## Please write your name and netID:

Name: Mina Mohammadi
NetID:mm9494

# DS-UA 111: Lab 3

This lab is due Wednesday, March 4 by 8:00pm. Late labs, even by one minute, will be graded as zero, no exceptions. Improperly formatted labs also count as late. Note that the course academic honesty policy applies to every assignment, including this one. This lab is worth 19 points. Each lab is worth 2% of your grade.

## Instructions

Please complete your answers in the spaces provided. They will be either cells for code or Markdown, and we will make it clear within each question how to reply.

The submission process for this and all assignments is explained in separate documentation from Lecture 2.2.

## Question 1: Importing and inspecting a dataset

**(a) After our fascinating Lecture 5.1 on horses, you're now enthralled by the `horses.csv` data and can't wait to explore it. This is your lucky day! We've provided some starter code that gets the location of `horses.csv` and stores its filepath into a variable called `csv_fpath`. Use this filepath along with Pandas' `read_csv()` function to load the dataset into a DataFrame variable called `data`. In the last line of your code cell, call the DataFrame's `head` function to verify that all was imported properly.**

In [9]:
```
import pandas as pd
import os
csv_fpath = os.path.expanduser('~/shared/horses.csv')
data = pd.read_csv(csv_fpath)
data.head()
```

Out[9]:

| | name | price | sex | height | color | location | markings | weight | foaldat |
|---|---|---|---|---|---|---|---|---|---|
| **0** | Captain | 5000.0 | Gelding | 14.212 | Dun | Nantucket, Massachusetts | NaN | NaN | 4-Ma |
| **1** | Eternal Goodness | 8500.0 | Gelding | 16.205 | Chestnut | Brooklyn, Connecticut | NaN | NaN | 3-Ma |
| **2** | Dustys Fly Boy | 15000.0 | Gelding | 15.192 | Grulla | Dallas, Texas | NaN | 1200 pounds | 6-Ap |
| **3** | A FEDERAL HOLIDAY | 8500.0 | Mare | 14.999 | Grey | HOLSTEIN, Iowa | star, strip, & snip. 3 white socks. | NaN | 5-Ap |
| **4** | WIMPYS TRADITIONSTEP | 15000.0 | Gelding | 14.999 | Palomino | HOWELL, Michigan | NaN | 1000 pounds | 9-Ap |

In [ ]:
```
### BEGIN PUBLIC TESTS
assert 'data' in locals()
### END PUBLIC TESTS
```

**(b) Write one line of code (don't use `print` or `display`) to find out the type of each variable in the dataset.**

In [10]:
```
data.dtypes
```

Out[10]:
```
name                object
price              float64
sex                 object
height             float64
color               object
location            object
markings            object
weight              object
foaldate            object
registrations       object
disciplines         object
temperament        float64
dtype: object
```

In [ ]:
```
### BEGIN PUBLIC TESTS
assert isinstance(_, pd.Series)
### END PUBLIC TESTS
```

**(c) Create an object called `data_shape` that contains the number of columns and rows in your DataFrame `data`. Call `print` on `data_shape` and make sure you understand which part of the output represents the number of columns and which represents the number of rows.**

In [12]:
```
data_shape = data.shape
```

In [26]:
```
### BEGIN PUBLIC TESTS
assert 'data_shape' in locals()
assert type(data_shape) == tuple
### END PUBLIC TESTS
```

```
---------------------------------------------------------------------------
----
AssertionError                            Traceback (most recent call l
ast)
<ipython-input-26-d4ae2e98dd70> in <module>
      1 ### BEGIN PUBLIC TESTS
      2 assert 'data_shape' in locals()
----> 3 assert type(data_shape) == tuple
      4 ### END PUBLIC TESTS

AssertionError:
```

**(d) First, sort your dataframe by the height of the horses from smallest to largest, making sure you are sorting *in place* so that the sorting will persist in future code. Then use the `display` function (instead of the `print` function -- Jupyter will format your DataFrame nicely if you use `display` instead of `print`) on the DataFrame's `head`, to check that the sorting worked.**

In [13]:
```
data = data.sort_values(by=['height'])
display(data.head())
```

| | name | price | sex | height | color | location | markings | weight | foaldate | registrati |
|---|---|---|---|---|---|---|---|---|---|---|
| **362** | Bay Mare | 15000.0 | Mare | 0.015 | Bay | Algodones, New Mexico | Small Star | NaN | 4-Apr | AQI Amer Qu: H Associa (4 |
| **942** | DAR-L | 6000.0 | Stallion | 1.003 | Chestnut | WROCLAW, | NaN | NaN | 9-Apr | |
| **958** | Spirit | 375.0 | Gelding | 3.004 | Other | Milton, West Virginia | NaN | NaN | May-99 | |
| **717** | books and more!! | 25.0 | Stallion | 4.997 | Chestnut | ohio, Ohio | none | NaN | Mar-90 | |
| **581** | Little Kings Bay Mint Julep | 1200.0 | Mare | 4.999 | Bay | Columbia, Kentucky | none | NaN | 13-May | AMI Amer Minia H Associa |

In [ ]:
```
### BEGIN PUBLIC TESTS
assert data.iloc[0]["name"] == "Bay Mare"
assert data.iloc[-1]["name"] == "Gulliver"
### END PUBLIC TESTS
```

**(e) I am no horse expert, but some of these horses seem *really* small. If I think that these small heights are due to random errors, which of the following may have plausibly generated these errors?**

Type the letter of the option that reflects your answer:
A. This website is popular among circus and carnival owners who happen to have really small animals as part of their (very cruel) business model
B. The people who filled out the forms on equine.com/horses-for-sale typed in the heights wrong
C. People who have really small horses are far more likely to try to sell these useless (though adorable) creatures
D. Genetic mutations are real and sometimes horses are just tiny
E. None of the above

In [34]:
```
q1e_answer = 'B'
```

In [ ]:
```
### BEGIN PUBLIC TESTS
assert 'q1e_answer' in locals()
assert q1e_answer.upper() in ['A','B','C','D','E']
### END PUBLIC TESTS
```

**(f) Now re-sort your dataframe by the height of the horses from largest to smallest, making sure once again that the sort is being performed *in place*, then use `display` to show the first ten rows of the dataset.**

In [14]:
```
data = data.sort_values(by=['height'], ascending = False)
display(data.head(10))
```

| | name | price | sex | height | color | location | markings | weight | foaldate | reg |
|---|---|---|---|---|---|---|---|---|---|---|
| **236** | Gulliver | 16500.0 | Gelding | 18.200 | Chestnut | Coto de Caza, California | Blaze, flaxen mane and tail | NaN | May-96 | |
| **130** | Thunder | 10000.0 | Gelding | 18.184 | Chestnut | Hanover, Pennsylvania | Blaze | NaN | 2-May | |
| **421** | Ole Dobbin | 5500.0 | Gelding | 17.310 | Pinto | Utica, Ohio | NaN | 1300 pounds | 6-Apr | |
| **781** | Caesar | 6500.0 | Gelding | 17.302 | Grey | Blythewood, South Carolina | NaN | NaN | 2-Apr | |
| **424** | Bravadorro | 75000.0 | Gelding | 17.301 | Grey | London, Ontario, Canada | 4 high socks and a blaze | NaN | 6-May | S As |
| **457** | Stellar Moves | 35000.0 | Gelding | 17.300 | Bay | Sarasota, Florida | Star, strip, snip | 1800 pounds | 3-Mar | As |
| **679** | Royal Blue | 25000.0 | Gelding | 17.292 | Bay | Wainfleet, Ontario, Canada | 2 white socks and a star | 1500 pounds | 7-Jun | |
| **802** | Sail the Bay | 9000.0 | Gelding | 17.217 | Black | powhatan, Virginia | NaN | 1400 pounds | May-98 | |
| **767** | Ben | 25000.0 | Gelding | 17.193 | Grey | Newfane, Vermont | NaN | NaN | 1-Jan | |
| **291** | Jake & Flash | 4500.0 | Gelding | 17.187 | Grey | Stonewall, Manitoba, Canada | NaN | NaN | 1-May | |

In [ ]:
```
### BEGIN PUBLIC TESTS
assert data.iloc[0]["name"] == "Gulliver"
assert data.iloc[-1]["name"] == "Bay Mare"
### END PUBLIC TESTS
```

**(g) Again, while most of us are not horse experts, we are skilled at evaluating datasets. Based on the height values of the top ten largest horses, do you suspect there are random errors afoot with these upper values?**

Type the letter of the option that reflects your answer:
A. Yes, the numbers are definitely riddled with errors
B. No, the numbers are 100 percent without-a-doubt correct
C. It's always possible to have random errors in data, but these look within the bounds of what I would expect given the other values of height
D. There is no way to possibly have any idea whatsoever
E. None of the above

In [15]:
```
q1g_answer = 'C'
```

In [ ]:
```
### BEGIN PUBLIC TESTS
assert 'q1g_answer' in locals()
assert q1g_answer.upper() in ['A','B','C','D','E']
### END PUBLIC TESTS
```

**(h) Since currently the variable `weight` is a string variable, we can't do much numerical analysis on it. But we can explore the values it takes. Write a line of code that tells you how many of each weight group there are in this dataset.**

In [8]:
```
data['weight'].value_counts()
```

Out[8]:
```
1000 pounds    87
1100 pounds    58
1200 pounds    56
900 pounds     23
1300 pounds    20
800 pounds     18
1050 pounds    16
950 pounds     11
1500 pounds     9
1250 pounds     8
850 pounds      8
750 pounds      7
1400 pounds     6
1150 pounds     5
600 pounds      4
500 pounds      3
700 pounds      3
350 pounds      2
860 pounds      2
650 pounds      2
200 pounds      2
1001 pounds     1
150 pounds      1
1075 pounds     1
100 pounds      1
550 pounds      1
300 pounds      1
2000 pounds     1
1420 pounds     1
1225 pounds     1
1335 pounds     1
400 pounds      1
930 pounds      1
1800 pounds     1
1600 pounds     1
1550 pounds     1
250 pounds      1
1650 pounds     1
Name: weight, dtype: int64
```

In [ ]:
```
### BEGIN PUBLIC TESTS
assert "300 pounds" in _
### END PUBLIC TESTS
```

**(i) Now we are going to create a new dataframe using the table of horse weights we just created. Use the exact code you wrote in question 1h to create a dataframe called `horse_weights` (henceforth the greatest-named df in the history of pandas).**

In [15]:
```
horse_weights = data ['weight'].value_counts()
```

In [ ]:
```
### BEGIN PUBLIC TESTS
assert 'horse_weights' in locals()
### END PUBLIC TESTS
```

**(j) Next week we will use tables like this to create bar charts. In the meantime, we can work with `horse_weights` just like we would any other table. To see this, use the `shape` command to inspect its dimensions. What do you make of the output? (This is a rhetorical question, but please do think about it!)**

In [16]:
```
horse_weights.shape
```

Out[16]:
```
(38,)
```

In [ ]:
```
### BEGIN PUBLIC TESTS
assert type(_) == tuple
### END PUBLIC TESTS
```

# Question 2: A bit o' descriptive statistics!

**(a) We're going to continue working with the horses data (YAY! Or should I say "neigh!" ha ha ha sorry) to practice generating and interpeting descriptive statistics. Let's start by continuing to explore horse heights. Write a line of code that generates the mean of the variable `height` and name it `mean_height` . (Feel free to print `mean_height` to learn about it!)**

In [21]:
```
mean_height = data['height'].mean()
print(mean_height)
```

```
14.860920750782064
```

In [ ]:
```
### BEGIN PUBLIC TESTS
assert 'mean_height' in locals()
### END PUBLIC TESTS
```

**(b) Write a line of code that generates the median of the variable `height` and name it `median_height`. (Feel free to print `median_height` to learn about it! Feel free also to contemplate why this value might be different from `mean_height`.)**

In [23]:
```
median_height = data['height'].median()
print(median_height)
```

```
15.182
```

In [ ]:
```
### BEGIN PUBLIC TESTS
assert 'median_height' in locals()
### END PUBLIC TESTS
```

**(c) Write a line of code that generates the mode of the variable `height` and name it `mode_height`. Print the value of `mode_height`, and think about what the type of this variable is and why it might be this type instead of (for example) a numeric type**

In [24]:
```
mode_height = data['height'].mode()
print(mode_height)
```

```
0    15.206
dtype: float64
```

In [ ]:
```
### BEGIN PUBLIC TESTS
assert 'mode_height' in locals()
### END PUBLIC TESTS
```

**(d) I don't know about you, but I'm surprised we got a single value for the mode given that it's a continuous variable. Generate the `value_counts` for `height` to inspect whether the outcome for mode makes sense.**

In [17]:
```
data['height'].value_counts()
```

Out[17]:
```
15.206    8
14.982    7
14.282    7
16.117    6
15.988    6
         ..
16.995    1
14.306    1
14.113    1
12.090    1
14.213    1
Name: height, Length: 519, dtype: int64
```

In [ ]:
```
### BEGIN PUBLIC TESTS
assert isinstance(_, pd.Series)
### END PUBLIC TESTS
```

**(e) Does the result you got for `mode` make sense in light of the value counts?**

**Write the letter that best reflects your answer:**
**A. Heck yes it makes sense!**
**B. No, I suspect the mode is wrong**
**C. I hate this**

In [18]:
```
q1e_answer = 'A'
```

In [ ]:
```
### BEGIN PUBLIC TESTS
assert 'q1e_answer' in locals()
assert q1e_answer.upper() in ['A','B','C']
### END PUBLIC TESTS
```

**(f) Now that we know the mean, we are intrigued and want to explore further. Use Pandas' `describe` function (along with `print`) to print some descriptive statistics about the `height` column. On a second line in the same cell, create an object called `height_stdev` and manually give it the value of the standard deviation of `height` to two decimal places of accuracy.**

In [19]:
```
print(data.describe())
height_stdev = round(data['height'].std(),2)
```

```
                price      height  temperament
count      959.000000  959.000000   959.000000
mean      7439.958290   14.860921     3.402208
std      13278.614627    1.836368     1.513152
min          0.000000    0.015000     1.005000
25%       1500.000000   14.287000     2.216000
50%       4000.000000   15.182000     3.219000
75%       8500.000000   16.006000     4.286000
max     180000.000000   18.200000     9.000000
```

In [37]:
```
### BEGIN PUBLIC TESTS
assert 'height_stdev' in locals()
### END PUBLIC TESTS
```

**(g) Find the variance of `height` by using your newly created object `height_stdev`, and call this `height_var`. Print `height_var` to check the output.**

In [20]:
```
height_var = float(height_stdev*height_stdev)
print(height_var)
```

```
3.3856
```

In [ ]:
```
### BEGIN PUBLIC TESTS
assert 'height_var' in locals()
assert type(height_var) == float
### END PUBLIC TESTS
```

**(h) Now let's do something that requires several steps: We want to find the mean height of the horses with the most common color. First, write a single line of code (using a Pandas function we have already used above) to display the most common color(s) of horse.**

In [21]:
```
data['color'].value_counts()
```

Out[21]:
```
Bay              278
Chestnut         147
Grey             100
Sorrel            93
Black             89
Palomino          45
Buckskin          41
Pinto             36
Brown             32
Dun               31
Roan              24
Other             19
White              8
Cremello           6
Silver Dapple      4
Grulla             4
Champagne          1
Perlino            1
Name: color, dtype: int64
```

In [ ]:
```
### BEGIN PUBLIC TESTS
assert isinstance(_, pd.Series)
### END PUBLIC TESTS
```

**(i) Now write one line of code to create a new DataFrame `data_popcolor` containing the subset of `data` for which the horse's `color` is the most common color. Make sure you use Pandas' `copy` function so that `data_popcolor` is a separate DataFrame from (rather than a view of) the original `data` variable. Write a second line of code using the `display` function to show the first 5 rows of your new DataFrame to make sure it worked as expected.**

In [22]:
```
data_popcolor = data[data.color == 'Bay'].copy()
display(data_popcolor.head(5))
```

| | name | price | sex | height | color | location | markings | weight | foaldate | registr |
|---|---|---|---|---|---|---|---|---|---|---|
| 457 | Stellar Moves | 35000.0 | Gelding | 17.300 | Bay | Sarasota, Florida | Star, strip, snip | 1800 pounds | 3-Mar | A Am Q Assoc |
| 679 | Royal Blue | 25000.0 | Gelding | 17.292 | Bay | Wainfleet, Ontario, Canada | 2 white socks and a star | 1500 pounds | 7-Jun | |
| 254 | Onxy a True Gem | 2500.0 | Mare | 17.112 | Bay | Delton, Michigan | none | 1300 pounds | Jan-00 | |
| 345 | Urama *Fence Dancers Sport Horse Farm* | 175000.0 | Mare | 17.091 | Bay | Pilesgrove, New Jersey | 4 white socks | NaN | 2-Feb | K\ Warm Studb A |
| 370 | Doc's Option | 4500.0 | Gelding | 17.083 | Bay | Johnstown, Ohio | Star, front left pastern, hind left sock | NaN | Mar-00 | JC - J ( |

In [ ]:
```
### BEGIN PUBLIC TESTS
assert 'data_popcolor' in locals()
### END PUBLIC TESTS
```

**(j) Finally, write one line of code utilizing your `data_popcolor` DataFrame to find the mean of the horses that are the most common color. Store that value as `popcolor_mean`. (Of course, feel free to print it to see the result of your work!)**

In [23]:
```
popcolor_mean = data_popcolor.mean()
print(popcolor_mean)
```

```
price          7982.176259
height           15.121777
temperament       3.564460
dtype: float64
```

In [ ]:
```
### BEGIN PUBLIC TESTS
assert 'popcolor_mean' in locals()
### END PUBLIC TESTS
```

# Lab Complete! Now make sure to submit here on JupyterGub and also download as HTML, convert to PDF, and submit the PDF to Gradescope

In [ ]:

```
```