Lab 2: Working with data, visualizations

The Howell Data set

This data set will be our friend. It is pretty small, but still has some interesting characteristics. Our goal in this lab is to do an initial exploration and prepare it for use by a model.

Start up with getting the data.

I suggest creating a folder with a name like lab2_work.

Copy lab2_starter.ipynb. Copy howell.csv into the same directory (workspace) as lab1_starter.ipynb. In the third cell, enter the following three lines of python code.

```
import pandas as pd

howell_full = pd.read_csv("Howell.csv", sep=(";")
howell_full.info()
```

1. This trio of lines is our standard opening. Code descriptions are available in the previous lab with one exception. Some CSV files use semicolons instead of commas to separate values. This data set is one of them. We let the read_csv function know this by telling it that the separator character is ";". (While we could edit the csv file with a text editor, this is cleaner.)
2. I know that I am going to be breaking this data frame into smaller pieces, so I will use a descriptive name. I thought about calling it Howell_full_df, but I don't expect there to be much confusion between my data frame and anything else.

You should see the following

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 544 entries, 0 to 543
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   height  544 non-null    float64
 1   weight  544 non-null    float64
 2   age     544 non-null    float64
 3   male    544 non-null    int64
dtypes: float64(3), int64(1)
memory usage: 17.1 KB
```

Submission 1 of 14: Progress mark (5)
Screen shot of notebook with code and result

Submission 2 of 14: Analysis (15) from info() determine

1) How many data instances are there?
2) How many features are there?
3) What are the names?
4) Are there any missing values?
5) Are there any non-numeric features?

Quick look at distributions.

In the next code cell, enter the following three lines of python code.

```
print(howell_full.head(n=10))
print(howell_full.describe())
howell_full.corr()
```

1. This trio of lines is a good next step. We have seen head before.
2. The describe function is going to give us basic statistics for each of our features. We get the average and standard deviation, the quartiles, max and min.
3. The corr function will give us the correlation between the features. Correlation coefficients range from -1 to 1, with 0 indicating no correlation. Just because the correlation is close to zero, does not mean that there is no pattern, just that it isn't a linear pattern. See the book Fig 2-14.

Here is what the output should look like:

```
    height     weight   age  male
0  151.765  47.825606  63.0     1
1  139.700  36.485807  63.0     0
2  136.525  31.864838  65.0     0
3  156.845  53.041915  41.0     1
4  145.415  41.276872  51.0     0
5  163.830  62.992589  35.0     1
6  149.225  38.243476  32.0     0
7  168.910  55.479971  27.0     1
8  147.955  34.869885  19.0     0
9  165.100  54.487739  54.0     1

           height      weight         age        male
count  544.000000  544.000000  544.000000  544.000000
mean   138.263596   35.610618   29.344393    0.472426
std     27.602448   14.719178   20.746888    0.499699
min     53.975000    4.252425    0.000000    0.000000
25%    125.095000   22.007717   12.000000    0.000000
50%    148.590000   40.057844   27.000000    0.000000
75%    157.480000   47.209005   43.000000    1.000000
max    179.070000   62.992589   88.000000    1.000000
                                                                  [15]:
```

```
          height    weight       age      male
height  1.000000  0.940822  0.683689  0.139229
weight  0.940822  1.000000  0.678335  0.155443
   age  0.683689  0.678335  1.000000  0.005887
  male  0.139229  0.155443  0.005887  1.000000
```

Submission 3 of 14: Progress mark (5)
Screen shot of notebook with code and result for distribution

Submission 4 of 14: Analysis (15) from the results determine

6) Are the data instances sorted on any of the attributes?
7) What are the units of height?
8) What are the units of weight?
9) What are the minimum, median and max age?
10) What two different features have the highest correlation?

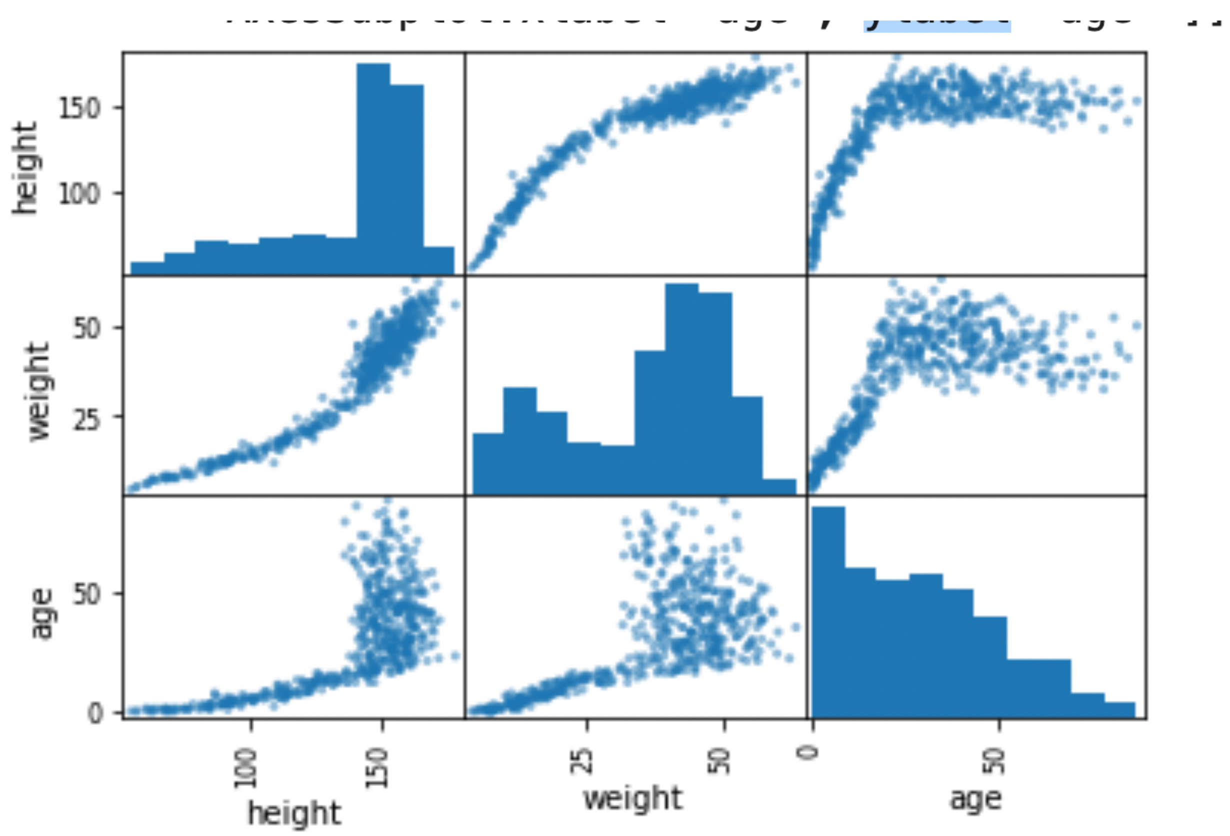
Quick Visualization.

In the next code cell, enter the following three lines of python code.

```
from pandas.plotting import  scatter_matrix
attributes = ['height', 'weight', 'age']
scatter_matrix(howell_full[attributes])
```

1. We will use a utility from pandas that creates a grid with scatter plots and histograms
2. We select the names of the features to compare and put them in a list.
3. We give the method a dataframe. We use [] to just select the features of interest. The diagonal contains histograms for that features.

The result should look like the following.



Submission 5 of 14: Progress mark (5)
Screen shot of graph

Submission 6 of 14: Analysis (15) from the results determine

11) Describe each distribution as skew, uniform, bimodal, or gaussian.
12) Look at the age-weight scatter plot and explain the character of the graph.
13) What does the age histogram tell us about this group?
14) How does the age histogram compare with that of people living in?
15) What two different features have the highest correlation?

Making Graphs

Now that we have a data frame, we want to use matplotlib to create graphs for us.

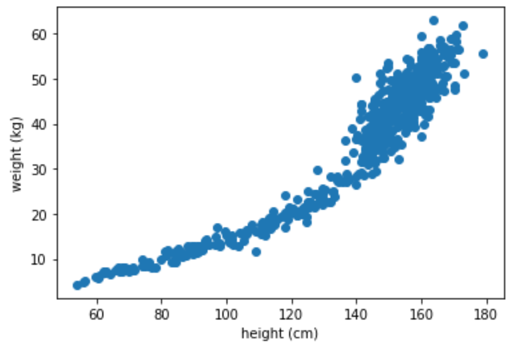
Make a new Code cell in your notebook after "A Better Graph" and add the following code:

```
import matplotlib.pyplot as plt
height = howell_full['height']
weight = howell_full['weight']

plt.scatter(height, weight)
plt.xlabel('height (cm)')
plt.ylabel('weight (kg)')
plt.show()
```

1. We want python plotting. We use plt as shorthand.
2. Grab the values for height
3. Grab the values for weight
4. We do a scatter plot with height on the x-axis and weight on the y-axis
5. Add a label to the x-axis. Give units
6. Add a label to the y-axis.
7. Once we have finished the construction of the plot, show it. We can now start another plot by doing a plt.scatter after the show.

When you run the code you should see something like

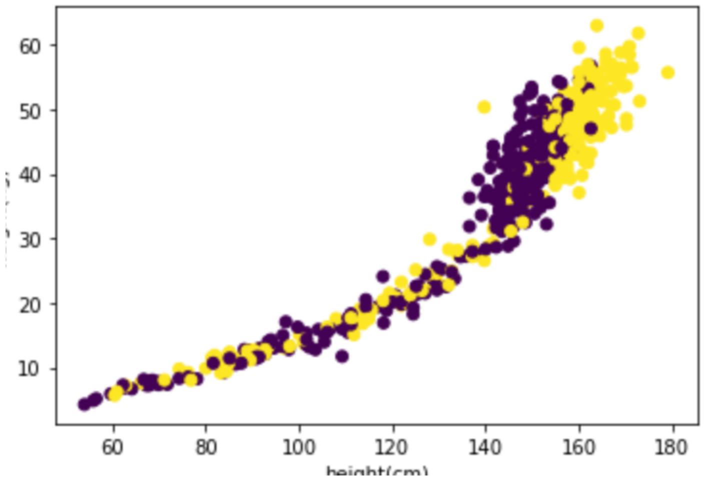


We would like to be able to distinguish male from female in this graph. Replace the plt.scatter line with the following two lines.

```
gender = howell_full['male']
plt.scatter(height, weight, c=gender)
```

1. Grab the values for gender.
2. Create a plot of height vs. weight again, but this time use the value of gender to color the point. Since we have 2 values for gender (0 and 1), Two colors will be chosen and displayed automatically.

When you run the code, you should see something like:



Create another plot where you display age vs height. Use gender for the color.

Submission 7 of 14: Code and Analysis (10)

1) Screen shot of age vs height scatter plot.
2) At what age does the character of the plot change?

Data Cleaning.

Look at the code in this section. The Howell data set doesn't have any missing data, so we don't need to clean it. We have seen some of these functions before, but I will make some comments.

```
howell_full['new'] = howell_full['male'] + howell_full['age']

howell_full.drop(axis='columns', labels='new', inplace=True)

height_median = howell_full['height'].median()
weight_average = howell_full['weight'].mean()

howell_full['height'].fillna(height_median, inplace=True)
```

1) We are creating a new feature with the name "new". We are using the existing attributes "male" and "age". These behave like arrays, so we add together the corresponding values to get the new ones.
2) We delete the attribute we just created
3) Get the height and compute the median value. This computation will work even if there are missing values.
4) Compute the average (mean) of the values in weight.

5) Replace any missing values in height with the median. If we wanted to replace the missing values with 0, we would use that as the first argument.

You may run this code or skip it as it should not affect the contents of the data frame.

Adding a new Feature.

We want to explore how we can create new features in our data frame. You saw an example in the last segment, but let's look at a fancier example.

Make a new Code cell in your notebook after "Adding a new Feature" and add the following code:

```
# Compute bmi based on height and weight metric units
def bmi(height, weight):
    return 10000*weight/(height**2)

print(bmi(150, 40))
new_feature = bmi(howell_full['height'], howell_full['weight'])
print(new_feature)
howell_full['bmi'] = new_feature
howell_full.head(n=10)
```

1 A comment in the code is always an option. We are making note that the formula is for the metric system.
2 Define a function with the name bmi and has two argument height and weight
3 We return 10000 times the weight divided by the height squared. Notice that this formula only does numpy methods and basic math operations. Also note that the code is indented. This along with the colon are important and show how lines are grouped together.
4 If we give the function just plain values, it computes the bmi and then we print it.
5 If we give the function two array-like things, the it automatically works in parallel on all the data instances.
6 We add in the new_feature data values into the frame under the feature name "bmi".

Run the code and you should see values for bmi in the data frame that are around 20.

Submission 8 of 14: Progress Mark (10)
Screen shot of the head of the data set with bmi.

Adding a new Categorical Feature.

Some times it is useful to group together instances into categories. We already have a categorical feature in our data set. The feature male only has two possible values. Age is the next closest thing we have in the Howell data set. Even so, I could group up the age groups into a decades worth of ages. So we would have groups 0-9, 10-19, 20-29, etc.

In this section, we will add in a classification based on BMI. The boundaries of the categories are somewhat arbitrary, but pretty standard. We have UnderWeight, Normal, OverWeight and Obese. Again we are going to build a function and we would like to rely on array processing as we did before. Unfortunately the IF statement does not work that way. We build up our function as before and then use the numpy method vectorize which creates a new function that will work on an array.

Make a new Code cell in your notebook after "Adding a Categorical Feature" and add the following code:

```
import numpy as np

def bmi_category(bmi):
    # bmi can only be a single value
    if bmi < 18.5: return 'Underweight'
    if bmi < 25.0: return 'Normal'
    if bmi < 30.0: return 'Overweight'
    return 'Obese'
```

1 The name of the function is bmi_category and it will take in a number and return the category.
2 The colon marks the start of the function code which all must be indented the same amount.
3 We check to see if the value is less than 18.5 and immediately return the value if it is. The value could have been numerical instead of a string. Using a string helps to understand what the category is, while learning methods might work better on a number.
4 We continue to pick off the categories, until only one is left which we just return.

Add the following code after the function:

```
    vector_bmi_category = np.vectorize(bmi_category)
    howell_full['bmi class'] = vector_bmi_category(howell_full['bmi'])

    print(howell_full.head(n=10))

    howell_full['bmi class'].value_counts()
```

1. We create a new function vector_bmi_category that will work on an array of BMI value and returns an array of the category values
2. We apply the new function to the feature BMI from the data set and make a new feature "bmi class".
3. We print the head so we can see that we have a new feature and can verify that the classification is correct based on the bmi.
4. The method value_counts allows us to see how many instances we have for each of the category values.

We have a potential problem. All instances except one are underweight or normal. The single overweight instance would cause a problem if we wanted to stratify on this feature. We need to have enough members in each category that they can be distributed over the split. We would either need to remove the lone instance or move it into a neighboring category (Normal in this case.)

Submission 9 of 14: Progress mark (10)
Screen shot of head and value counts

Make a new Code cell in your notebook after the one you were just working on. Create a graph on the full data set plotting age vs bmi. Use the gender for the color.

Submission 10 of 14: Progress mark (10)
Screen shot of age vs bmi scatter plot

Split the Data by Age

Since the character of the data varies based on age, we are going to want to split the data set into two parts: Children and Adults. We are going to somewhat arbitrarily going to use 18 as our cutoff. Depending on the goals of the model we are creating, we may use either of the splits or consider the entire data set.

Make a new Code cell in your notebook after "Split the Data by Age" and add the following code:

```
over18 = howell_full["age"] > 18
print(over18)

# Only keep the true instances
howell_adults = howell_full[over18]
howell_children = howell_full[~over18]       # ~ is not in numpy

print("There are ", len(howell_adults), " adult instances")
print(howell_adults)

print("There are ", len(howell_children), " child instances")
print(howell_children)
```

1 We get the feature for age which is an array. The greater than 18 is applied to each value in the array and over18 is now an array of true and false; one for each instance
2 Print it so we can see.
3 If we use square bracket access and give it an array of true/false, it will give us back a new data frame where only the instances that corresponded to true will remain.
4 Tilde flips true and false, so we get the remaining instances and a split is achieved
5 We print the number of instances in each of the reduced data sets.
6 We print each of reduced data sets. Only a small subset of the data frames will be displayed, but you should look at the results and verify that everything looks consistent.

Submission 11 of 14: Progress mark (10)
Screen shot of data set prints

Plot with masking

In this part we are going to do two plots on the same graph. This will give us more control over what is displayed and allow us to create a legend. We are also going to use masking to limit what gets plotted. We have seen that we can remove instances, but often we want to keep them around and just not use them. The masking ability in numpy allows us to do that. The values are still there, but have been masked and will be skipped. The masking notation is --.

We are going to do a plot on the adults using two masks. The first masked vector will be the male height values. The second masked vector will be the female height values.

Make a new Code cell in your notebook after "Plot with masking" and add the following code:

```
import matplotlib.pyplot as plt
male_height = np.ma.masked_where(howell_adults['male']==0,
                     howell_adults['height'])
female_height = np.ma.masked_where(howell_adults['male']==1,
                     howell_adults['height'])

weight = howell_adults['weight']
plt.scatter(male_height, weight, c='red', marker='+')
plt.scatter(female_height, weight, c='blue', marker='^')

plt.xlabel('weight')
plt.ylabel('height')
plt.legend(['Male', 'Female'])
plt.show()
```

1 We use the masked_where method from numpy. The first argument determines which values are masked. The second argument is what the masking is applied to. In this case, instances where the value for male is 0 (the females) will be masked. This is applied to height.
2 Similarly, we mask the males, leaving female heights unmasked.
3 We get all weights for the adults. There are no masks here, but we could add them if needed.
4 Make a scatter plot for the male instances. We set the color to red. We control the marker to be a plus.
5 Overlay a second scatter plot for the female instances. We set the color to blue and the marker to a triangle. There are other options for markers which you can look for in the documentation of matplotlib.
6 Create a legend with two labels (one for each of the plots)

For a little more work, we get more control over what is displayed.

Submission 12 of 14: Progress mark (10)
Screen shot of plot

Basic Train/Test split

As long as you don't need anything fancy like stratification or folds, the following code will get the job done.

Make a new Code cell in your notebook after "Train/Test Data split" and add the following code:

```
from sklearn.model_selection import train_test_split

train_set, test_set = train_test_split(howell_adults,
                        test_size=0.2, random_state=123)

print('Train size: ', len(train_set), 'Test size: ', len(test_set))

print('Adult counts', howell_adults['male'].value_counts())

print('Train counts', train_set['male'].value_counts())
print('Test counts', test_set['male'].value_counts())
```

1 Get the method
2 Use it to split the source (Howell_adults) into two data frames – for training and testing.
3 The rest of this is just showing what the splits are.

Submission 13 of 14: Computation (5)
Compute the ratio of male/female for
1) Adults data frame
2) Training data frame
3) Test data frame

Stratefied Train/Test split

We are going to use male as the target of our models next week, which suggests that we might want to stratify our split. Unfortunately, this is going to be a bit more complicated and involve the creation of object which will have a method we call to do the split. It is also complicated, by the fact that instead of producing data frames, we instead get an array of the indices in the split.

Make a new Code cell in your notebook after "Stratefied Train/Test Data split" and add the following code:

```
from sklearn.model_selection import StratifiedShuffleSplit

splitter = StratifiedShuffleSplit(n_splits=1, test_size=0.2,
                                  random_state=123)
for train_indices, test_indices in splitter.split(howell_adults,
                                        howell_adults['male']):
  train_set = howell_adults.iloc[train_indices]
  test_set = howell_adults.iloc[test_indices]

print('Train size: ', len(train_set), 'Test size: ', len(test_set))

print('Adult counts', howell_adults['male'].value_counts())

print('Train counts', train_set['male'].value_counts())
print('Test counts', test_set['male'].value_counts())
```

1 Get the splitter class (There are others too)
2 Create an object of the class. We are going to create a single split, test will have 20% of the instances and we pick a seed for the randomizer.
3 The split method will generate multiple splits as requested. Even though we only asked for one split, it will come as something to iterate over. So we use the for to do the iteration, which just gives a single split. The first argument of the split function is not critical as long as it has the right number of instances. The second gives the feature that we are going to stratify. We get back arrays of indices.
4 We create the training set by using iloc with [] holding the array of indices we want.)
5 Similarly create the test set.
6 We fall out of the loop and can now work on the train/test sets created.

7 Use it to split the source (Howell_adults) into two data frames – for training and testing.
8 The rest of this is just showing what the splits are.

Submission 14 of 14: Computation (5)

Compute the ratio of male/female for

4) Adults data frame
5) Training data frame
6) Test data frame