions**, **while duplicates can inflate or skew results**. Python libraries like `pandas` offer robust solutions for these issues.

Functions such as `dropna()` and `fillna()` allow for removing or imputing missing values with specified constants or statistical measures (e.g., mean or median). The `duplicated()` function identifies duplicate rows, which can then be removed using `drop_duplicates()`.

Additionally, pandas provides flexible options for handling missing values during data import, such as treating specific characters as NaN. These capabilities streamline the data cleaning process, ensuring that the dataset is accurate and reliable for further analysis.

That’s it folks!

Remaining Ed Lessons, Questions, Assignment etc.