Metadata

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

Student Info

- Name: Hilde Younce
- Net UD: ksg8xy
- URL of this file in GitHub: https://github.com/hyounce/DS5100-ksg8xy/blob/main/lessons/M06/hw06.ipynb

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

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.

Setting Up

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

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`.

In [ ]:
```
path_to_data = "http://archive.ics.uci.edu/ml/machine-learning-databases/005
```

Task 1

(1 PT)

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

In [ ]:
```
ml_data = pd.read_csv(path_to_data, header=None )
print(len(ml_data.index))
```

```
1700
```

Task 2

(1 PT)

Show the first three records in the dataset

In [ ]:
```
ml_data.head(3)
```

Out[ ]:

	0	1	2	3	4	5	6	7	8	9	...	114	115	116	117	118	119	120	121	122	1
0	1	77	1	2	1	1	2	?	3	0	...	0	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	0	0	
2	3	52	1	0	0	0	2	?	2	0	...	0	0	0	0	0	0	0	0	0	

3 rows × 124 columns

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).

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 `?`.

In [ ]:
```
num_quest = [val for val in ml_data[1] if val=='?']
len(num_quest)
```

Out[ ]: 8

Task 4

(1 PT)

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

In [ ]:
```
ml_data = ml_data.replace("?", np.nan)
```

Task 5

(1 PT)

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

In [ ]:
```
num_nan = [val for val in ml_data[1] if val is np.nan]
len(num_nan)
```

Out[ ]: 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)
```

Task 6

(1 PT)

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

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

Out[ ]: dtype('float64')

Task 7

(1 PT)

Compute the median age.

In [ ]:
```
ml_data[1].median()
```

Out[ ]:
```
63.0
```

Working with GENDER

The third column contains patient gender.

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

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.

In [ ]:
```
ml_data[2].value_counts()
```

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

In [ ]:
```
ml_data[2].value_counts(normalize=True)
```

Out[ ]:
```
2
1    0.626471
0    0.373529
Name: proportion, dtype: float64
```

Working with Essential Hypertension (EH)

Reference this column with `df[8]`.

Task 9

(1 PT)

Enter the most frequent value.

In [ ]:
```
freq_eh = ml_data[8].value_counts()
freq_eh
```

Out[ ]:
```
8
2    880
0    605
3    195
1     11
Name: count, dtype: int64
```

In [ ]:
```
freq_eh.index[0]
```

Out[ ]:
```
'2'
```

Working with Atrial Fibrillation (AFIB)

Reference this column with `df[112]`.

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

Task 10

(1 PT)

Print the number of AFIB cases.

Note that 1 means there is a case.

In [ ]:
```
freq_afib = ml_data[112].value_counts()
freq_afib
```

Out[ ]:
```
112
0    1530
1     170
Name: count, dtype: int64
```

In [ ]:
```
freq_afib[1]
```

Out[ ]:
```
170
```

Combining Age and AFIB

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.

In [ ]:
```
df2 = pd.DataFrame({'1': ml_data[1],
                    '2': ml_data[112]})
df2.shape
```

Out[ ]:
```
(1700, 2)
```

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`.

In [ ]:
```
df2 = df2.rename(columns={'1': 'age', '2': 'AFIB'})
df2
```

Out[ ]:

	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 × 2 columns

Task 12

(1 PT)

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

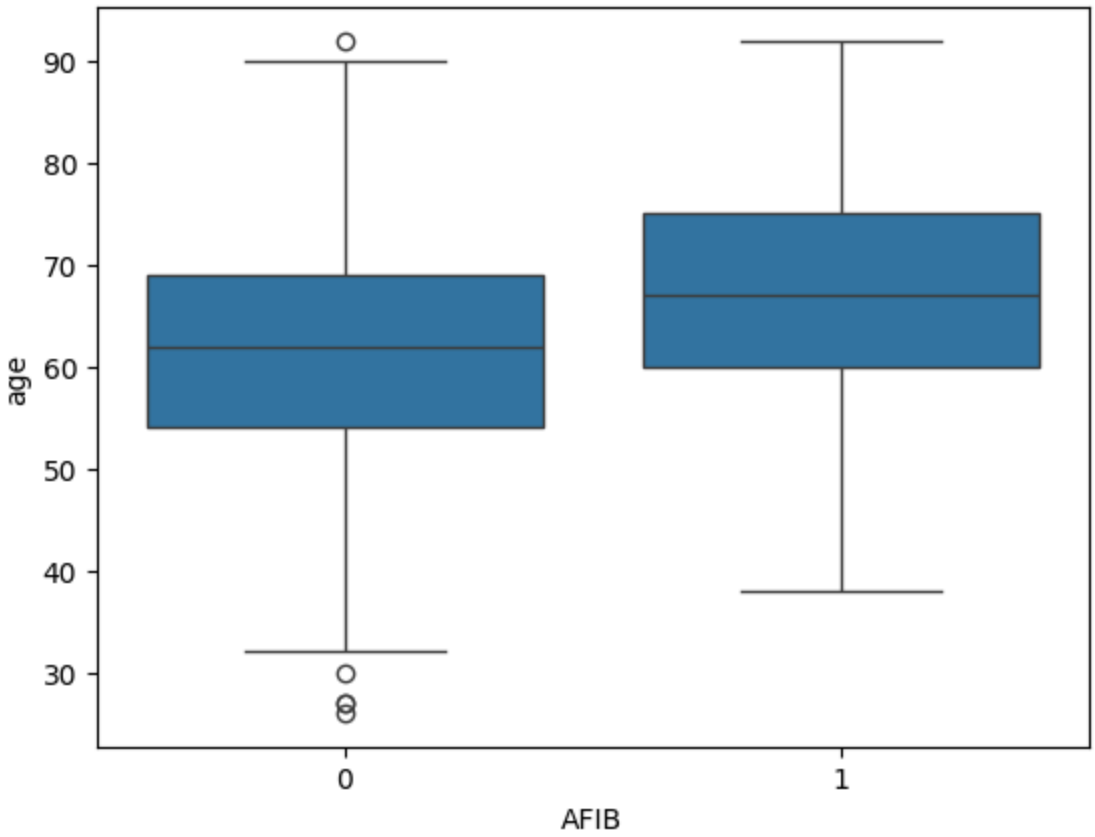
Use the the `boxplot()` function from the `seaborn` package for this.

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

```
from seaborn import boxplot
```

In [ ]:
```
from seaborn import boxplot
boxplot(x=df2['AFIB'], y=df2['age'])
```

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



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