M06-HW-1

July 26, 2023

1 Metadata

```
Course:  DS 5100
Module:  06 Pandas
Topic:   HW Myocardial Infarction Analytics with Pandas
Author:  R.C. Alvarado (adapted)
Date:    7 July 2023
```

2 Student Info

- Name: Lindley Slipetz
- Net UD: ddj6tu
- URL of this file in GitHub: https://github.com/sliplr19/DS5100-ddj6tu/blob/main/lessons/M06/M06-HW-1.ipynb

3 Instructions

In your **private course repo on Rivanna**, use this Jupyter notebook and the data file described to write code that performs the tasks below.

Save your notebook in the `M06` directory.

Remember to add and commit these files to your repo.

Then push your commits to your repo on GitHib.

Be sure to fill out the **Student Info** block above.

To submit your homework, save the notebook as a PDF and upload it to GradeScope, following the instructions.

TOTAL POINTS: 12

4 Overview

In this homework, you will be working with the Myocardial Infarction (MI) Complications Data Set housed at UCI.

A myocardial infarction is commonly called a heart attack.

You may Read about the dataset in the Data Description File (DDF).

You will work with some of the columns (aka features).

A subset of these could be predictors in an ML model, while others could be outcome variables.

The section **Attribute Information** in the DDF provides details.

5 Setting Up

[1]:
```
import pandas as pd
import numpy as np
```

6 Prepare the Data

Read in the dataset from the UCI Machine Learning Repository.

Use Pandas' `read_csv()` function, giving the path to the dataset as an argument.

There is no header in this data, so pass a second argument `header=None`.

[2]:
```
path_to_data = "http://archive.ics.uci.edu/ml/machine-learning-databases/00579/
 ↪MI.data"
```

6.1 Task 1

(1 PT)

Import the data into a datafram and then print the number of records in the dataset

[3]:
```
dat = pd.read_csv(path_to_data, header = None)
dat.shape[0]
```

[3]: 1700

6.2 Task 2

(1 PT)

Show the first three records in the dataset

[4]:
```
dat.iloc[:3,]
```

[4]:

	0	1	2	3	4	5	6	7	8	9	...	114	115	116	117	118	119	120	\
0	1	77	1	2	1	1	2	?	3	0	...	0	0	0	0	0	0	0	
1	2	55	1	1	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
2	3	52	1	0	0	0	2	?	2	0	...	0	0	0	0	0	0	0	

	121	122	123
0	0	0	0
1	0	0	0
2	0	0	0

```
[3 rows x 124 columns]
```

7 Working with AGE

The second column contains patient age.

If your dataframe is named `df`, you can reference the column with: `df[1]`.

Generally the field names will be strings and you can use `df['age']` to access field `age`, as an example).

7.1 Task 3

(1 PT)

One complication: missing values are filled with `?` which will cause problems (e.g., stats can't be computed easily).

Count the number of records in `df[1]` containing `?`.

[5]:
```
print(dat[1].value_counts()['?'])
```

```
8
```

7.2 Task 4

(1 PT)

Replace `'?'` with `np.nan` in the age column.

[6]:
```
replace_map = {'?': np.nan}
dat[1] = dat[1].replace(replace_map)
```

7.3 Task 5

(1 PT)

Print the number of records containing `np.nan` in the column `df[1]` of your dataframe.

[8]:
```
dat[1].isna().sum()
```

[8]:
```
8
```

8 Another complication

Another complication: the age data is saved as strings, and there are the null values.

Here's an example:

```
# inspect first element
df[1].iloc[0]
```

```
'77'

# check the column type
df[1].dtype
```

```
dtype('O')
```

To convert the column to numeric, we can use `apply()` with a lambda function.

If the type is string, we cast to numeric, e.g. `float` or `int`, otherwise it's null and we leave things alone.

`isinstance(x, str)` checks if `x` is a string, returning a bool.

Review this code for understanding:

```
df[1] = df[1].apply(lambda x: float(x) if isinstance(x, str) else x)
```

9 Task 6

(1 PT)

Run the lambda function above, then show the data type of `age` is no longer string type.

[9]:
```
dat[1] = dat[1].apply(lambda x: float(x) if isinstance(x, str) else x)
dat[1].dtype
```

[9]:
```
dtype('float64')
```

10 Task 7

(1 PT)

Compute the median age.

[11]:
```
dat[1].median()
```

[11]:
```
63.0
```

11 Working with GENDER

The third column contains patient gender.

Again, since indexing starts at zero, you'll reference `df[2]`.

12 Task 8

(1 PT)

Print the frequency AND percentage of each gender.

Hint: The function you'll use to compute frequencies will take an argument to compute normalized values, which may be converted to percentages.

[14]:
```
dat[2].value_counts()
```

[14]:
```
1    1065
0     635
Name: 2, dtype: int64
```

[15]:
```
dat[2].value_counts(normalize=True).mul(100).round(1)
```

[15]:
```
1    62.6
0    37.4
Name: 2, dtype: float64
```

13 Working with Essential Hypertension (EH)

Reference this column with `df[8]`.

14 Task 9

(1 PT)

Enter the most frequent value.

[16]:
```
dat[8].mode()
```

[16]:
```
0    2
Name: 8, dtype: object
```

15 Working with Atrial Fibrillation (AFIB)

Reference this column with `df[112]`.

AFIB is one of the complications and outcomes of myocardial infarction.

16 Task 10

(1 PT)

Print the number of AFIB cases.

Note that 1 means there is a case.

[19]:
```
sum(dat[dat[112]==1])
```

[19]:
```
7626
```

17 Combining Age and AFIB

18 Task 11

(1 PT)

Construct a new dataframe `df2` containing only the columns for AGE and AFIB.

Recall that AGE is in `df[1]` and AFIB is in `df[112]`.

Print the shape of this dataframe.

Hint: you can pass a list of column names to the dataframe indexer to get a dataframe with a subset of columns.

[22]:
```
cols = [1,112]
df2 = dat[cols]
```

[22]:

	0	1	2	3	4
1	77.0	55.0	52.0	68.0	60.0
112	0.0	0.0	0.0	0.0	0.0

19 Plotting

We are going to plot AGE and AFIB, so renaming the columns to strings will make our visualization more readable.

You can rename columns using the dataframe `.rename()` method, which takes a dictionary as an argument of the form:

```
{
    current_column_name1: new_column_name1,
    ...
    current_column_nameN: new_column_nameN
}
```

Rename column `1` to `'age'` and `2` to `'AFIB'` for `df2`.

[30]:
```
df = df2.rename(columns={
    1: 'age',
    112: 'AFIB'
})
```

[30]:

	age	AFIB
0	77.0	0
1	55.0	0
2	52.0	0
3	68.0	0
4	60.0	0
...	...	...
1695	77.0	0

```
1696  70.0     0
1697  55.0     0
1698  79.0     0
1699  63.0     0

[1700 rows x 2 columns]
```

20 Task 12

(1 PT)

Dispplay a boxplot with AFIB on the x-axis and Age on the y-axis

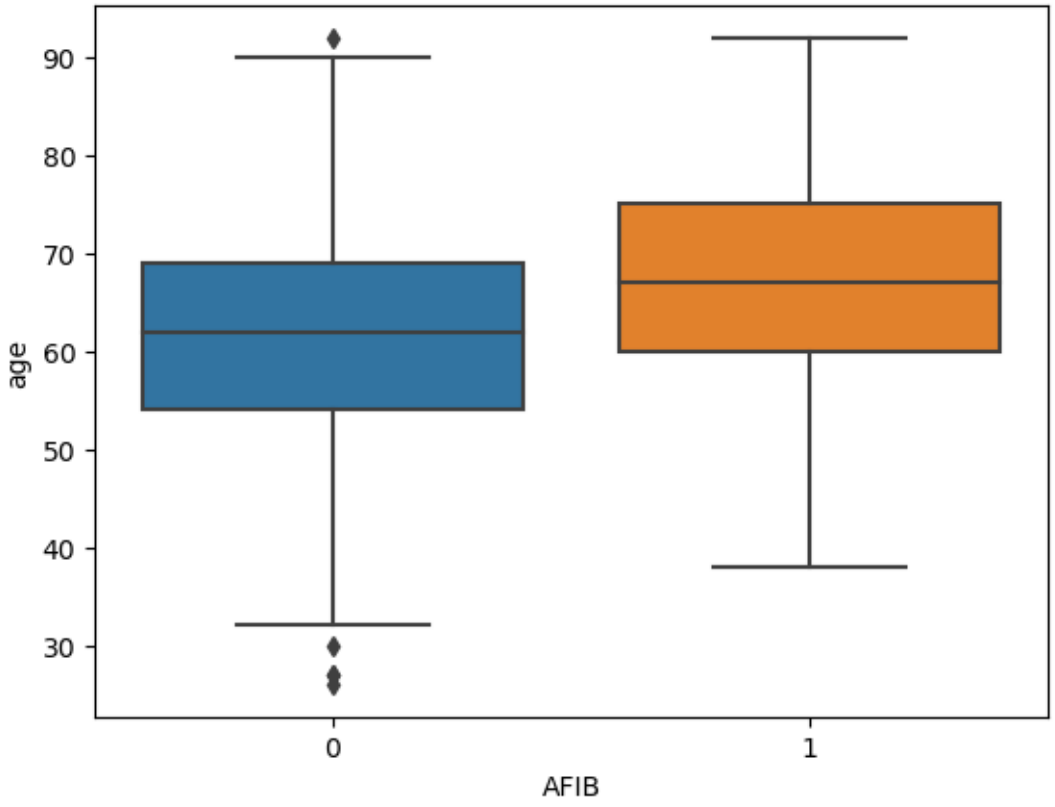
Use the the `boxplot()` function from the `seaborn` package for this.

Here is the documentation, but all you need to do is this:

```
from seaborn import boxplot
```

[31]:
```
from seaborn import boxplot
boxplot(data = df, x = 'AFIB', y='age')
```

[31]:
```
<Axes: xlabel='AFIB', ylabel='age'>
```



Ungraded question: What do you notice about the difference in age distributions between AFIB and non-AFIB groups?

The variability is about the same for both, but the AFIB group has a higher median age than the non-AFIB group.

[ ]: