Practical 05: Data Exploration

Dr. Todd Jones

t.r.jones@reading.ac.uk

Department of Computer Science

Follow the instructions to complete each of these tasks.

This set of exercises focuses on writing basic Python code and exploring the use of `pandas` and `numpy` to manipulate and pre-process data in advance of data visualisation and applying machine learning models.

This work is not assessed but will help you gain practical experience for the coursework.

Configuration

In [ ]:
```
import ???

NaN = np.nan

# Configure sample DataFrame
dogs = pd.DataFrame({
            'Name': ['Tom', 'Lupita', 'Lupita', 'Olivia', NaN, 'Marcel', 'Choco'],
            'Age': [5, 4, 4, 1, 10, NaN, NaN],
            'Breed': [NaN, 'Corgi', 'Corgi', 'Husky', 'Poodle', 'Bulldog', NaN],
            'Grade': ['a', 'b', 'b', 'c', 'd', 'e', 'f']})
dogs
```

Detecting Duplicated Rows

Look up the `duplicated()` method. Use this to **display** an indication of any duplicated rows.

https://pandas.pydata.org/pandas-docs/version/2.2.2/reference/api/pandas.DataFrame.duplicated.html

In [ ]:
```

```

Use the `.sum()` method in conjuction with the previous result to display a count of the number of duplicated rows.

In [ ]:
```

```

Dropping Duplicated Rows

Look up the `drop_duplicates()` method. Use this display a corrected version of the `DataFrame`

Does this method alter the original `DataFrame`?

- If so, continue on.
- If not, save the result before continuing.

In [ ]:

Dropping Irrelevant Columns

It has come to our attention that the data under `Grade` is meaningless.

Look up the `drop()` method. Use this to remove the irrelevant column from the data.

https://pandas.pydata.org/pandas-docs/version/2.2.2/reference/api/pandas.DataFrame.drop.html

In [ ]:

Identifying and handling missing values

`pandas` represents missing numeric values using **NaN** (Not a Number). There are several methods to detect and drop missing values from a dataframe.

- `isnull()`
  - generates a data structure of booleans, indicating whether each value is missing
- `dropna()`
  - drops some rows or columns if they contain missing data
- `fillna()`
  - fills in missing values

Display a boolean dataset showing which entries are missing.

In [ ]:

Display a count of the number of missing values in each column.

In [ ]:

Display what data would remain if we were to drop any row with a missing value.

In [ ]:

Fill in the missing values and save the result to a variable.

- Where a name is missing, enter: `'Unknown'`
- Where an age is missing, enter the mean of the known ages.
- Where a breed is missing, enter: `'Poodle'`

In [ ]:

Philadelphia Bike Share Live Data

Complete the code below to load a JSON live feed for a Philadelphia bike share program into a `pandas DataFrame`. It may help to look at the JSON data in a visual inspector. One way of doing this is to open the given URL in a browser.

Note: As this is live data, the results of the following analyses will vary based on when they are performed.

(I've sometimes had to add in some header data to the request, as the server might reject requests that do not include a user agent string.)

You can use the `pandas` function `pd.json_normalize`, but you need to pass it a suitable part of the JSON data. https://pandas.pydata.org/pandas-docs/version/2.2.2/reference/api/pandas.json_normalize.html

The `indego_bikes_data` object returned by `requests.get()` can be converted to a Python data structure using the `json()` method.

Once you have loaded the data, look at the `head` of the data frame, and **list** all of the columns.

Have you imported `pandas` already?

What does each row represent?

In [ ]:
```
import requests
indego_bikes_url = ("https://www.rideindego.com/stations/json/")
headers = {'User-Agent': "Mozilla/5.0 (Macintosh; Intel Mac OS X 10.13; rv:63.0) Gecko/20100101
indego_bikes_data = requests.get(indego_bikes_url)#,headers=headers)
```

In [ ]:
```
# To display the data you can use the following line to convert the returned
# data to JSON format
indego_bikes_data.json()
```

In [ ]:

Run `info()` and `describe()` on the data to see what you can learn.

In [ ]:

Display entries where the number of available docks is greater than 10.

In [ ]:

Display the average number of available bikes for stations north of 30.96 degrees latitude and east of -75.16 degrees longitude. **Round** the result to the nearest whole number.

In [ ]:

Display the Zip Code (American version of post code) that has the greatest number of stations.

How can you count stations in each Zip Code?

In [ ]:

Use `pandas` to **count** the total number of **available docks** in each Zip Code, producing a `Series` of Zip Codes and available dock counts.

You can use the `pandas` method `sum()` on a grouby object to add the all values in a particular group.

In [ ]:

Using `pandas`, find the minimum and maximum number of **available bikes** and the **range between these two values** within each Zip Code.

- Write your own function to calculate the **range**.

Use the `.agg()` method and **include your own function** as one of the arguments.

Save the resulting `DataFrame`, with all three values for each Zip Code, to a CSV file.

In [ ]:

Write Python code using `pandas` to determine the Zip Code with the **highest median** of *docks available*.

Print the solution in an f-string to say something like:

`Zip Code 19149 has the largest median: 24.`

Help: https://pandas.pydata.org/pandas-docs/version/2.2.2/reference/api/pandas.DataFrame.sort_values.html

In [ ]:

Iris Flower Data

Load the `./data/iris.csv` flower data from the practical archive, and **display** the first 5 rows.

In [ ]:

Add two extra columns to the `DataFrame` giving the ratios of sepal length to width and petal length to width.

In [ ]:

Calculate and **display** the means of the ratios of sepal length over width and petal length over width.

In [ ]:

Perform some exploration on the dataset.

- **How many** classes (species) are there, and what are the species names?
- What is the distribution of the classes? (How many in each?)
- What are the characteristics of the data in general and per class?

You can use methods like `unique()`, `value_counts()`, and `describe()`.

In [ ]:
```
# How many classes (species) are there,
# and what are the species names?
```

In [ ]:
```
# What is the distribution of the classes?
```

In [ ]:
```
# What are the characteristics of the data in general?
iris.describe()
```

In [ ]:
```
# What are the characteristics of the data per class?
```

In [ ]:
```
# What are the characteristics of the data per class? (again)
```

In [ ]:
```
# What are the characteristics of the data per class? (again)
```

For flowers with a petal ratio greater than 3, **report** the mean sepal ratio for each species.

In [ ]:

Working with NumPy

Have you imported `numpy`?

Helpful `numpy` routines for this section:

- `np.all()`
- `np.isfinite()`
- `np.identity()`
- `np.random.normal()` (slightly different from `np.random.randn`)

Write code to test whether these `numpy` arrays contain 0.

```
np.array([1, 2, 3, 4])
np.array([1, 2, 3, 4, 0])
```

In [ ]:

Test whether the array below contains any NaNs or infite numbers.

```
np.array([1, 0, np.nan, np.inf])
```

In [ ]:

Create and display a 3x3 identity matrix (i.e., the diagonal elements are 1, the rest are 0).

In [ ]:

Write code to generate an array of 10 random numbers from a normal distribution.

In [ ]:

Write code to compute the coordinates for points on a cosine curve from zero to 3π. Calculate a value every $0.2rad$.

Store the inputs to the function in `x` and the result in `y`. Then, create a `pandas` `DataFrame` to hold both of these. Display the contents of the `DataFrame`.

In [ ]: