Assignment 1

##Exercise 1.

- Code explanation: by using an inner function - range_temp_function(x), the temp_tester checks the temperature in the range +/-1 degree of either a human or chicken temperature. Furthermore, the inner function allows us to print the result as either True for not having a fever or False to indicate the temperature is either too low or too high.
- Testing: for the testing purpose the human temperature was set to 37, and the chicken's was 41.1. The code below shows the testing process

```
#Tesing code:
human_tester = temp_tester(37)
chicken_tester = temp_tester(41.1)

chicken_tester(42)
human_tester(42)
chicken_tester(43)
human_tester(35)
human_tester(98.6)
```

##Exercise 2.

> The dataset cointains are 4 columns and 152361 rows. Thus there 152,361 people in the pupulation with columns names

1. name
2. age
3. weight
4. eyecolor

#Age

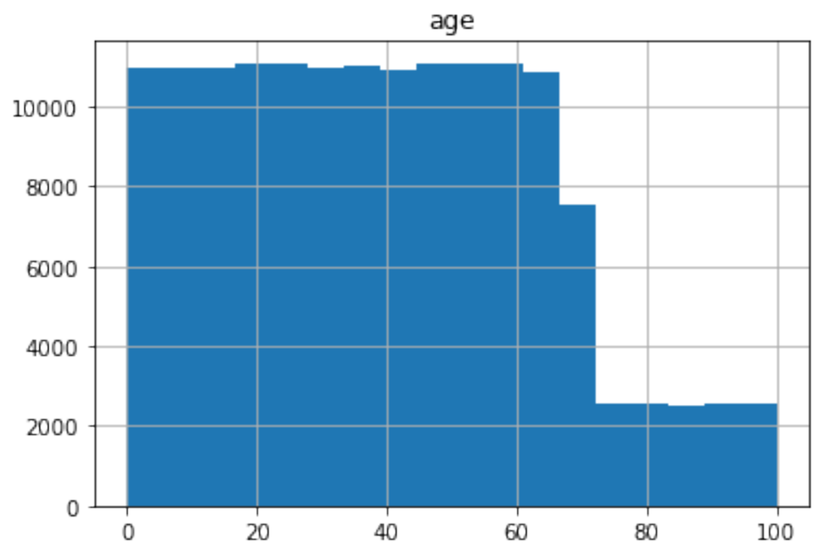
- Mean = 39.510528
- Standard Deviation = 24.152760
- Minimum = 0.000748
- Maximum = 99.991547

> Function data. describe() a variable was used to find out the statistical description of the varaible.

> Each bin is plotted as a bar whose height corresponds to how many data points are in that bin. Thus, we could use Sturges' Rule to calculate the optimal number of bins to use in a histigram:

> Optima Bins = [$\log_2(n) + 1$]

```
log2 = math.log2(152361)+1
log2
```

> Thus, showing that the optima number of bins would be 18.
>
> 
>

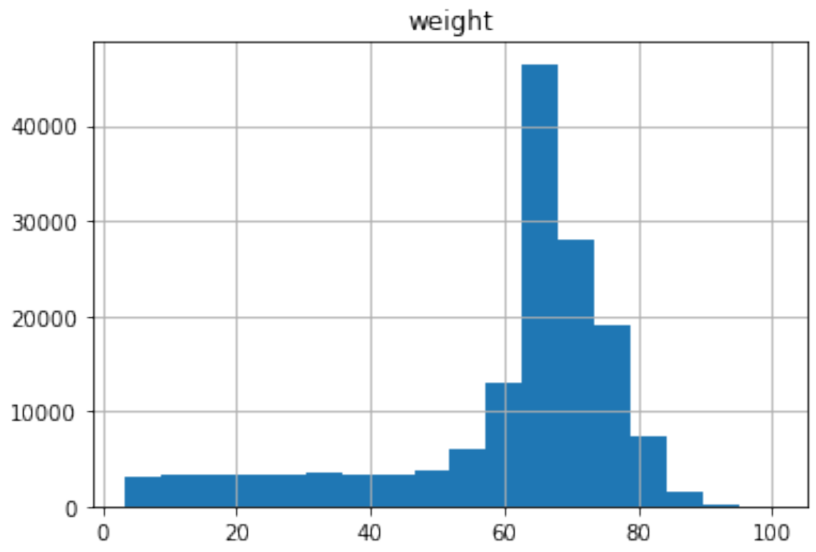
> Furthermore, from the graph, it could be noted that the distribution is

> somewhat not normal and skewed to the right. Plus, it could be noted that there are extrim drops in age counts when it comes to 60 and then an extreme jump dowm to 80. And it could be noted that the graph was constructed using data.hist function that allows to build of histograms depending on the variables.

#Weight

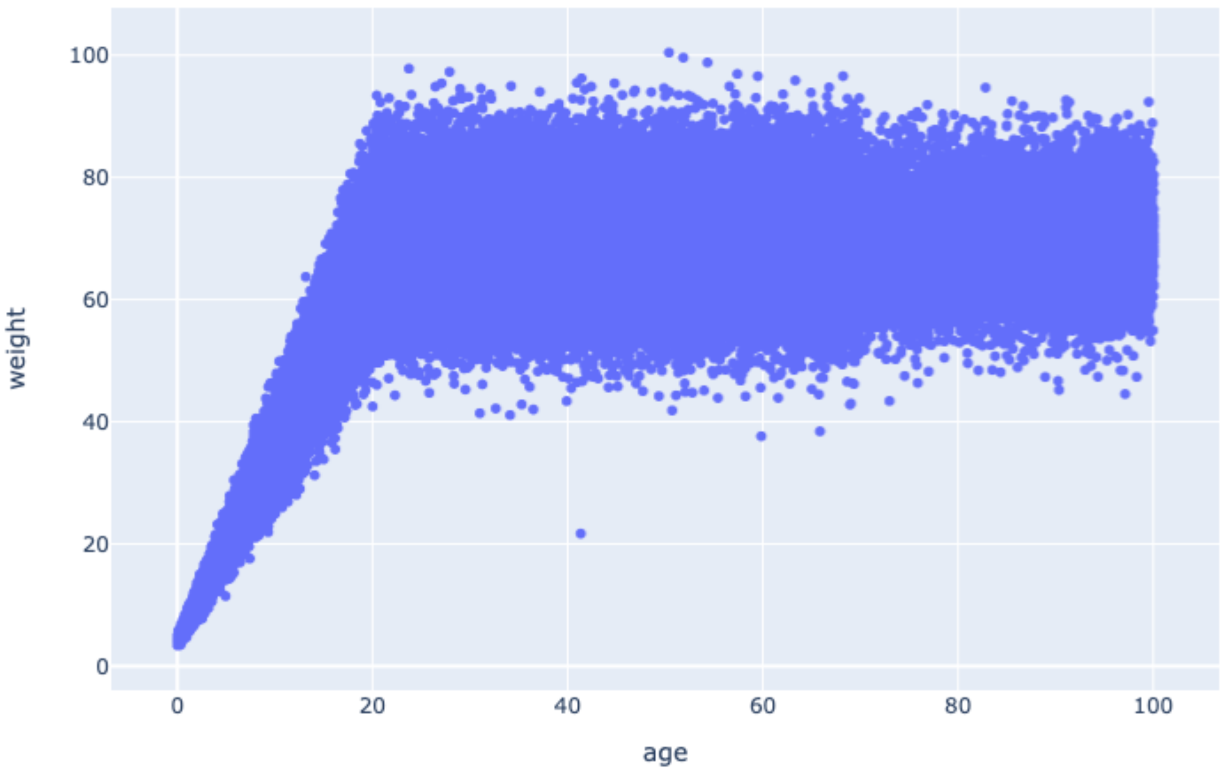
- Mean = 60.884134
- Standard Deviation = 18.411824
- Minimum = 3.382084
- Maximum = 100.435793

> Function data. describe() a variable was used to find out the statistical description of the varaible.



> The graph indicates a skewed to the left side histogram, as most of the values are on the right side of the graph. Thus, indicating that there are fewer people that weight then 60. Furthermore, an extreme drop coudl be noted around 70-pound weight, and some of the outliers are located near a 100 and a 0. And it could be noted that the graph was constructed

> using data.hist function that allows to build of histograms depending on the variables.



> The scatter plot has a positive correlation to zero correlation as the graph indicates to have both a vertical and horizon line. Thus, age and weight have somewhat of a weak relationship and are closer to not having a relationship with each other at all. To build the scatter plot `import plotly.express as px` was used to onstruct a scatter plot graph.

> Outlier:

- count: 537
- name: Anthony Freeman
- age: 41.3
- weight: 21.7

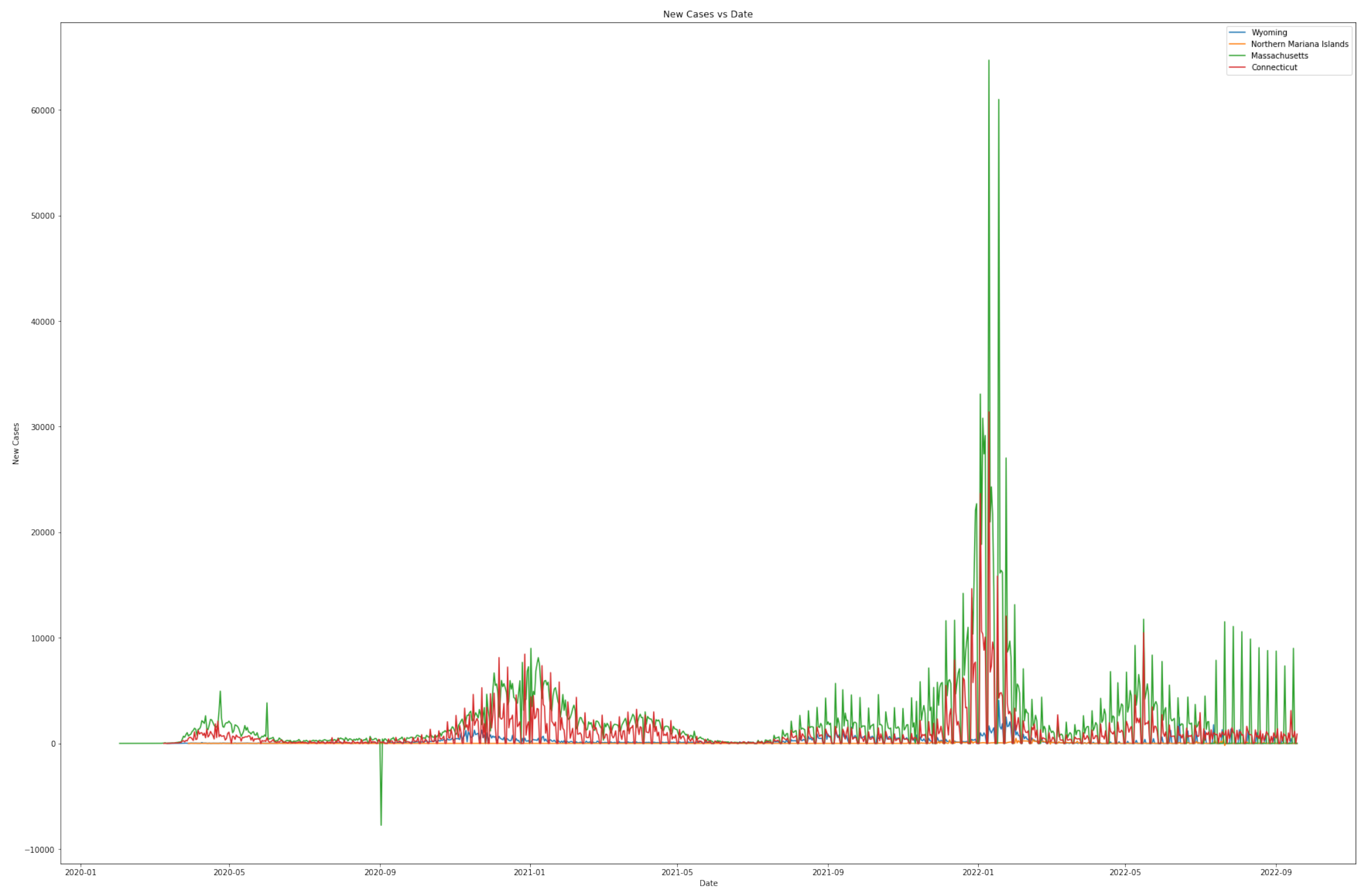
> The outlier was found using the plotly histogram that allows you to hover on the scatter plot dots and see their exact x and y values, thus by looking at the outlier on the closer to the bottom of the graph, his x and y values were given. Then, to print his name and excat patient id, a print function was used containing the exact values for the age and weight that were noted by hovering over the outlier. Futhermore, to make sure for sure about outliers another filter was set to print the names of patients whose age is more than 20, and weights either less then 40 or more than a 100.

##Exercise 3 This data was taken GitHub Pages. License: https://github.com/nytimes/covid-19-data

To create the graphs first we had to convert the 'Date' column to DateTime format using the pandas to_datetime function. Then, the function to plot states new cases vs date. Creating a new dataset df_new, that stores the new cases allows us to use the function .diff(), which finds the first discrete difference of objects over the given cases. Then, the new dataset set is graphed using dates as the x-values, and cases as the y-values. Plus, to make sure it was readable different colors are assigned automatically to each state when graphed.

> Testing the graph:

```
# Example to test plot function
state_list = ['Wyoming', 'Northern Mariana Islands', 'Massachusetts','Connect
plot(state_list)
```



To create a function that takes the name of a state and returns the date of its highest number of new cases, we used a similar method as the one used for creating plots in the above example. We used to define new data and new variables that contain values for a maximum number of cases using the max() function. Furthermore, to insure that we only got one date for the maximum number of cases for each state a date.iloc[0] was used.

> Testing the function:

```
print(highest_case('Washington'))
print(highest_case('Illinois'))
print(highest_case('Massachusetts'))
```

2022-01-18 00:00:00 2022-01-18 00:00:00 2022-01-10 00:00:00

Following the method used in the prior example above a similar function was created using the highest_case function from above to calculate which state has the highest number of new cases when compared. Furthermore, the if and else statement was used to print which state has the highest number of daily new cases by a number of days, which were calculated using the abs() function and looking at the difference between two dates (of two different states).

> Testing the function:

```
peak('Massachusetts', 'Connecticut')
peak('Florida', 'Washington')
peak('Ohio','Montana')
```

```
Massachusetts and Connecticut had its highest number of daily new cases on 202
Florida had its highest number of daily new cases by 14 days
Ohio had its highest number of daily new cases by 9 days
```

##Exercise 4

```
import xml.etree.ElementTree as ET
from pprint import pprint as pp
tree = ET.parse('/Users/polina/Desktop/desc2022.xml')
root = tree.getroot()
```

The function to find UI was built by first founding the first element of the child in DescriptorUI, by accessing an individual child. Then, the function run through all the root's children, and by knowing the individual child root it was easier to write an if a function that would find a specific UI, which in our case was DescriptorUI 'D007154'.

> Testing the function:

```
print(find_ui('D007154'))
Immune System Diseases
```

A similar function was used to find the 'Nervous System Diseases', as the only thing that has changed is was instead of finding UI it found a DescriptorName.

> Testing the function:

```
print(find_name('Nervous System Diseases'))
D009422
```

The function is build in the similar way as the ones above, where first function finds the Treenumber given either a DescriptorName or DescriptorUI. Also, to make it easier for the function, since all of the uis begin with the letter D we could specify that in the loop to make the function look for the "D" uis.Then it runs throught a loop looking for the DescriptorName or DescriptorUI.

> Testing the function:

```
print(treeNumber_find('Nervous System Diseases'))
print(treeNumber_find('D007154'))

C10
C20
```

> This shows that both diseases play the main disease in which there could

> be most of the descendants are due to either a nervous system and/or immune system disease.

Then to find the neighboring descendants we could write a similar function that would find the Treenumber for each of the names/UI. Plus, to find parents we could first write two new variables that would allow us to get the tree names for each of the names/UI.

> Testing the function:

```
print(descendents_common('Nervous System Diseases', 'D007154'))
```

```
{'AIDS Dementia Complex', 'Neuritis, Autoimmune, Experimental', 'Vasculitis, (
```

The above search has found neighboring diseases using the MeSH hierarchy that are part of the Nervous System Diseases and/or Immune System Diseases (D007154). Thus, diseases that are shown have a descendant relationship with the Nervous System Diseases and/or Immune System Diseases, which are also shown in hierarch order (meaning the one that has more connection to either or disease are shown first).

Assignment 1

November 17, 2022

Exercise 1:

```
[ ]: def temp_tester(temp):

      def range_temp_function(x):
         if x<= temp + 1 and x >= temp - 1:
             return True
         else:
             return False

      return range_temp_function
```

```
[ ]: human_tester = temp_tester(37)
     chicken_tester = temp_tester(41.1)
```

```
[ ]: # Example to test plot function
     print(chicken_tester(42))
     print(human_tester(42))
     print(chicken_tester(43))
     print(human_tester(35))
     print(human_tester(98.6))
```

Exercise 2:

```
[ ]: import pandas as pd
     import sqlite3
     with sqlite3.connect("/Users/polina/Desktop/hw1-population.db") as db:
         data = pd.read_sql_query("SELECT * FROM population", db)
         data.head
```

```
[ ]: data.head
```

```
[ ]: #Age
     data.describe()[["age"]]
```

```
[ ]: ! conda install -c conda-forge plotnine -y
```

```
[ ]: from plotnine import *
[ ]: #Find bin width
     import math
     log2 = math.log2(152361)+1
     log2
```

```
[ ]: data.hist(column = 'age', bins = 18)
```

```
[ ]: #Weight
     data.describe()[["weight"]]
```

```
[ ]: data.hist(column = 'weight', bins = 18)
```

```
[ ]: import plotly.express as px
     fig = px.scatter(data, x="age", y="weight")
     fig.show()
```

```
[ ]: # Filter to detect outlier
     data[(data['age'] > 20) & ((data['weight'] < 40)|(data['weight'] > 100))]
```

```
[ ]: # Filter for Anthony Freeman
     data[(data['age'] == 41.3) | (data['weight'] == 21.7)]
```

Exercise 3:

```
[ ]: import numpy as np
     from matplotlib import pyplot as plt
     import pandas as pd
     url = 'https://raw.githubusercontent.com/nytimes/covid-19-data/master/us-states.
      ↪csv'
     df = pd.read_csv(url)
     df.head()
```

```
[ ]: # convert the 'Date' column to datetime format
     df['date']= pd.to_datetime(df['date'])
     # Check the format of 'Date' column
     df.info()
```

```
[ ]: #look for new cases
     (df
      .groupby(['state'])
      .cases
      .diff()
     )
```

```
[ ]: def plot(state_list):
         plt.figure(figsize=(30,20))
         for state in state_list:
             df_new = df[df['state']==state]
```

```
        df_new['cases'] = df_new['cases'].diff()
        plt.plot(df_new['date'], df_new['cases'], label=state)
    plt.legend()
    plt.title('New Cases vs Date')
    plt.xlabel('Date')
    plt.ylabel('New Cases')
    plt.show()
```

[ ]:
```
# Example to test plot function
state_list = ['Wyoming', 'Northern Mariana Islands', 'Massachusetts',␣
 ↪'Connecticut']
plot(state_list)
```

[ ]:
```
def highest_case(state):
    df_new_date = df[df["state"] == state]
    df_new_date['cases'] = df_new_date['cases'].diff()
    date = df_new_date[df_new_date['cases'] == np.
 ↪max(df_new_date['cases'])]['date']
    return date.iloc[0]
```

[ ]:
```
# Example to test plot function
print(highest_case('Washington'))
print(highest_case('Illinois'))
print(highest_case('Massachusetts'))
```

[ ]:
```
def peak(state1, state2):
    df_peak_1 = (highest_case(state1))
    df_peak_2 = (highest_case(state2))

    if df_peak_1 >  df_peak_2:
        print(state2, 'had its highest number of daily new cases by',␣
 ↪abs((df_peak_1 - df_peak_2).days, 'days'))
    elif df_peak_1 < df_peak_2:
        print(state1, 'had its highest number of daily new cases by',␣
 ↪abs((df_peak_1 - df_peak_2).days), 'days')
    else:
        print(state1, 'and', state2, 'had its highest number of daily new cases␣
 ↪on', df_peak_1)
```

[ ]:
```
# Example to test plot function
peak('Massachusetts', 'Connecticut')
peak('Florida', 'Washington')
peak('Ohio','Montana')
```

Exercise 4:

[ ]:
```
import xml.etree.ElementTree as ET
from pprint import pprint as pp
tree = ET.parse('/Users/polina/Desktop/desc2022.xml')
root = tree.getroot()
```

[ ]:
```
pp(root)
```

[ ]:
```
desk1 = root[0]
desk1_xml = ET.tostring(desk1)
print(desk1_xml)
```

[ ]:
```
ET.indent(desk1)
print(ET.tostring(desk1).decode('utf-8'))
```

[ ]:
```
#Select the child by using its tag name
Descriptor1 = desk1.find("DescriptorUI")
pp(Descriptor1)
```

[ ]:
```
ui_1 = Descriptor1.text
pp(ui_1)
```

[ ]:
```
def find_ui(ui):
    for child in root:
        if child[0].text == ui:
            return child[1][0].text
```

[ ]:
```
print(find_ui('D007154'))
```

[ ]:
```
def find_name(name):
    for child in root:
        if child[1][0].text == name:
            return child[0].text
```

[ ]:
```
print(find_name('Nervous System Diseases'))
```

[ ]:
```
def treeNumber_find(name):
    if not 'D0' in name:
        name = find_name(name)
    for child in root:
        if child[0].text == name:
            for record in child.iter('TreeNumberList'):
                return record[0].text
```

[ ]:
```
print(treeNumber_find('Nervous System Diseases'))
print(treeNumber_find('D007154'))
```

[ ]:
```
def descendents_common(descendents1, descendents2):



    descendentsTree1 = treeNumber_find(descendents1) + '.'
    descendentsTree2 = treeNumber_find (descendents2) + '.'

    answer = set()  ## empty set

    for child in root:
        for record in child.iter('TreeNumberList'):
            for treeNumber in record:
                if descendentsTree1 in treeNumber.text:
                    for treeNumber in record:
                        if descendentsTree2 in treeNumber.text:
                            answer.add(child[1][0].text)
    return answer
```

[ ]:
```
print(descendents_common('Nervous System Diseases', 'D007154'))
```