Part 2 — Workshop 2

TECH2: Introduction to Programming, Data, and Information Technology

Richard Foltyn
Norwegian School of Economics (NHH)

October 11, 2024

See GitHub repository for notebooks and data:

https://github.com/richardfoltyn/TECH2-H24

1 Exercise: Data cleaning

Before doing actual data analysis, we usually first need to clean the data. This might involve steps such as dealing with missing values and encoding categorical variables as integers.

Load the Titanic data set in `titanic.csv` and perform the following tasks:

1. Report the number of observations with missing `Age`, for example using `isna()`.
2. Compute the average age in the data set. Use the following approaches and compare your results:
    1. Use the `mean()` method.
    2. Convert the `Age` column to a NumPy array using `to_numpy()`. Experiment with NumPy's `np.mean()` and `np.nanmean()` to see if you obtain the same results.
3. Replace the all missing ages with the mean age you computed above, rounded to the nearest integer. Note that in "real" applications, replacing missing values with sample means is usually not a good idea.
4. Convert this updated `Age` column to integer type using `astype()`.
5. Generate a new column `Female` which takes on the value one if `Sex` is equal to `"female"` and zero otherwise. This is called an *indicator* or *dummy* variable, and is preferrable to storing such categorical data as strings. Delete the original column `Sex`.
6. Save your cleaned data set as `titanic-clean.csv` using `to_csv()` with `,` as the field separator. Tell `to_csv()` to *not* write the `DataFrame` index to the CSV file as it's not needed in this example.

Solution.

[1]:
```
# Path to data directory
DATA_PATH = '/home/richard/repos/teaching/TECH2-H24/data'

# Alternatively, load data directly from GitHub
# DATA_PATH = 'https://raw.githubusercontent.com/richardfoltyn/TECH2-H24/main/data'
```

[2]:
```
import pandas as pd

# Path to Titanic CSV file
fn = f'{DATA_PATH}/titanic.csv'

df = pd.read_csv(fn)
```

Number of missing values

The number of non-missing values can be displayed using the `info()` method. Alternatively, we can count the number of missing values directly by summing the return values of `isna()`.

[3]:
```
# Display missing counts for each column
df.info(show_counts=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 10 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object 
 4   Sex          891 non-null    object 
 5   Age          714 non-null    float64
 6   Ticket       891 non-null    object 
 7   Fare         891 non-null    float64
 8   Cabin        204 non-null    object 
 9   Embarked     889 non-null    object 
dtypes: float64(2), int64(3), object(5)
memory usage: 69.7+ KB
```

[4]:
```
# Alternative way to get the number of missing values:
df['Age'].isna().sum()
```

[4]:
```
177
```

Compute mean age

We compute the mean age using the three different methods. As you can see, `np.mean()` cannot deal with missing values and returns `NaN` (*"not a number"*).

[5]:
```
import numpy as np

# Compute mean age using the DataFrame.mean() method
mean_age = df['Age'].mean()

# Convert Age column to NumPy array
age_array = df['Age'].to_numpy()

# Compute mean using np.mean()
mean_age_np = np.mean(age_array)

# Compute mean using np.nanmean()
mean_age_np_nan = np.nanmean(age_array)

print(f'Mean age using pandas:       {mean_age:.3f}')
print(f'Mean age using np.mean():    {mean_age_np:.3f}')
print(f'Mean age using np.nanmean(): {mean_age_np_nan:.3f}')
```

```
Mean age using pandas:       29.699
Mean age using np.mean():    nan
Mean age using np.nanmean(): 29.699
```

Replace missing values

There are several ways to replace missing values. First, we can "manually" identify these using boolean indexing and assign a new value to such observations.

[6]:
```
# Round average age
mean_age = np.round(mean_age)

# boolean arrays to select missing observations
is_missing = df['Age'].isna()

# Update missing observations with rounded mean age
df.loc[is_missing, 'Age'] = mean_age
```

There is also the convenience routine `fillna()` which automates this step. To illustrate, we need to reload the original data as we have just replaced all missing values.

[7]:
```
# Re-load data to get the original missing values
df = pd.read_csv(fn)

df['Age'] = df['Age'].fillna(value=mean_age)
```

Convert age column to integer type

Since age is usually recorded as an integer, there is no reason to store it as a `float` once we have dealt with the missing values.

[8]:
```
df['Age'] = df['Age'].astype(int)
```

Generate Female indicator

An indicator variable can be obtained as a result of a logical operation (`==`, `!=`, etc.). This value contains `True` or `False` values, which we can convert to 1 or 0 by changing the data type to integer.

[9]:
```
# Generate boolean array (True/False) whether passenger is female
is_female = (df['Sex'] == 'female')

# Add Female dummy variable, converted to integer
df['Female'] = is_female.astype(int)

# Delete original Sex column, no longer needed
del df['Sex']

# Alternatively, you can use
# df = df.drop(columns=['Sex'])
```

Save cleaned file

We can use `info()` again to confirm that `Age` has no missing values and all columns are of the desired data type:

[10]:
```
df.info(show_counts=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 10 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
```

```
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Age          891 non-null    int64
 5   Ticket       891 non-null    object
 6   Fare         891 non-null    float64
 7   Cabin        204 non-null    object
 8   Embarked     889 non-null    object
 9   Female       891 non-null    int64
dtypes: float64(1), int64(5), object(4)
memory usage: 69.7+ KB
```

[11]:
```
# Save cleaned file
fn_clean = f'{DATA_PATH}/titanic-cleaned.csv'

df.to_csv(fn_clean, sep=',', index=False)
```

2 Exercise: Working with strings

Most of the data we deal with contain strings, i.e., text data (names, addresses, etc.). Often, such data is not in the format needed for analysis, and we have to perform additional string manipulation to extract the exact data we need. This can be achieved using the pandas string methods.

To illustrate, we use the Titanic data set for this exercise.

1. Load the Titanic data and restrict the sample to men. (This simplifies the task. Women in this data set have much more complicated names as they contain both their husband's and their maiden name)
2. Print the first five observations of the `Name` column. As you can see, the data is stored in the format *"Last name, Title First name"* where title is something like Mr., Rev., etc.
3. Split the `Name` column by `,` to extract the last name and the remainder as separate columns. You can achieve this using the `partition()` string method.
4. Split the remainder (containing the title and first name) using the space character `" "` as separator to obtain individual columns for the title and the first name.
5. Store the three data series in the original `DataFrame` (using the column names `FirstName`, `LastName` and `Title`) and delete the `Name` column which is no longer needed.
6. Finally, extract the ship deck from the values in `Cabin`. The ship deck is the first character in the string stored in `Cabin` (A, B, C, ...). You extract the first character using the `get()` string method. Store the result in the column `Deck`.

Hint: Pandas's string methods can be accessed using the `.str` attribute. For example, to partition values in the column `Name`, you need to use

```
df['Name'].str.partition()
```

Solution.

[12]:
```
# Path to data directory
DATA_PATH = '/home/richard/repos/teaching/TECH2-H24/data'

# Alternatively, load data directly from GitHub
# DATA_PATH = 'https://raw.githubusercontent.com/richardfoltyn/TECH2-H24/main/data'
```

Import data and restrict to male sub-sample

[13]:
```
import pandas as pd

# Path to Titanic CSV file
fn = f'{DATA_PATH}/titanic.csv'

df = pd.read_csv(fn)
```

We restrict the sample either with boolean indexing of with the `query()` method.

[14]:
```
# Restrict sample to men
df = df.loc[df['Sex'] == 'male'].copy()

# Alternatively, we can do this with a query()
df = df.query('Sex == "male"')
```

Inspect the `Name` column

[15]:
```
# Print first 10 Name observations
df['Name'].head(10)
```

[15]:
```
0              Braund, Mr. Owen Harris
4             Allen, Mr. William Henry
5                     Moran, Mr. James
6              McCarthy, Mr. Timothy J
7       Palsson, Master Gosta Leonard
12    Saundercock, Mr. William Henry
13       Andersson, Mr. Anders Johan
16                Rice, Master Eugene
17      Williams, Mr. Charles Eugene
20               Fynney, Mr. Joseph J
Name: Name, dtype: object
```

Split into last name and remainder

Note that `partition()` returns *three* columns, the second on containing the separator you specified. This second column can be ignored.

[16]:
```
# Split names by comma, create DataFrame with a column for each token
names = df['Name'].str.partition(',')

# Print first 5 rows or resulting DataFrame
names.head(5)
```

[16]:
```
          0  1                      2
0    Braund  ,        Mr. Owen Harris
4     Allen  ,      Mr. William Henry
5     Moran  ,               Mr. James
6  McCarthy  ,           Mr. Timothy J
7   Palsson  ,   Master Gosta Leonard
```

[17]:
```
# Extract last name stored in 1st column, strip any remaining white space
last_name = names[0].str.strip()

# Print first 5 observations
last_name.head(5)
```

[17]:
```
0      Braund
4       Allen
5       Moran
6    McCarthy
7     Palsson
Name: 0, dtype: object
```

Split title and first name

[18]:
```
# Title and first name (potentially multiple) are separated by space
title_first = names[2].str.strip().str.partition(' ')

title_first.head(5)
```

[18]:
```
         0  1              2
0     Mr.       Owen Harris
4     Mr.     William Henry
5     Mr.             James
6     Mr.         Timothy J
7  Master     Gosta Leonard
```

[19]:
```
# Extract title from 1st column, strip any remaining white space
title = title_first[0].str.strip()
title.head(5)
```

[19]:
```
0       Mr.
4       Mr.
5       Mr.
6       Mr.
7    Master
Name: 0, dtype: object
```

[20]:
```
# Extract first name(s) from 3rd column, strip any remaining white space
first_name = title_first[2].str.strip()

# Print first 5 observations
first_name.head(5)
```

[20]:
```
0      Owen Harris
4    William Henry
5            James
6        Timothy J
7    Gosta Leonard
Name: 2, dtype: object
```

Store name components in original DataFrame

[21]:
```
# Merge all name components back into original DataFrame
df['FirstName'] = first_name
df['LastName'] = last_name
df['Title'] = title

# Delete Name column
del df['Name']
```

[22]:
```
df.head(5)
```

[22]:
```
   PassengerId  Survived  Pclass   Sex   Age     Ticket     Fare Cabin  \
0            1         0       3  male  22.0  A/5 21171   7.2500   NaN
4            5         0       3  male  35.0     373450   8.0500   NaN
5            6         0       3  male   NaN     330877   8.4583   NaN
6            7         0       1  male  54.0      17463  51.8625   E46
7            8         0       3  male   2.0     349909  21.0750   NaN

  Embarked      FirstName  LastName   Title
0        S    Owen Harris    Braund     Mr.
4        S  William Henry     Allen     Mr.
5        Q          James     Moran     Mr.
6        S      Timothy J  McCarthy     Mr.
7        S  Gosta Leonard   Palsson  Master
```

Extract deck

We can use the `get()` string method to extract the first element of the cabin string (if present). Note that observations with a missing value for `Cabin` will also be assigned a missing value for `Deck`.

[23]:
```
df['Deck'] = df['Cabin'].str.strip().str.get(0)
```

[24]:
```
# Print histogram of the number of cabins by deck
df['Deck'].value_counts().sort_index()
```

[24]:
```
Deck
A    14
B    20
C    32
D    15
E    17
F     8
T     1
Name: count, dtype: int64
```