Instructions

All python libraries used are listed on the top of each Jupyter notebook file.

Answers

Exercise1:

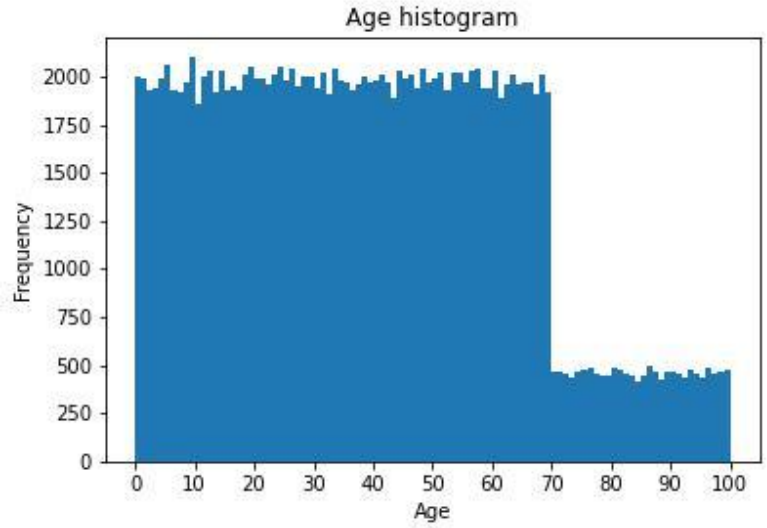
No question needs to be answered. Test code results are the same with examples.

Exercise2:

1. Examine data
   It has columns labelled: 'name', 'age', 'weight', 'eyecolor'. It has 152361 rows.

2. Examine the distribution of Age
   Mean(age)= 39.510527927396524
   Standard deviation(age)= 24.152760068601445. Note that the Std is normalized by N-1, rather than by N. Standard deviation(age) by N = 24.152680806849094
   Minimun(age)= 0.0007476719217636152
   Maximun(age)= 99.99154733076972

   Plot a histogram with 100 bins to show the frequency of people in each round age.

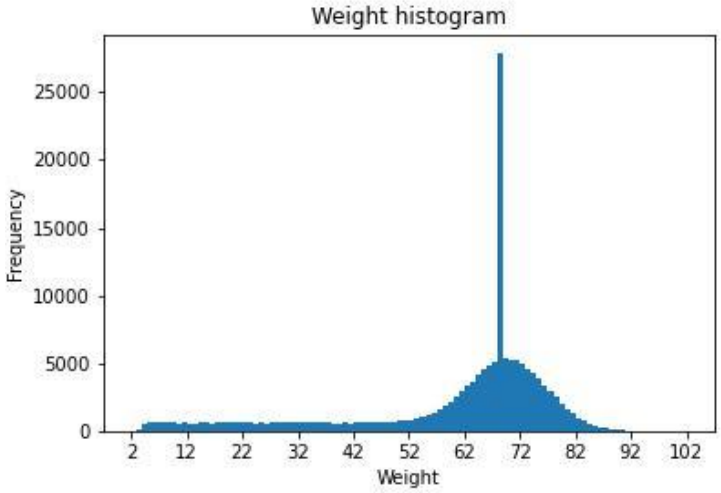


   The Pattern is that the number of people over 70 years old is obviously smaller than that of people under 70.

3. Examine the distribution of Weight
   Mean(weight)= 60.88413415993031

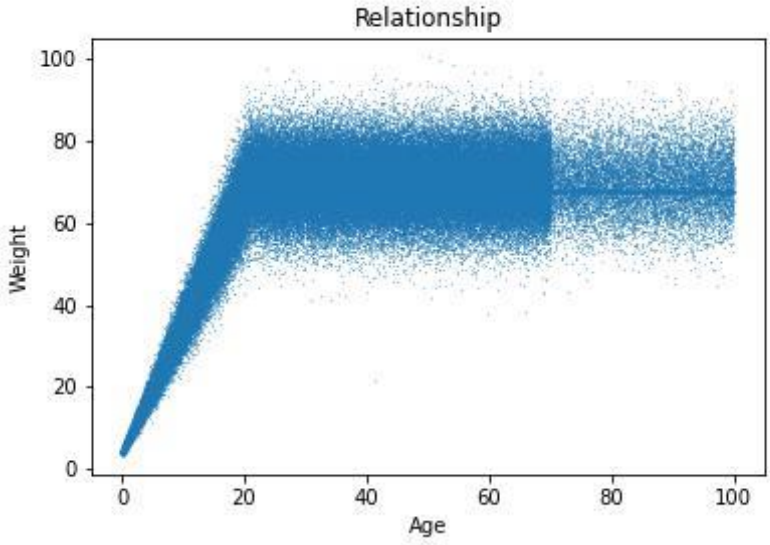
Standard deviation(weight)= 18.411824265661494. Note that the Std is normalized by N-1, rather than by N. Standard deviation(weight) by N = 18.411763843852512
Minimun(weight)= 3.3820836824389326
Maximun(weight)= 100.43579300336947

Plot a histogram with 100 bins to show the frequency of people in each round weight.



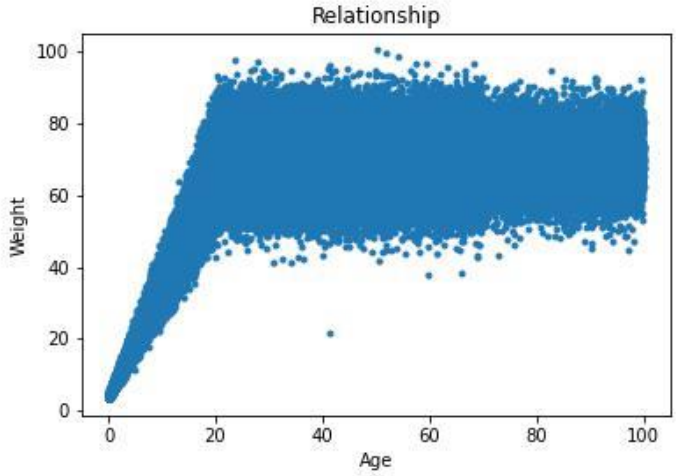
The Outlier is that the number of people around 68~69 kg is obviously larger than that of other people in other bins. The datapoints of people weighing from 50 to 90 kg seems to obey a Gaussian distribution.

4. Relationship between Weight and Age



As the scatterplot shows, for people younger than about 25 years old, Weight is approximately proportional to Age. We can assume $\widehat{Weight} = a * Age$. For

people over 25 years old, Weight seems to be normally distributed, regardless of Age. In other words, Weight is independent from Age and they are unrelated.

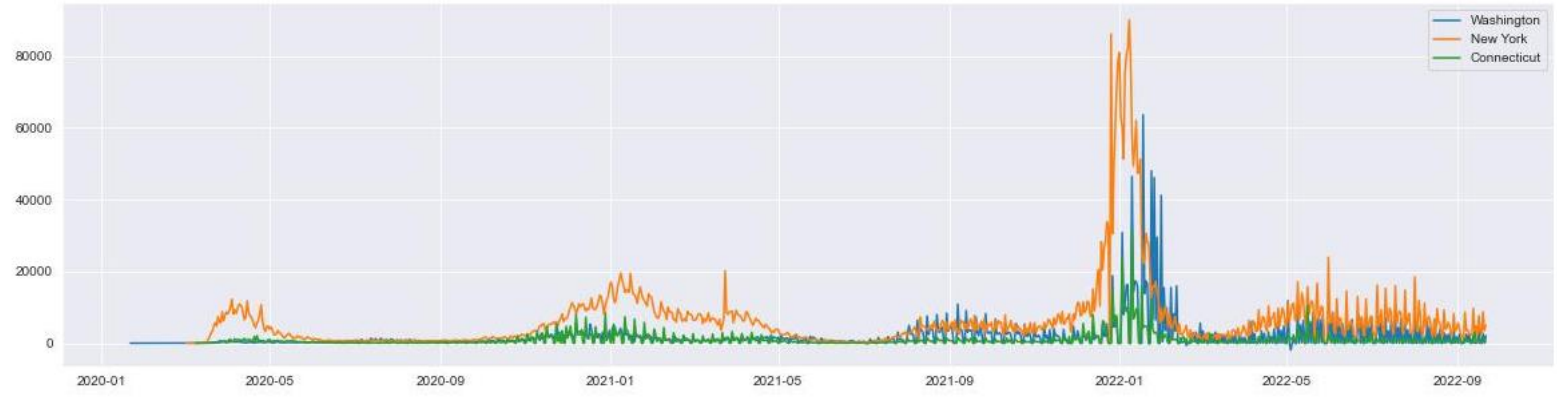


Make the scatterplot point size larger and we can see there is an outlier above 40 years old and weighing below 35 kg. The Outlier's name is Anthony Freeman. We use conditional filter to select this row from the dataframe, which is print(data[(data.age>40) & (data.weight<35)]).

Exercise 3:

The COVID-19 dataset is downloaded on Sep 22, 2022 from The New York Times's GitHub at https://raw.githubusercontent.com/nytimes/covid-19-data/master/us-states.csv.

1. Function to plot new cases v.s. date of several states
   Colors, labels and a legend are used to plot new cases.
   The generic function plot_state_newcase(states) is tested and an example of Washington, New York and Connecticut is shown below.



2. Function to return the date of a state's highest number of new cases
   The generic function: find_highest_date(statename) is tested and the highest new cases's corresponding date of Washington is 1/18/2022.

3. Function to compare 2 states' new cases and calculate days between peaks
   The generic function: compare_states(state1,state2) is tested and an Example of Connecticut and Washington is shown: Washington's daily new case peak is higher than that of Connecticut. There are 8 days between their peaks.

Exercise 4:

1. The DescriptorName associated with DescriptorUI D007154 is Immune System Diseases.
2. The DescriptorUI associated with DescriptorName "Nervous System Diseases" is D009422.

3. The 35 DescriptorNames of descendants of both are: {'Kernicterus', 'Neuritis, Autoimmune, Experimental', 'Neuromyelitis Optica', 'Giant Cell Arteritis', 'Polyradiculoneuropathy', 'AIDS Arteritis, Central Nervous System', 'Nervous System Autoimmune Disease, Experimental', 'Lambert-Eaton Myasthenic Syndrome', 'Myasthenia Gravis, Autoimmune, Experimental', 'Demyelinating Autoimmune Diseases, CNS', 'Multiple Sclerosis, Chronic Progressive', 'Guillain-Barre Syndrome', 'Myasthenia Gravis', 'Myasthenia Gravis, Neonatal', 'Mevalonate Kinase Deficiency', 'Microscopic Polyangiitis', 'Encephalomyelitis, Acute Disseminated', 'Autoimmune Hypophysitis', 'Diffuse Cerebral Sclerosis of Schilder', 'Polyradiculoneuropathy, Chronic Inflammatory Demyelinating', 'POEMS Syndrome', 'Vasculitis, Central Nervous System', 'Lupus Vasculitis, Central Nervous System', 'AIDS Dementia Complex', 'Autoimmune Diseases of the Nervous System', 'Stiff-Person Syndrome', 'Encephalomyelitis, Autoimmune, Experimental', 'Miller Fisher Syndrome', 'Ataxia Telangiectasia', 'Multiple Sclerosis', 'Uveomeningoencephalitic Syndrome', 'Leukoencephalitis, Acute Hemorrhagic', 'Myelitis, Transverse', 'Multiple Sclerosis, Relapsing-Remitting', 'Anti-N-Methyl-D-Aspartate Receptor Encephalitis'}

4. The above search finds that these DescriptorNames are both Immune System Diseases and Nervous System Diseases. They are specific branches of immune and nervous system diseases.
5. A generic function, find_by_UI(UI), is written for Question1
6. A generic function, find_by_name(name), is written for Question2
7. A generic function, find_intersection(UI1,UI2), is written for Question3

Appendix

Code for Exercise1:

```
def temp_tester(normal_temp):

    def test_norm(real_temp):
```

```
        if abs(normal_temp - real_temp) <= 1:
            return True
        else:
            return False

    return test_norm


'''This is the Test code'''
human_tester = temp_tester(37)
chicken_tester = temp_tester(41.1)

print(chicken_tester(42))  # True -- i.e. not a fever for a chicken
print(human_tester(42))  # False -- this would be a severe fever for a human
print(chicken_tester(43))  # False
print(human_tester(35))  # False -- too low
print(human_tester(98.6))  # False -- normal in degrees F but our reference temp was in degrees C
```

Code for Exercise2:

```
import pandas as pd
import numpy as np
import sqlite3
import matplotlib.pyplot as plt
with sqlite3.connect("hw1-population.db") as db:
    data = pd.read_sql_query("SELECT * FROM population", db)
'''Examine the data'''
# print(data)
print(data.columns) #columns
```

```
print(len(data)) #number of rows
'''Examine the distribution of Age'''
Age=data.age
print(Age.mean())
print(Age.std()) #normalized by N-1, ddof=0 is normalized by N
# print(Age.std(ddof=0))
print(Age.min())
print(Age.max())
agehist=plt.hist(Age,bins=100,range=(0,100))
plt.title('Age histogram')
plt.xlabel('Age')
plt.ylabel('Frequency')
x=range(0,101,10)
plt.xticks(x)
plt.savefig('AgeHistogram.jpg')
plt.show()
# print(agehist)
'''Examine the distribution of Weight'''
Weight=data.weight
print('mean',Weight.mean())
print('std',Weight.std()) #normalized by N-1, ddof=0 is normalized by N
# print(Weight.std(ddof=0))
print('min',Weight.min())
print('max',Weight.max())
weighthist=plt.hist(Weight,bins=100,range=(2,102))
plt.title('Weight histogram')
plt.xlabel('Weight')
plt.ylabel('Frequency')
```

```
x=range(2,103,10)
plt.xticks(x)
plt.savefig('WeightHistogram.jpg')
plt.show()
# print(weighthist)
print('outlier index(start from 2kg)',np.argmax(weighthist[0])) #find the outlier
# print(weighthist[0][66])
'''Relationship between weight and age'''
plt.scatter(Age,Weight,marker='.',s=0.1)
plt.title('Relationship')
plt.xlabel('Age')
plt.ylabel('Weight')
plt.savefig('Relationship.jpg')
plt.show()
'''Find outlier'''
plt.scatter(Age,Weight,marker='.')
plt.title('Relationship')
plt.xlabel('Age')
plt.ylabel('Weight')
plt.savefig('Outlier.jpg')
plt.show()

# print((data.age>40) & (data.weight<35))
# print(sum((data.age>40) & (data.weight<35)))
print(data[(data.age>40) & (data.weight<35)])
```

Code for Exercise3:

```
import pandas as pd
```

```
import numpy as np
from dateutil.parser import parse as parse_date
import matplotlib.pyplot as plt

data = pd.read_csv("us-states.csv")
print(data.head())
'''plot a list of states and their new cases'''
def plot_state_newcase(states):
    plt.figure(figsize=(20, 5))
    for statename in states:
        df_state=data[data.state==statename]
        df_state['new_case']=df_state.cases.diff() #get new cases
        df_state.date=pd.to_datetime(df_state.date) #transform pandas date object
        # print(df_state)
        plt.plot(df_state.date[1:],df_state.new_case[1:],label=statename)
        plt.legend()

    plt.savefig('state_newcases.jpg')
    plt.show()

plot_state_newcase(['Washington','New York','Connecticut'])
'''find the date of highest new cases of a state'''
def find_highest_date(statename):
    df_new=data.loc[data.state==statename]
    df_new['new_cases']=df_new.cases.diff() #get new cases
    maxnum=df_new.new_cases.max()
    row=df_new.loc[df_new.new_cases==maxnum]
    row=row.reset_index()
```

```
    # print(row)
    return row.date.values

High_date=find_highest_date('Washington')
print('the highest new cases corresponding date For the chosen state is',High_date)
def compare_states(state1,state2):
    df_1=data[data.state==state1].reset_index()
    df_2=data[data.state==state2].reset_index()
    df_1['new_case']=df_1.cases.diff()
    df_2['new_case']=df_2.cases.diff()
    max1=df_1.new_case.max()
    max2=df_2.new_case.max()
    if max1>max2:
        print('This state has larger the highest new cases than another one:',state1)
    elif max1==max2:
        print('The two states have the same highest new cases')
    else:
        print('This state has larger the highest new cases than another one:',state2)
    date1=df_1.loc[df_1.new_case==max1].date.values
    date2=df_2.loc[df_2.new_case==max2].date.values
    date1=parse_date(date1[0])
    date2=parse_date(date2[0])
    print('Days between 2 peaks (second state peak minus first state peak):',(date2-
date1).days)

compare_states('Connecticut','Washington')
```

Code for Exercise4:

```
import xml.etree.ElementTree as ET
```

```
import pandas as pd
tree = ET.parse('desc2022.xml')
# print(tree)
root = tree.getroot()
# test1=root[953]
# test1_string=ET.tostring(test1).decode('utf-8')
# print(test1_string)
'''Find DescriptorName by DescriptorUI'''
def find_by_UI(UI):
    MeshData=[]
    for desc_record in root:
        desc_dict={
            'desc_UI':desc_record.find('DescriptorUI').text,
            'desc_name':desc_record.find('DescriptorName/String').text,
        }
        MeshData.append(desc_dict)
    MeSH_df=pd.DataFrame(MeshData)
    # print(MeSH_df)
    result=MeSH_df.loc[MeSH_df.desc_UI==UI]
    # print(result)
    return result.desc_name.values

find1=find_by_UI('D007154')
print('The DescriptorName found by UI:',find1)
'''Find DescriptorUI by DescriptorName'''
def find_by_name(name):
    MeshData=[]
    for desc_record in root:
```

```
        desc_dict={
            'desc_UI':desc_record.find('DescriptorUI').text,
            'desc_name':desc_record.find('DescriptorName/String').text
        }
        MeshData.append(desc_dict)
    MeSH_df=pd.DataFrame(MeshData)
    result=MeSH_df.loc[MeSH_df.desc_name==name]
    # print(result)
    return result.desc_UI.values

find2=find_by_name('Nervous System Diseases')
print('The DescriptorUI found by DescriptorName:',find2)
'''Find the intersection DescriptorNames of descendants by 2 DescriptorUI'''
def find_intersection(UI1,UI2):
    Tree_df=[]
    for desc_record in root:
        if desc_record.find('TreeNumberList') is not None:
            desc_dict={
                'desc_UI':desc_record.find('DescriptorUI').text,
                'desc_name':desc_record.find('DescriptorName/String').text,
                'Tree_num':desc_record.findall('TreeNumberList/TreeNumber')
            }
        else:
            desc_dict={
                'desc_UI':desc_record.find('DescriptorUI').text,
                'desc_name':desc_record.find('DescriptorName/String').text,
                'Tree_num':None
            }
```

```
    Tree_df.append(desc_dict)
Tree_df=pd.DataFrame(Tree_df)
UI1_tree=Tree_df[Tree_df['desc_UI']==UI1]['Tree_num'].values[0][0].text
UI2_tree=Tree_df[Tree_df['desc_UI']==UI2]['Tree_num'].values[0][0].text
# print(UI1_tree.text)
# print(UI1_tree[0])
# print(type(UI1_tree))
# print(UI2_tree)
# print(UI1_tree.values[0])
Search1=[]
for index, row in Tree_df.iterrows():
    if row['Tree_num']!=None:
        for i in range(len(row['Tree_num'])):
            if UI1_tree in row['Tree_num'][i].text:
                Search1.append(row)
Search2=[]
for index, row in Tree_df.iterrows():
    if row['Tree_num']!=None:
        for i in range(len(row['Tree_num'])):
            if UI2_tree in row['Tree_num'][i].text:
                Search2.append(row)
name1=[]
for i in range(len(Search1)):
    name1.append(Search1[i]['desc_name'])
name1=set(name1)
name2=[]
for i in range(len(Search2)):
    name2.append(Search2[i]['desc_name'])
```

```
    name2=set(name2)
    print('Intersection DescriptorNames:',name1&name2)
    print('Intersection set size is:',len(name1&name2))

find_intersection('D007154','D009422')
```