

Pandas and Seaborn based homework

DSE5002 HD Sheets, revised Feb 2025

We will be working with the heart.csv data set

<https://www.kaggle.com/fedesoriano/heart-failure-prediction?select=heart.csv>

using tools in pandas and seaborn, and ideas from the two Jupyter notebooks we've seen this week

```
In [70]: !pip install pandas numpy seaborn plotnine
```

```
Requirement already satisfied: pandas in c:\users\water\anaconda3\envs\dse5002\lib\site-packages (2.2.3)
Requirement already satisfied: numpy in c:\users\water\anaconda3\envs\dse5002\lib\site-packages (2.2.4)
Requirement already satisfied: seaborn in c:\users\water\anaconda3\envs\dse5002\lib\site-packages (0.13.2)
Requirement already satisfied: plotnine in c:\users\water\anaconda3\envs\dse5002\lib\site-packages (0.14.5)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\water\anaconda3\envs\dse5002\lib\site-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\users\water\anaconda3\envs\dse5002\lib\site-packages (from pandas) (2025.1)
Requirement already satisfied: tzdata>=2022.7 in c:\users\water\anaconda3\envs\dse5002\lib\site-packages (from pandas) (2025.1)
Requirement already satisfied: matplotlib!=3.6.1,>=3.4 in c:\users\water\anaconda3\envs\dse5002\lib\site-packages (from seaborn) (3.10.1)
Requirement already satisfied: mizani~0.13.0 in c:\users\water\anaconda3\envs\dse5002\lib\site-packages (from plotnine) (0.13.1)
Requirement already satisfied: scipy>=1.8.0 in c:\users\water\anaconda3\envs\dse5002\lib\site-packages (from plotnine) (1.15.2)
Requirement already satisfied: statsmodels>=0.14.0 in c:\users\water\anaconda3\envs\dse5002\lib\site-packages (from plotnine) (0.14.4)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\water\anaconda3\envs\dse5002\lib\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (1.3.1)
Requirement already satisfied: cycler>=0.10 in c:\users\water\anaconda3\envs\dse5002\lib\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\water\anaconda3\envs\dse5002\lib\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (4.56.0)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\water\anaconda3\envs\dse5002\lib\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (1.4.8)
Requirement already satisfied: packaging>=20.0 in c:\users\water\anaconda3\envs\dse5002\lib\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (24.2)
Requirement already satisfied: pillow>=8 in c:\users\water\anaconda3\envs\dse5002\lib\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (11.1.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\water\anaconda3\envs\dse5002\lib\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (3.2.1)
Requirement already satisfied: six>=1.5 in c:\users\water\anaconda3\envs\dse5002\lib\site-packages (from python-dateutil>=2.8.2->pandas) (1.16.0)
Requirement already satisfied: patsy>=0.5.6 in c:\users\water\anaconda3\envs\dse5002\lib\site-packages (from statsmodels>=0.14.0->plotnine) (1.0.1)
```

```
In [71]: import pandas as pd
import numpy as np
import seaborn as sns
import plotnine as p9
```

```
In [72]: # make sure heart.csv is in your current working directory, or list the full path n

infile=r".\data\heart.csv"

bp_df=pd.read_csv(infile)
bp_df.head()
```

Out[72]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	Exer
0	40	M	ATA	140	289	0	Normal	172	
1	49	F	NAP	160	180	0	Normal	156	
2	37	M	ATA	130	283	0	ST	98	
3	48	F	ASY	138	214	0	Normal	108	
4	54	M	NAP	150	195	0	Normal	122	

Find or create the following

- a.) -Find the dimensions, memory used, and other basic information
- b.) -Run the data summary
- c.) Change the appropriate variables to type Categorical
- d.) -Create a pivot table (using the Pandas groupby operation) showing mean Resting BP by Sex, Resting ECG and HeartDisease-What does this tell you? What else can you figure out using a Pivot table, show me two other helpful pivot tables based on different variables, different groupings or different aggregation functions (count, mean, max etc)
- e.) -Show a histogram and the ECDF (empirical cumulative distribution function) for several continuous variables in the data set, in broad terms, what do the distributions look like, normal? exponential, poison-like?, uniform? Does this match your expectations?

<https://seaborn.pydata.org/generated/seaborn.ecdfplot.html>

https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.ecdf.html

f.) -Show An SNS Pairplot, the most informative version you can find, set the hue based on Heart Disease, try using at least one other variable as the Hue. Discuss what you think you are seeing in this plot

g.) Create several useful or informative boxplots of continuous variables by category, using Seaborn or PlotNine. Find an interesting result or contrast among the variables, discuss what you think it means or implies

h.) Create violin plots of these same results

i.) Find the mean, median and standard deviation of the Max heartrate variable in this data set

Turn this into a pivot table, grouping by one or more predictors.

Create all these results in this Notebook and turn it in

Source

[Pandas Docs](#)

Part A

a.) -Find the dimensions, memory used, and other basic information

In [73]: `bp_df.info()`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 918 entries, 0 to 917
Data columns (total 12 columns):
 #   Column            Non-Null Count  Dtype  
--- 
 0   Age               918 non-null    int64  
 1   Sex               918 non-null    object  
 2   ChestPainType    918 non-null    object  
 3   RestingBP         918 non-null    int64  
 4   Cholesterol      918 non-null    int64  
 5   FastingBS        918 non-null    int64  
 6   RestingECG       918 non-null    object  
 7   MaxHR             918 non-null    int64  
 8   ExerciseAngina   918 non-null    object  
 9   Oldpeak           918 non-null    float64 
 10  ST_Slope          918 non-null    object  
 11  HeartDisease     918 non-null    int64  
dtypes: float64(1), int64(6), object(5)
memory usage: 86.2+ KB
```

Part B

b.) -Run the data summary

In [74]: `bp_df.describe()`

Out[74]:

	Age	RestingBP	Cholesterol	FastingBS	MaxHR	Oldpeak	HeartDisea
count	918.000000	918.000000	918.000000	918.000000	918.000000	918.000000	918.0000
mean	53.510893	132.396514	198.799564	0.233115	136.809368	0.887364	0.5533
std	9.432617	18.514154	109.384145	0.423046	25.460334	1.066570	0.4974
min	28.000000	0.000000	0.000000	0.000000	60.000000	-2.600000	0.0000
25%	47.000000	120.000000	173.250000	0.000000	120.000000	0.000000	0.0000
50%	54.000000	130.000000	223.000000	0.000000	138.000000	0.600000	1.0000
75%	60.000000	140.000000	267.000000	0.000000	156.000000	1.500000	1.0000
max	77.000000	200.000000	603.000000	1.000000	202.000000	6.200000	1.0000

Part C

c.) Change the appropriate variables to type Categorical

Starting with the long approach here, I think this can be improved, though.

In [75]:

```
# Get unique values from the categorical columns
chest_pain_cats, sex_cats, resting_ecg_cats, exercise_angina_cats, st_slope_cats, h

#Set the chest pain type column to categorical with the categories defined by the u
bp_df["ChestPainType"] = pd.Categorical(bp_df["ChestPainType"], categories=chest_pa
bp_df["Sex"] = pd.Categorical(bp_df["Sex"], categories=sex_cats)
bp_df["RestingECG"] = pd.Categorical(bp_df["RestingECG"], categories=resting_ecg_c
bp_df["ExerciseAngina"] = pd.Categorical(bp_df["ExerciseAngina"], categories=exerci
bp_df["ST_Slope"] = pd.Categorical(bp_df["ST_Slope"], categories=st_slope_cats)
bp_df["HeartDisease"] = pd.Categorical(bp_df["HeartDisease"], categories=heart_dise
bp_df["FastingBS"] = pd.Categorical(bp_df["FastingBS"], categories=fasting_bs_cats)

#Verify
bp_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 918 entries, 0 to 917
Data columns (total 12 columns):
 #   Column            Non-Null Count  Dtype  
--- 
 0   Age               918 non-null    int64  
 1   Sex               918 non-null    category
 2   ChestPainType    918 non-null    category
 3   RestingBP         918 non-null    int64  
 4   Cholesterol      918 non-null    int64  
 5   FastingBS         918 non-null    category
 6   RestingECG        918 non-null    category
 7   MaxHR             918 non-null    int64  
 8   ExerciseAngina   918 non-null    category
 9   Oldpeak           918 non-null    float64 
 10  ST_Slope          918 non-null    category
 11  HeartDisease     918 non-null    category
dtypes: category(7), float64(1), int64(4)
memory usage: 43.2 KB
```

Now, let's make this a bit more functional...

```
In [76]: # Read in the original df
bp_df_original=pd.read_csv(infile)

# Convert HeartDisease and FastingBS to objects, as these are the exceptions for th
bp_df_original["HeartDisease"] = bp_df_original["HeartDisease"].astype("object")
bp_df_original["FastingBS"] = bp_df_original["FastingBS"].astype("object")

# All columns with the dtype "object" will become our categorical columns. Loop thr
for col in bp_df_original.columns:
    if bp_df_original[f"{col}"].dtype == "object":
        categories = pd.unique(bp_df_original[f"{col}"])
        bp_df_original[f"{col}"] = pd.Categorical(bp_df_original[f"{col}"], categor

#Verify
bp_df_original.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 918 entries, 0 to 917
Data columns (total 12 columns):
 #   Column            Non-Null Count  Dtype  
--- 
 0   Age               918 non-null    int64  
 1   Sex               918 non-null    category
 2   ChestPainType    918 non-null    category
 3   RestingBP         918 non-null    int64  
 4   Cholesterol      918 non-null    int64  
 5   FastingBS        918 non-null    category
 6   RestingECG        918 non-null    category
 7   MaxHR             918 non-null    int64  
 8   ExerciseAngina   918 non-null    category
 9   Oldpeak           918 non-null    float64 
 10  ST_Slope          918 non-null    category
 11  HeartDisease     918 non-null    category
dtypes: category(7), float64(1), int64(4)
memory usage: 43.2 KB
```

Great, now this is flexible and can be applied to arbitrarily large data sets. Much better than manually doing each type change.

Part D

d.) -Create a pivot table (using the Pandas groupby operation) showing mean Resting BP by Sex, Resting ECG and HeartDisease-What does this tell you? What else can you figure out using a Pivot table, show me two other helpful pivot tables based on different variables, different groupings or different aggregation functions (count, mean, max etc)

In [77]: `bp_df[["RestingBP", "Sex"]].groupby(["Sex"]).mean()`

C:\Users\water\AppData\Local\Temp\ipykernel_22168\233112352.py:1: FutureWarning: The default of observed=False is deprecated and will be changed to True in a future version of pandas. Pass observed=False to retain current behavior or observed=True to adopt the future default and silence this warning.

Out[77]: `RestingBP`

Sex
M 132.445517
F 132.212435

In [78]: `bp_df[["RestingBP", "RestingECG"]].groupby(["RestingECG"]).mean()`

```
C:\Users\water\AppData\Local\Temp\ipykernel_22168\2621982121.py:1: FutureWarning: The default of observed=False is deprecated and will be changed to True in a future version of pandas. Pass observed=False to retain current behavior or observed=True to adopt the future default and silence this warning.
```

Out[78]:

RestingBP**RestingECG****Normal** 130.635870**ST** 135.808989**LVH** 134.335106In [79]: `bp_df[["RestingBP", "HeartDisease"]].groupby(["HeartDisease"]).mean()`

```
C:\Users\water\AppData\Local\Temp\ipykernel_22168\2443954490.py:1: FutureWarning: The default of observed=False is deprecated and will be changed to True in a future version of pandas. Pass observed=False to retain current behavior or observed=True to adopt the future default and silence this warning.
```

Out[79]:

RestingBP**HeartDisease****0** 130.180488**1** 134.185039

These pivot tables help highlight the respective relationship between sex, resting ECG, heart disease, and mean resting blood pressure.

Now, lets take a look at how MaxHR is related to heart disease

In [80]: `bp_df[["MaxHR", "HeartDisease"]].groupby(["HeartDisease"]).mean()`

```
C:\Users\water\AppData\Local\Temp\ipykernel_22168\1575525349.py:1: FutureWarning: The default of observed=False is deprecated and will be changed to True in a future version of pandas. Pass observed=False to retain current behavior or observed=True to adopt the future default and silence this warning.
```

Out[80]:

MaxHR**HeartDisease****0** 148.151220**1** 127.655512

Okay, this is what I would expect. Heart disease has a significant impact on max heartrate, which is a good indicator of overall cardiovascular fitness. There is, however, a confounding variable here, which is age. Let's see the average ages of people with and without heart disease...

```
In [81]: bp_df[["Age", "HeartDisease"]].groupby(["HeartDisease"]).mean()
```

```
C:\Users\water\AppData\Local\Temp\ipykernel_22168\1810670197.py:1: FutureWarning: The default of observed=False is deprecated and will be changed to True in a future version of pandas. Pass observed=False to retain current behavior or observed=True to adopt the future default and silence this warning.
```

```
Out[81]:
```

HeartDisease	Age
0	50.551220
1	55.899606

Ahhh, okay. Since max heart rate is a function of age, as is the propensity for heart disease, I hypothesized that the heart disease group might be older on average. This hypothesis ended up being correct, as the heart disease group is about 5 years older on average than the non-heart disease group. Without any further analysis, it is implied that both age and heart disease affect max heart rate.

Part E

e.) -Show a histogram and the ECDF (empirical cumulative distribution function) for several continuous variables in the data set, in broad terms, what do the distributions look like, normal? exponential, poison-like?, uniform? Does this match your expectations?

[ECDF Source - Seaborn](#)

[ECDF Source - Matplotlib](#)

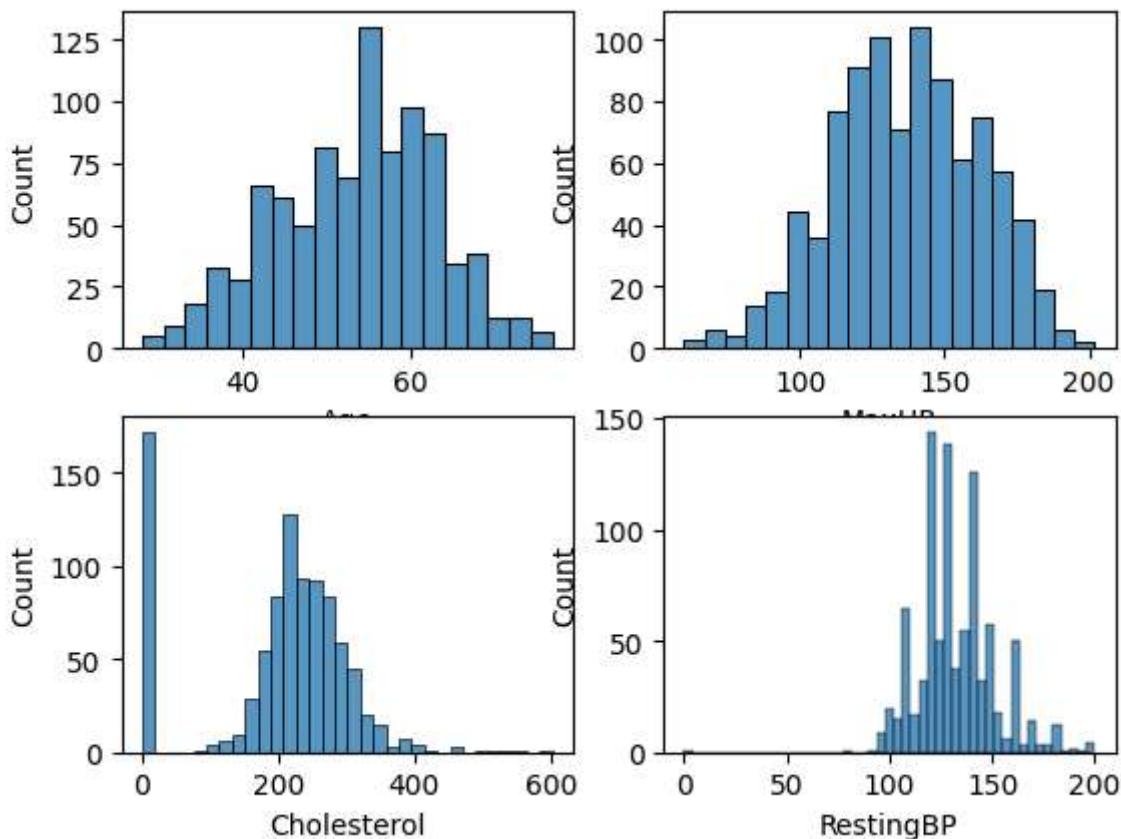
Let's do this with both seaborn and matplotlib to see how they differ. First, seaborn... Plot the hist and

ecdf for age, maxhr, cholesterol, and restingbp. I am going to cheat a bit and use the matplotlib subplot feature to pack everything into one plot and make this more compact.

```
In [82]: import matplotlib.pyplot as plt
```

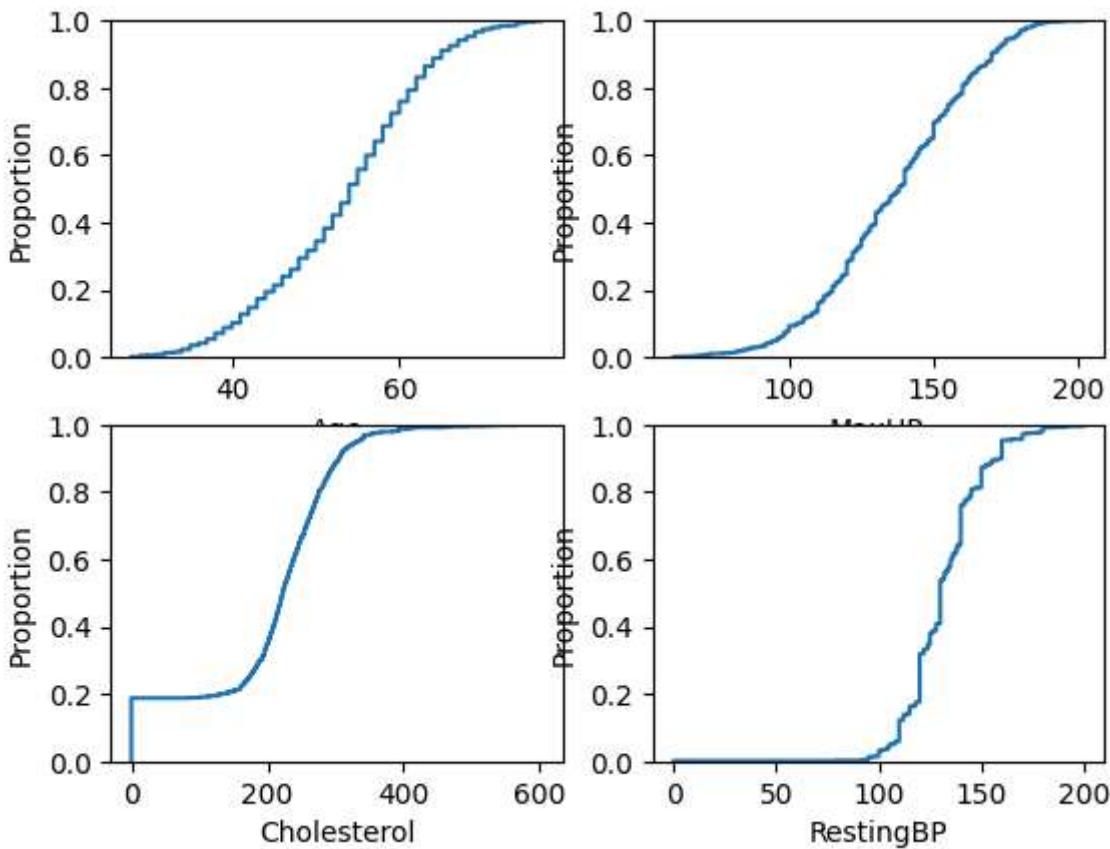
```
In [83]: fig, axs = plt.subplots(nrows=2, ncols=2)
sns.histplot(bp_df["Age"], ax=axs[0,0])
sns.histplot(bp_df["MaxHR"], ax=axs[0,1])
sns.histplot(bp_df["Cholesterol"], ax=axs[1,0])
sns.histplot(bp_df["RestingBP"], ax=axs[1,1])
```

```
Out[83]: <Axes: xlabel='RestingBP', ylabel='Count'>
```



```
In [84]: fig, axs = plt.subplots(nrows=2, ncols=2)
sns.ecdfplot(bp_df["Age"], ax=axs[0,0])
sns.ecdfplot(bp_df["MaxHR"], ax=axs[0,1])
sns.ecdfplot(bp_df["Cholesterol"], ax=axs[1,0])
sns.ecdfplot(bp_df["RestingBP"], ax=axs[1,1])
```

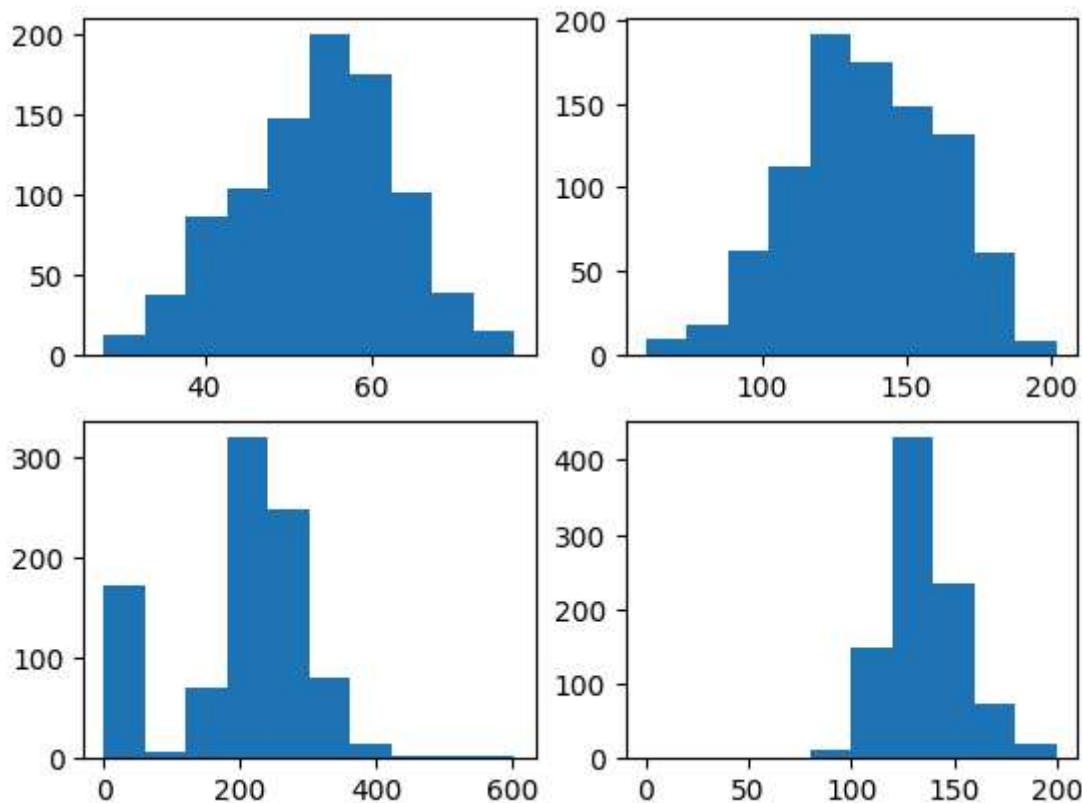
```
Out[84]: <Axes: xlabel='RestingBP', ylabel='Proportion'>
```



That was relatively painless. Now, let's try matplotlib.

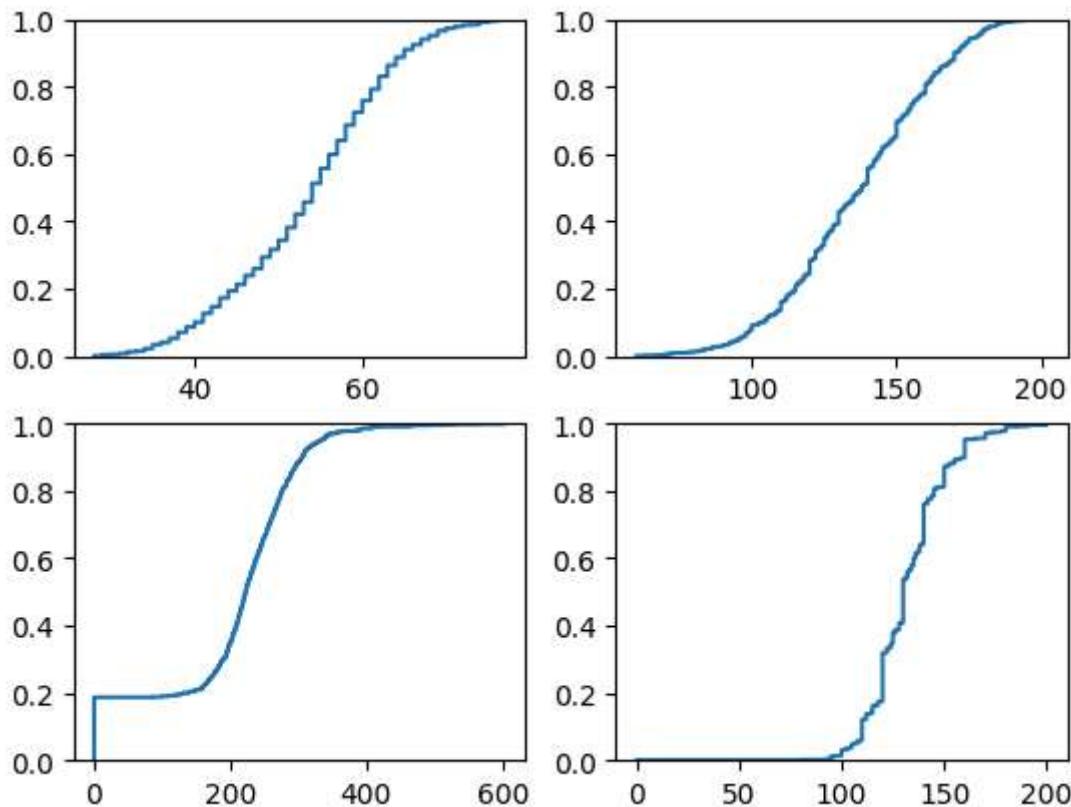
```
In [85]: fig, axs = plt.subplots(nrows=2, ncols=2)
axs[0,0].hist(bp_df["Age"])
axs[0,1].hist(bp_df["MaxHR"])
axs[1,0].hist(bp_df["Cholesterol"])
axs[1,1].hist(bp_df["RestingBP"])
```

```
Out[85]: (array([ 1.,  0.,  0.,  0., 12., 148., 430., 234., 73., 20.]),
 array([ 0., 20., 40., 60., 80., 100., 120., 140., 160., 180., 200.]),
 <BarContainer object of 10 artists>)
```



```
In [86]: fig, axs = plt.subplots(nrows=2, ncols=2)
axs[0,0].ecdf(bp_df["Age"])
axs[0,1].ecdf(bp_df["MaxHR"])
axs[1,0].ecdf(bp_df["Cholesterol"])
axs[1,1].ecdf(bp_df["RestingBP"])
```

```
Out[86]: <matplotlib.lines.Line2D at 0x1cf954560c0>
```



Slightly different, but I would say comparable effort. In the future, it seems like matplotlib will be better for this kind of analysis because of the multiplotting feature. That was not something I could find in seaborn for unrelated plots.

Time to discuss the plots... they are strongly normally distributed, aside from outliers in the cholesterol and resting blood pressure charts. Cholesterol has a large count of 0, which I assume are people who did not have their cholesterol measured, otherwise they would be dead. I assume the same is true for the set of blood pressure measurements that were 0, but that seems to only be one person.

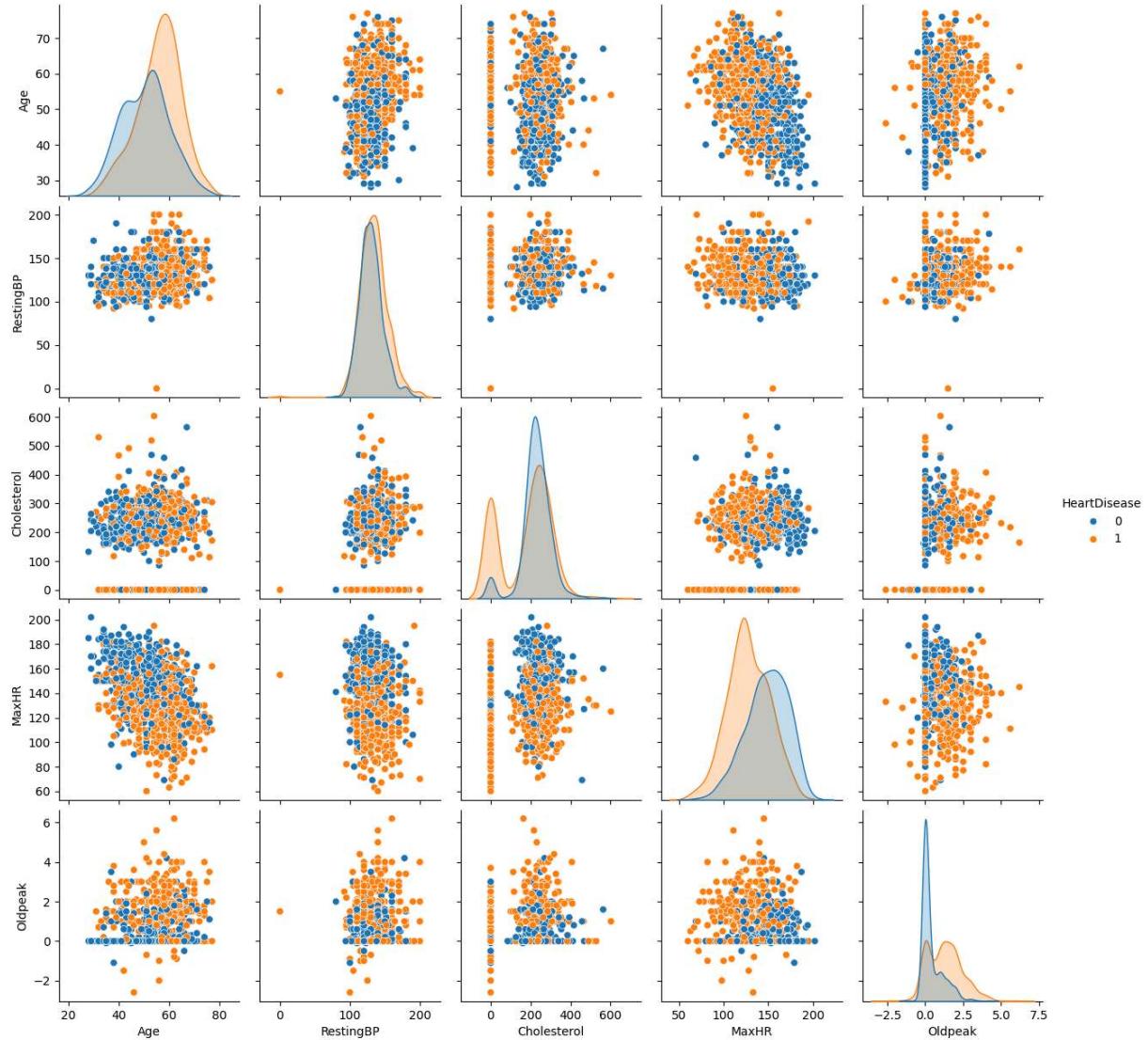
Part F

f.) -Show An SNS Pairplot, the most informative version you can find, set the hue based on Heart Disease, try using at least one other variable as the Hue. Discuss what you think you

are seeing in this plot

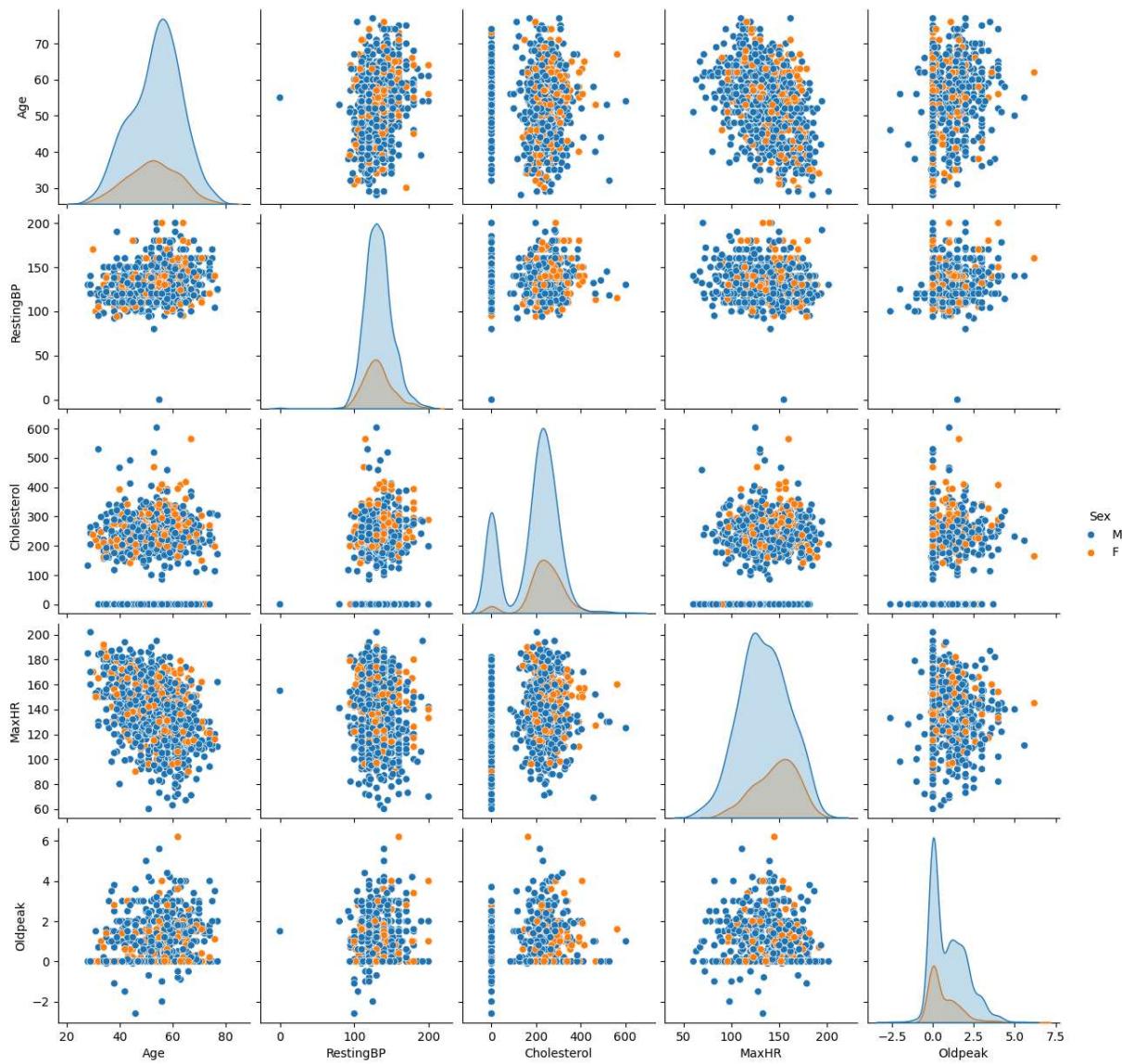
```
In [87]: sns.pairplot(bp_df, hue="HeartDisease")
```

```
Out[87]: <seaborn.axisgrid.PairGrid at 0x1cf928559d0>
```



```
In [88]: sns.pairplot(bp_df, hue="Sex")
```

```
Out[88]: <seaborn.axisgrid.PairGrid at 0x1d0029fe450>
```



The pairplot chart seems to compare all continuous variables against each other, with the hue as the categorical measure. The grid spaces where the variables are compared to themselves feature a distribution plot. This is very similar to the pairplot in R, but unlike R, the correlation between the variables is not explicitly defined.

Part G

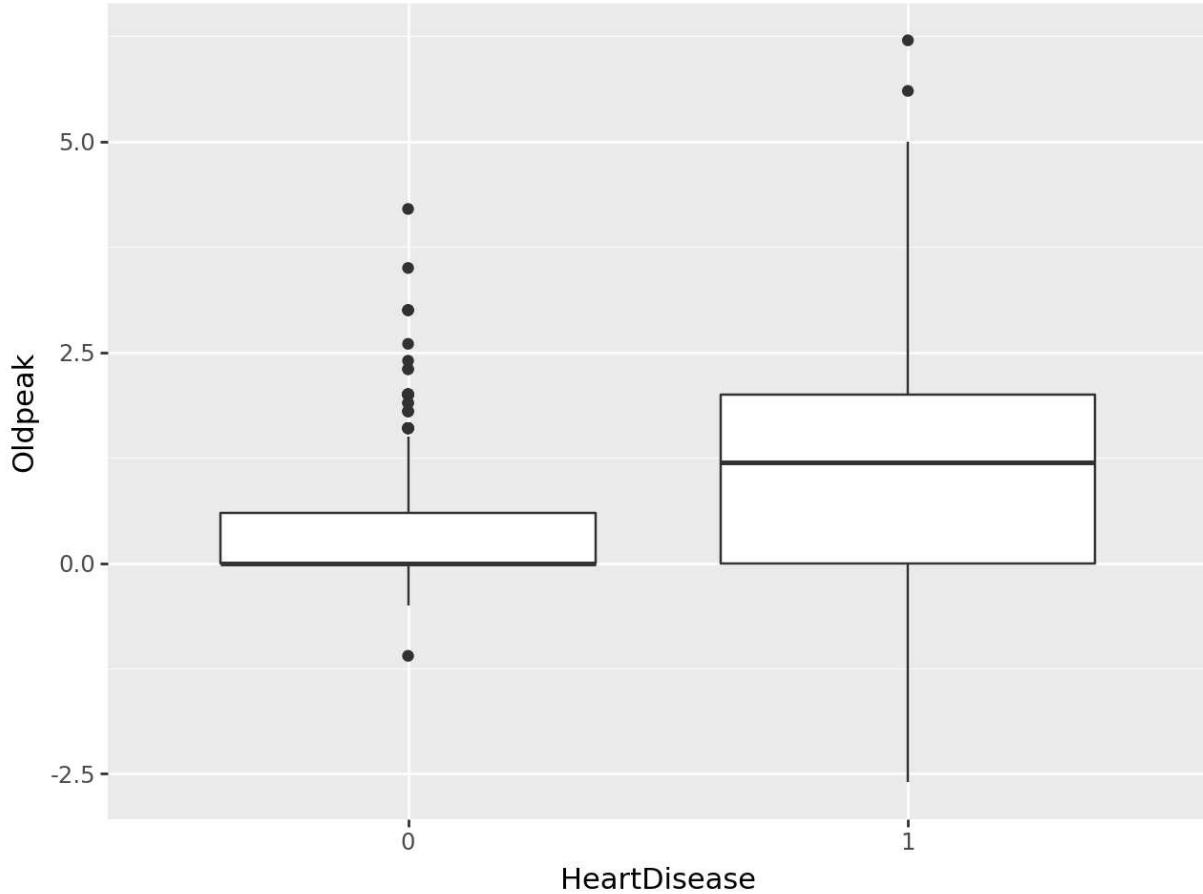
g.) Create several useful or informative boxplots of continuous variables by category, using Seaborn or PlotNine. Find an interesting result or contrast among the variables, discuss what you think it means or implies

Using plotnine here to see how it works...

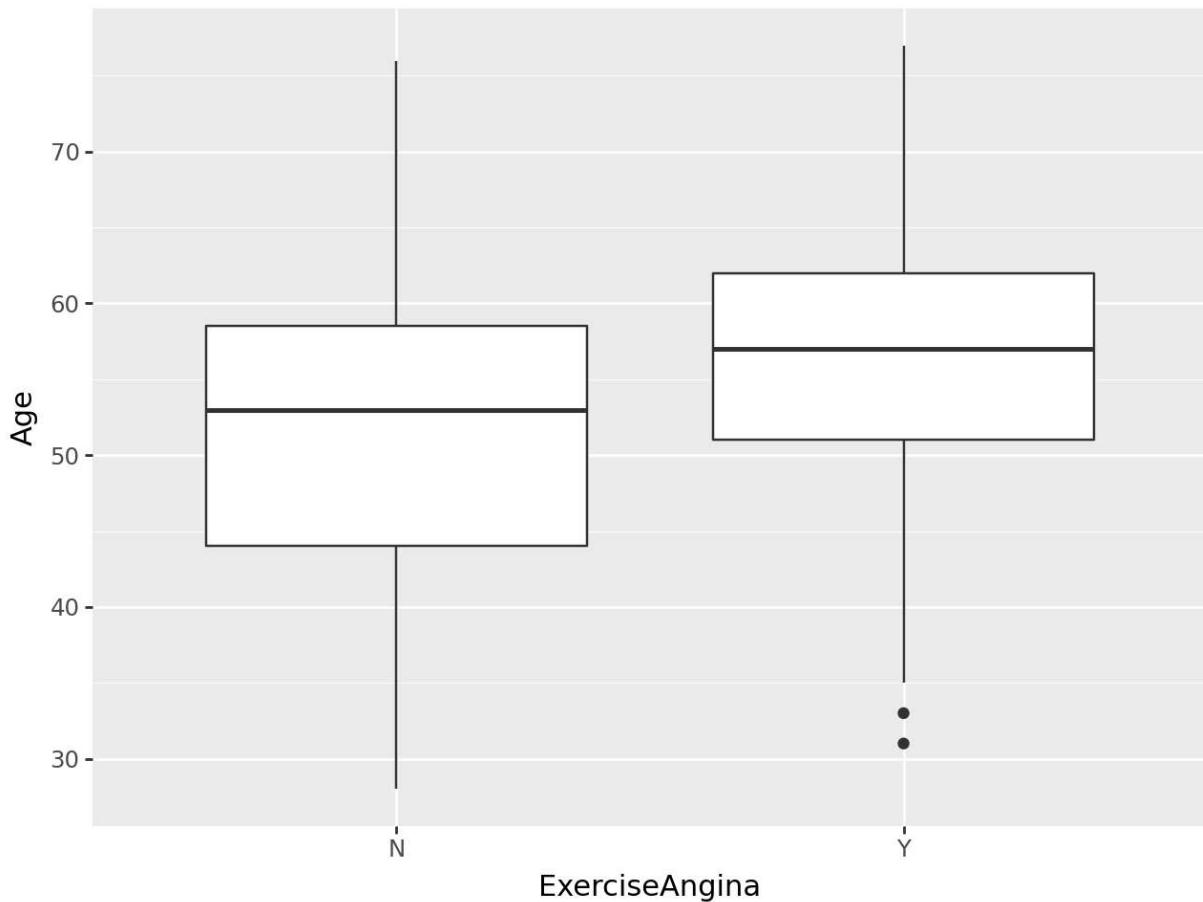
[PlotNine Reference](#)

Immediately, I get the sense that this is a python implementation of the ggplot R library. The syntax is identical.

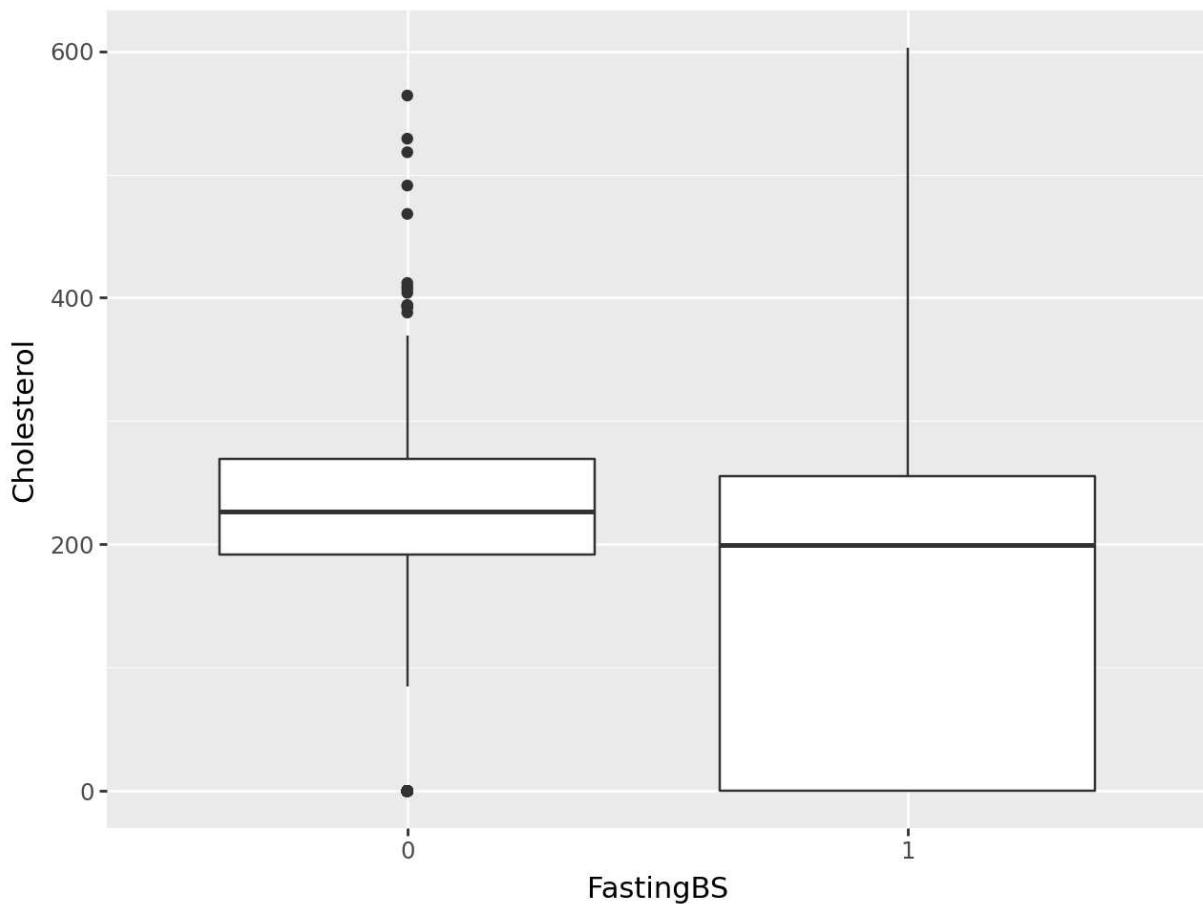
```
In [93]: ggplot = p9.ggplot  
geom_boxplot = p9.geom_boxplot  
aes=p9.aes  
  
(  
    ggplot(bp_df)  
    + geom_boxplot(aes(x="HeartDisease", y="Oldpeak"))  
)
```



```
In [94]: (  
    ggplot(bp_df)  
    + geom_boxplot(aes(x="ExerciseAngina", y="Age"))  
)
```



```
In [95]: (
    ggplot(bp_df)
    + geom_boxplot(aes(x="FastingBS", y="Cholesterol"))
)
```



Based on how these graphs look, it seems like `plotnine` literally ports R's rendering engine into python. Pretty neat. I already have R on my machine, but I would be interested in knowing if this package requires R to be installed, or if it installs the rendering engine as part of the package.

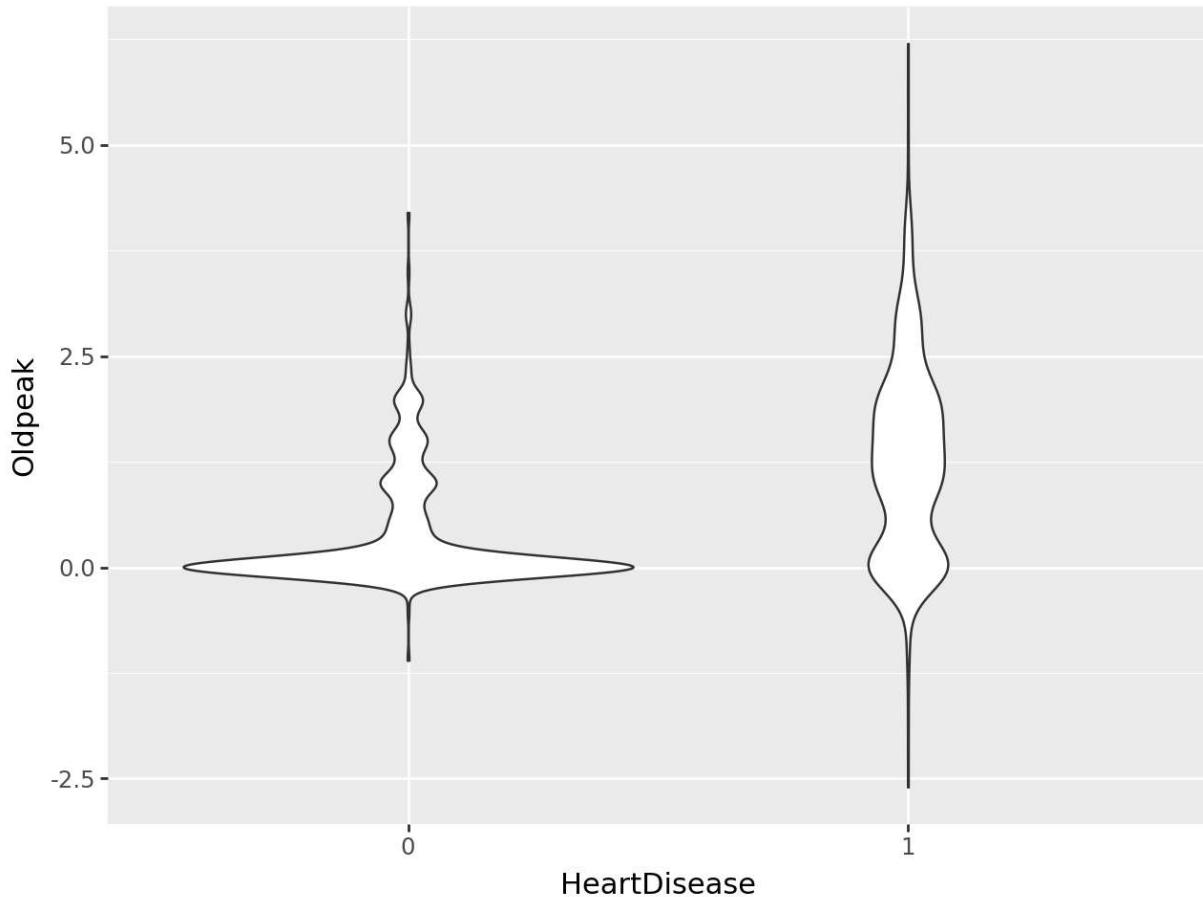
Part H

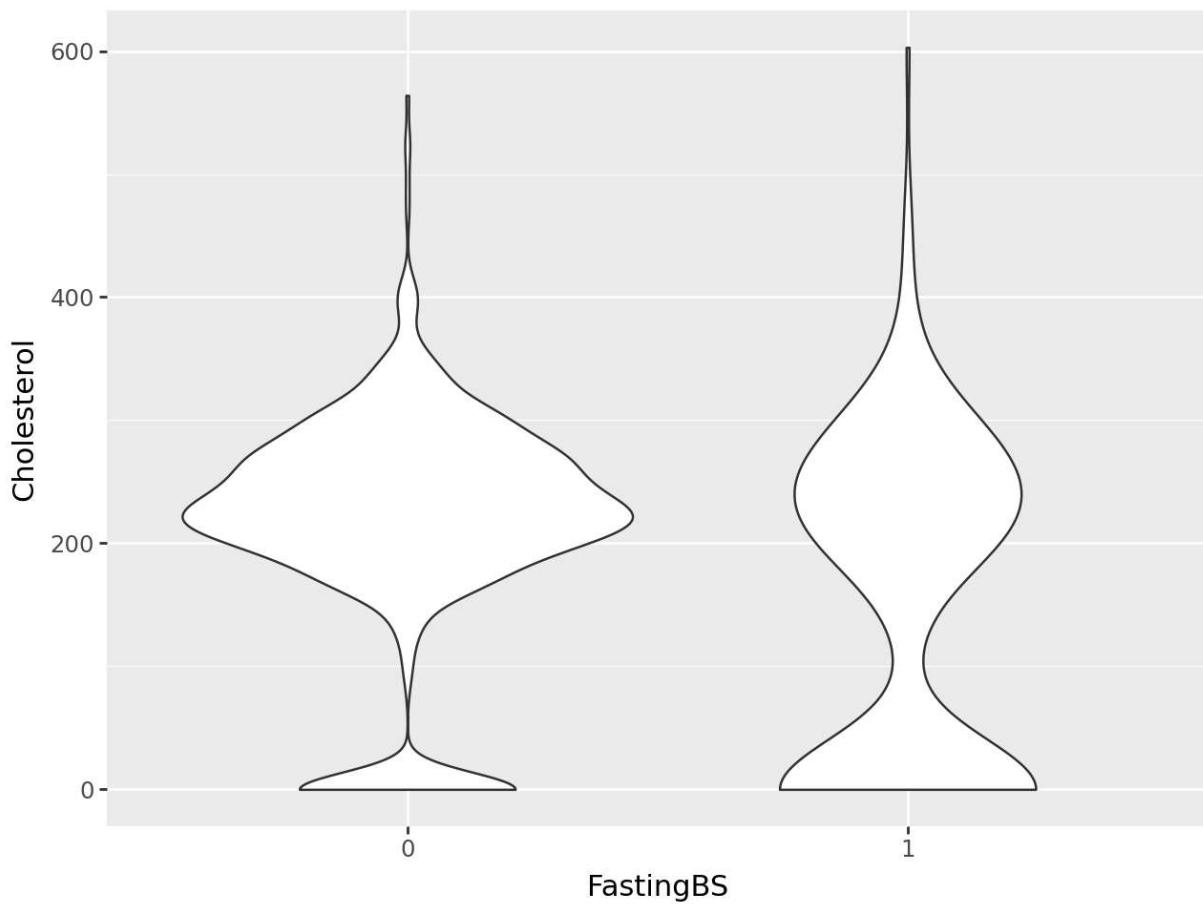
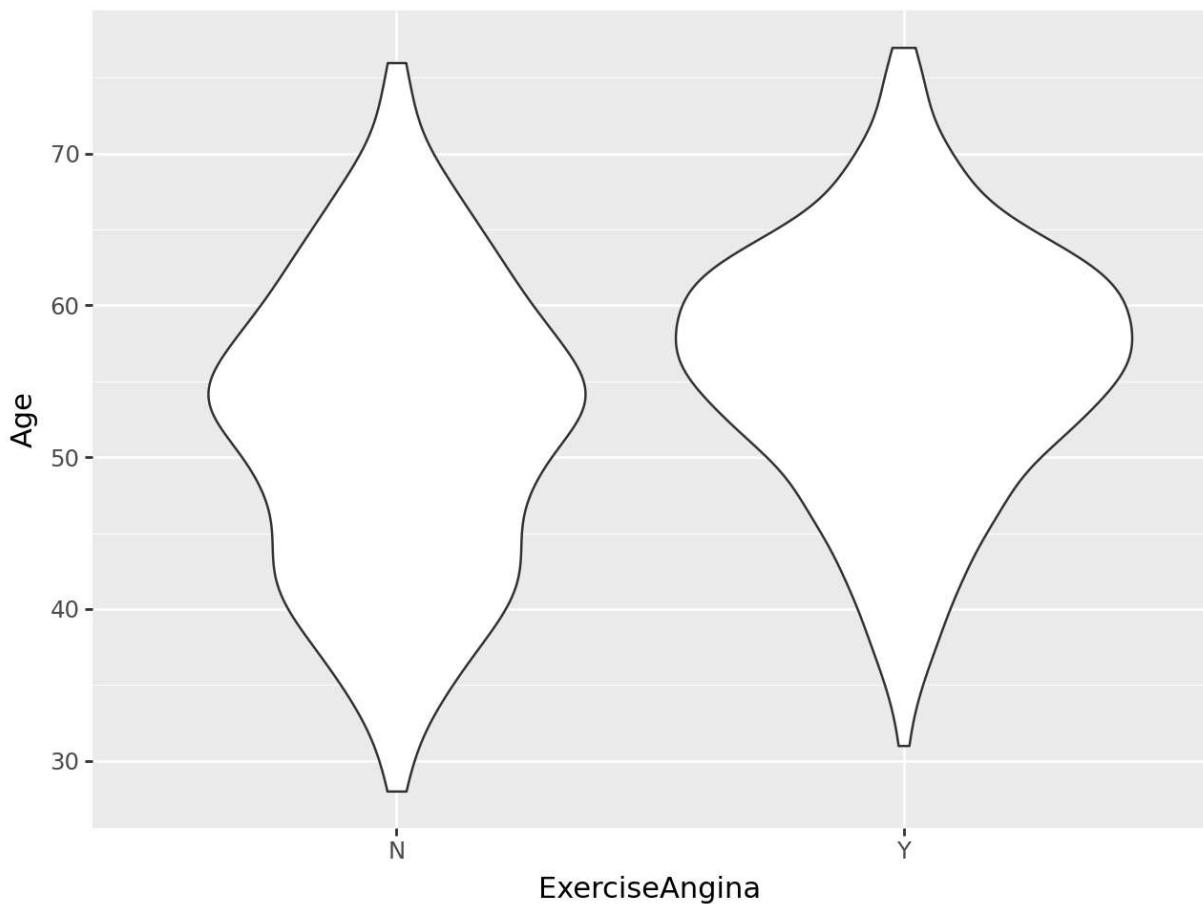
h.) Create violin plots of these same results

This is pretty easy to do in R with `ggplot`, but this would be a good time to compare and contrast libraries.

Plotnine implementation:

```
In [114]:  
geom_violin = p9.geom_violin  
  
plt1 = (  
    ggplot(bp_df)  
    + geom_violin(aes(x="HeartDisease", y="Oldpeak"))  
)  
  
plt2 = (  
    ggplot(bp_df)  
    + geom_violin(aes(x="ExerciseAngina", y="Age"))  
)  
  
plt3 = (  
    ggplot(bp_df)  
    + geom_violin(aes(x="FastingBS", y="Cholesterol"))  
)  
  
for i, plot in enumerate([plt1, plt2, plt3]):  
    plot.show()
```



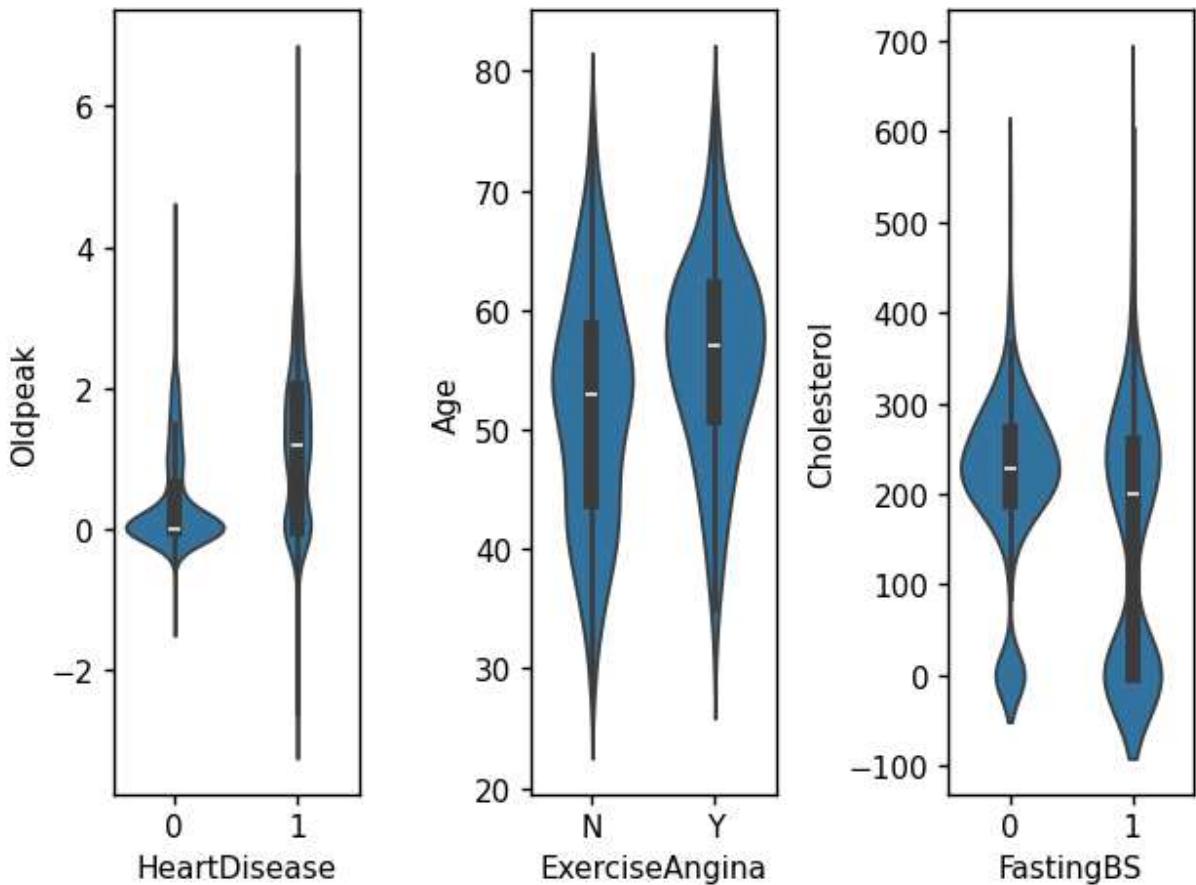


Seaborn Implementation:

```
In [ ]: fig, axs = plt.subplots(ncols=3)

sns.violinplot(data=bp_df, x="HeartDisease", y="Oldpeak", ax=axs[0])
sns.violinplot(data=bp_df, x="ExerciseAngina", y="Age", ax=axs[1])
sns.violinplot(data=bp_df, x="FastingBS", y="Cholesterol", ax=axs[2])

plt.tight_layout()
```



Matplotlib Implementation:

```
In [148...]: fig, axs = plt.subplots(ncols=3, figsize=(10, 4))

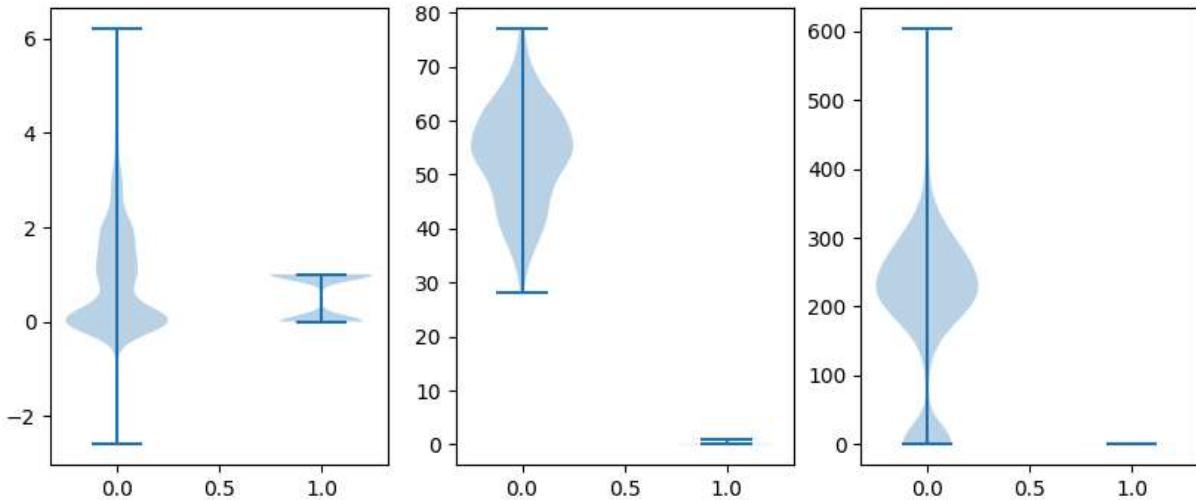
# After messing around with this for a while, it became clear that matplotlib's violin
# categorical variable is not an integer. Let's convert ExerciseAngina to an int.
converted = bp_df["ExerciseAngina"].replace(["Y", "N"], [1, 0])

axs[0].violinplot(bp_df[["Oldpeak", "HeartDisease"]], positions=[0,1])
axs[1].violinplot([bp_df["Age"]], converted, positions=[0,1])
axs[2].violinplot(bp_df[["Cholesterol", "FastingBS"]], positions=[0,1])
```

```
C:\Users\water\AppData\Local\Temp\ipykernel_22168\3703374422.py:5: FutureWarning: Downcasting behavior in `replace` is deprecated and will be removed in a future version. To retain the old behavior, explicitly call `result.infer_objects(copy=False)`. To opt-in to the future behavior, set `pd.set_option('future.no_silent_downcasting', True)`
```

```
C:\Users\water\AppData\Local\Temp\ipykernel_22168\3703374422.py:5: FutureWarning: The behavior of Series.replace (and DataFrame.replace) with CategoricalDtype is deprecated. In a future version, replace will only be used for cases that preserve the categories. To change the categories, use ser.cat.rename_categories instead.
```

Out[148... { 'bodies': [<matplotlib.collections.FillBetweenPolyCollection at 0x1d01296b440>, <matplotlib.collections.FillBetweenPolyCollection at 0x1d014acbbc0>], 'cmaxes': <matplotlib.collections.LineCollection at 0x1d012954bc0>, 'cmins': <matplotlib.collections.LineCollection at 0x1d012f86e70>, 'cbars': <matplotlib.collections.LineCollection at 0x1d0129031d0>}



I could be doing something wrong here, but it doesn't seem like matplotlib is equipped to properly handle categorical data in violin plots. Good thing there are two other libraries that do this just fine :).

Part I

i.) Find the mean, median and standard deviation of the Max heartrate variable in this data set. Turn this into a pivot table, grouping by one or more predictors.

In [149... # First find these metrics on the MaxHR data alone.
mean = bp_df["MaxHR"].mean()
median = bp_df["MaxHR"].median()
std = bp_df["MaxHR"].std()

print(f"Mean: {mean}\tMedian: {median}\tStandard Deviation: {std}")

Mean: 136.80936819172112
25.4603341382503 Median: 138.0 Standard Deviation:

```
In [ ]: # Now, pivot tables  
bp_df[["MaxHR", "Age"]].groupby("Age").mean()
```

Out[]: MaxHR

Age**28** 185.0**29** 170.0**30** 170.0**31** 151.5**32** 155.0**33** 167.5**34** 174.0**35** 156.0**36** 175.0**37** 158.0**38** 150.0**39** 150.0**40** 152.0**41** 159.0**42** 151.0**43** 144.0**44** 153.0**45** 144.0**46** 136.5**47** 145.0**48** 138.0**49** 145.0**50** 140.0**51** 139.0**52** 143.5**53** 130.0**54** 140.0**55** 136.0**56** 124.5

MaxHR**Age****57** 135.5**58** 130.5**59** 131.0**60** 140.0**61** 120.0**62** 122.0**63** 131.0**64** 130.5**65** 122.0**66** 120.0**67** 129.0**68** 135.5**69** 126.0**70** 125.0**71** 125.0**72** 111.5**73** 121.0**74** 116.0**75** 112.0**76** 118.0**77** 136.0In [153...]: `bp_df[['MaxHR', 'Age']].groupby('Age').median()`

Out[153...]

MaxHR**Age****28** 185.0**29** 170.0**30** 170.0**31** 151.5**32** 155.0**33** 167.5**34** 174.0**35** 156.0**36** 175.0**37** 158.0**38** 150.0**39** 150.0**40** 152.0**41** 159.0**42** 151.0**43** 144.0**44** 153.0**45** 144.0**46** 136.5**47** 145.0**48** 138.0**49** 145.0**50** 140.0**51** 139.0**52** 143.5**53** 130.0**54** 140.0**55** 136.0**56** 124.5

MaxHR**Age****57** 135.5**58** 130.5**59** 131.0**60** 140.0**61** 120.0**62** 122.0**63** 131.0**64** 130.5**65** 122.0**66** 120.0**67** 129.0**68** 135.5**69** 126.0**70** 125.0**71** 125.0**72** 111.5**73** 121.0**74** 116.0**75** 112.0**76** 118.0**77** 136.0In [154...]: `bp_df[['MaxHR', 'Age']].groupby('Age').std()`

Out[154...]

MaxHR**Age****28** NaN**29** 21.939310**30** NaN**31** 2.121320**32** 24.035391**33** 24.748737**34** 16.399768**35** 20.300022**36** 21.979536**37** 26.481898**38** 22.860446**39** 21.974661**40** 30.222466**41** 20.633271**42** 23.724522**43** 20.522125**44** 24.378593**45** 20.461745**46** 21.433923**47** 23.502395**48** 27.712929**49** 22.170873**50** 18.450113**51** 26.574361**52** 28.934354**53** 19.533673**54** 21.962963**55** 23.963387**56** 24.259620

MaxHR

Age
57 26.561933
58 25.313596
59 22.205193
60 23.134948
61 24.477200
62 25.895134
63 26.509508
64 17.637585
65 26.519624
66 23.751653
67 25.542309
68 12.564942
69 18.589355
70 21.866478
71 20.844664
72 11.898879
73 NaN
74 11.328430
75 2.309401
76 2.828427
77 36.769553

**Just for fun, let's group by multiple predictors.
There are several ages, so this will have to be
written to a csv to actually analyze.**

```
In [160...]: mean_data = bp_df[["MaxHR", "Age", "HeartDisease"]].groupby(["Age", "HeartDisease"]).mean().to_csv(r"Data\Mean_data.csv")
```

C:\Users\water\AppData\Local\Temp\ipykernel_22168\736809882.py:1: FutureWarning: The default of observed=False is deprecated and will be changed to True in a future version of pandas. Pass observed=False to retain current behavior or observed=True to adopt the future default and silence this warning.

```
In [161... median_data = bp_df[["MaxHR", "Age", "HeartDisease"]].groupby(["Age", "HeartDisease"]) median_data.to_csv(r"Data\Median_data.csv")
```

C:\Users\water\AppData\Local\Temp\ipykernel_22168\4175941233.py:1: FutureWarning: The default of observed=False is deprecated and will be changed to True in a future version of pandas. Pass observed=False to retain current behavior or observed=True to adopt the future default and silence this warning.

```
In [162... std_data = bp_df[["MaxHR", "Age", "HeartDisease"]].groupby(["Age", "HeartDisease"]). std_data.to_csv(r"Data\std_data.csv")
```

C:\Users\water\AppData\Local\Temp\ipykernel_22168\2868568099.py:1: FutureWarning: The default of observed=False is deprecated and will be changed to True in a future version of pandas. Pass observed=False to retain current behavior or observed=True to adopt the future default and silence this warning.