Tahir Manuel D Mello - BIS634 Assignment 1

Exercise 1

Body Temperature Checker

In [1]:

```python
#Building temperature check function

def temp_tester(normal_temp):

    def compare(reported_temp):
        if abs(reported_temp - normal_temp) > 1:
            return False
        else:
            return True

    return compare
```

In [2]:

```python
#Assigning human and chicken normal temperatures

human_tester = temp_tester(37)
chicken_tester = temp_tester(41.1)
```

Code testing

In [3]:

```python
chicken_tester(42) # True -- i.e. not a fever for a chicken
```

```
Out[3]: True
```

In [4]:

```python
human_tester(42) # False -- this would be a severe fever for a human
```

```
Out[4]: False
```

In [5]:

```python
chicken_tester(43) # False
```

```
Out[5]: False
```

In [6]:

```python
human_tester(35) # False -- too low
```

```
Out[6]: False
```

In [7]:

```python
human_tester(98.6) # False -- normal in degrees F but our reference temp was in degree
```

```
Out[7]: False
```

In [8]:

```python
chicken_tester(42.1)
```

```
Out[8]: True
```

Exercise 2

Population Analysis

In [1]:

```
import pandas as pd
import sqlite3

with sqlite3.connect("hw1-population.db") as db:
    data = pd.read_sql_query("SELECT * FROM population", db)
```

Examining data

In [2]:

```
data.head()
```

Out[2]:

	name	age	weight	eyecolor
0	Edna Phelps	88.895690	67.122450	brown
1	Cara Yasso	9.274597	29.251244	brown
2	Gail Rave	18.345613	55.347903	brown
3	Richard Adams	16.367545	70.352184	brown
4	Krista Slater	49.971604	70.563859	brown

What columns does it have?

In [3]:

```
column_names = list(data.columns.values)
print(column_names)
```

```
['name', 'age', 'weight', 'eyecolor']
```

How many rows does it have?

In [4]:

```
data.shape[0]
```

Out[4]:

```
152361
```

Examine the distribution of the ages in the dataset.
Mean, standard deviation, minimum, maximum plus other percentiles displayed.

In [5]:

```
age = data.iloc[:,1]

age.describe()
```

Out[5]:

```
count    152361.000000
mean         39.510528
std          24.152760
min           0.000748
25%          19.296458
50%          38.468955
75%          57.623245
max          99.991547
Name: age, dtype: float64
```

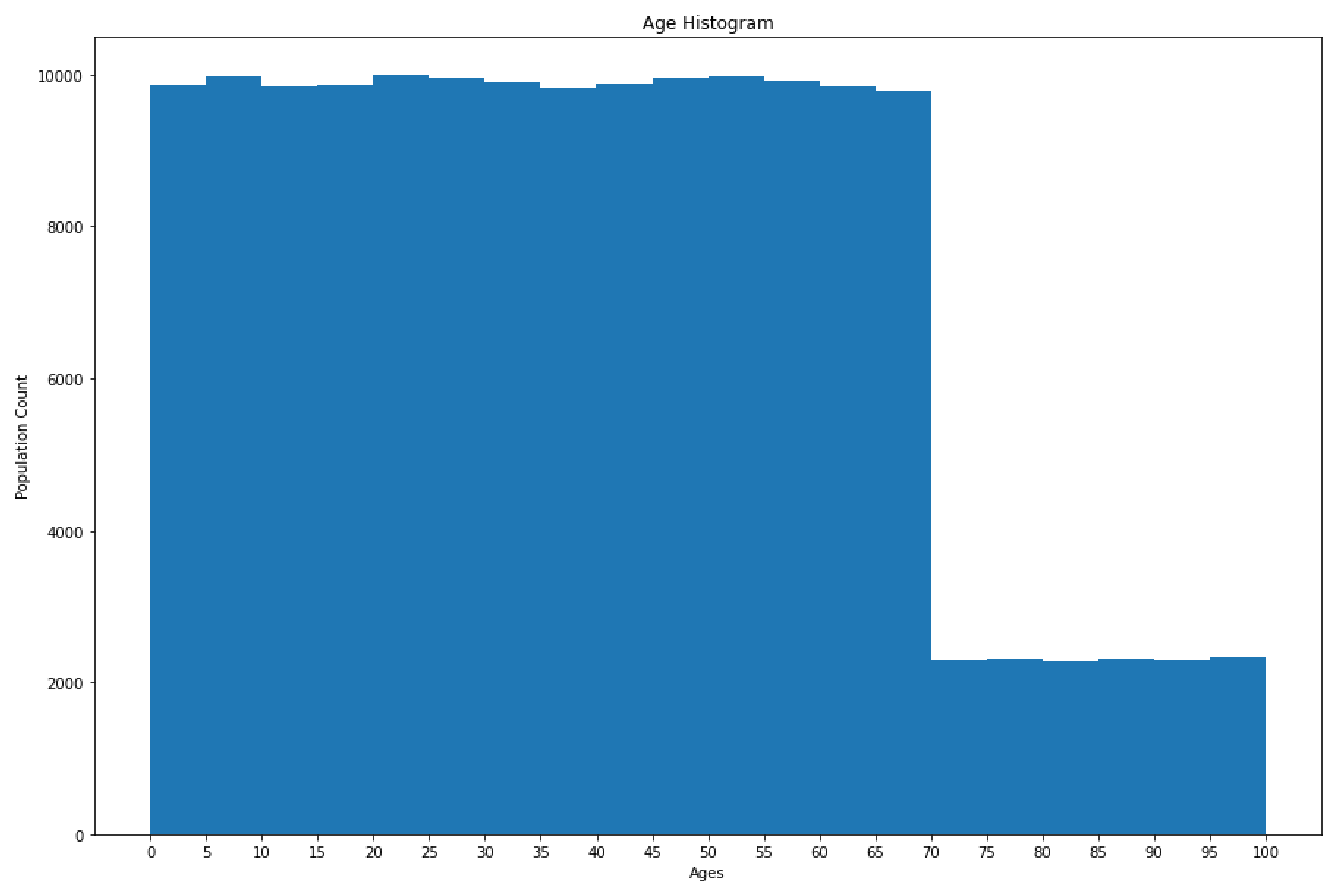
Histogram of the age distribution.

Number of bins chosen is 20.
The range of ages in the dataset is ~0 to ~100 years. Thus, the width of each bin is about 5 years.
The relatively high bin count also gives us better resolution in the histogram.

Bin count is inversely proportional to bin width.

In [6]:

```
import matplotlib.pyplot as plt
import numpy as np

plt.figure(figsize=(15,10))
age_hist = plt.hist(age, bins = 20)

plt.title('Age Histogram')
plt.xlabel('Ages')
plt.ylabel('Population Count')
plt.xticks(np.arange(0, 105, 5))
plt.show()
```



Comment on any outliers or patterns you notice in the distribution of ages.
A majority of the participants were younger than 70 years.

Examine the distribution of the weights in the dataset.
Mean, standard deviation, minimum, maximum plus other percentiles displayed.

In [7]:

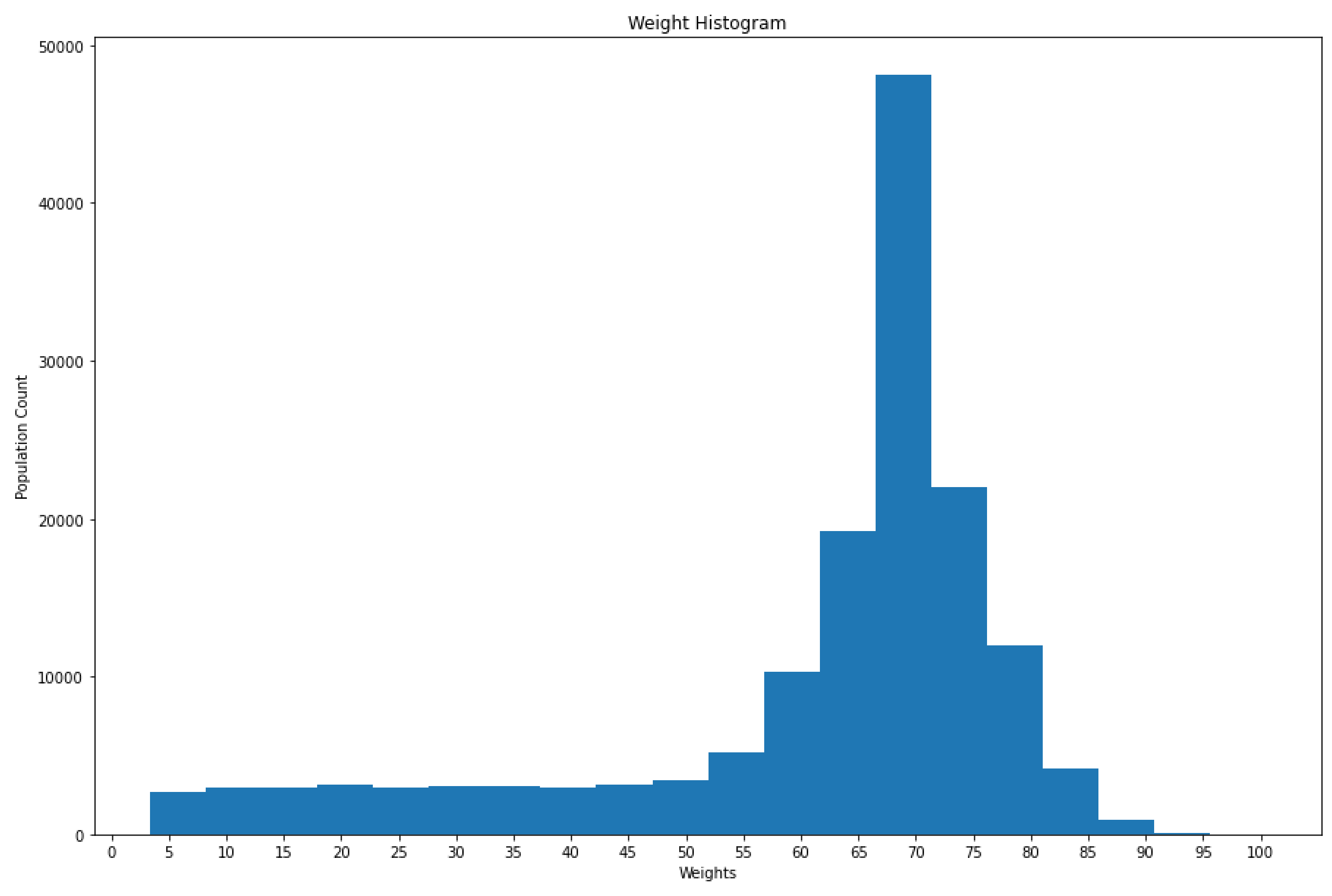
```
weight = data.iloc[:,2]

weight.describe()
```

Out[7]:

```
count    152361.000000
mean         60.884134
std          18.411824
min           3.382084
25%          58.300135
50%          68.000000
75%          71.529860
max         100.435793
Name: weight, dtype: float64
```

In [8]:

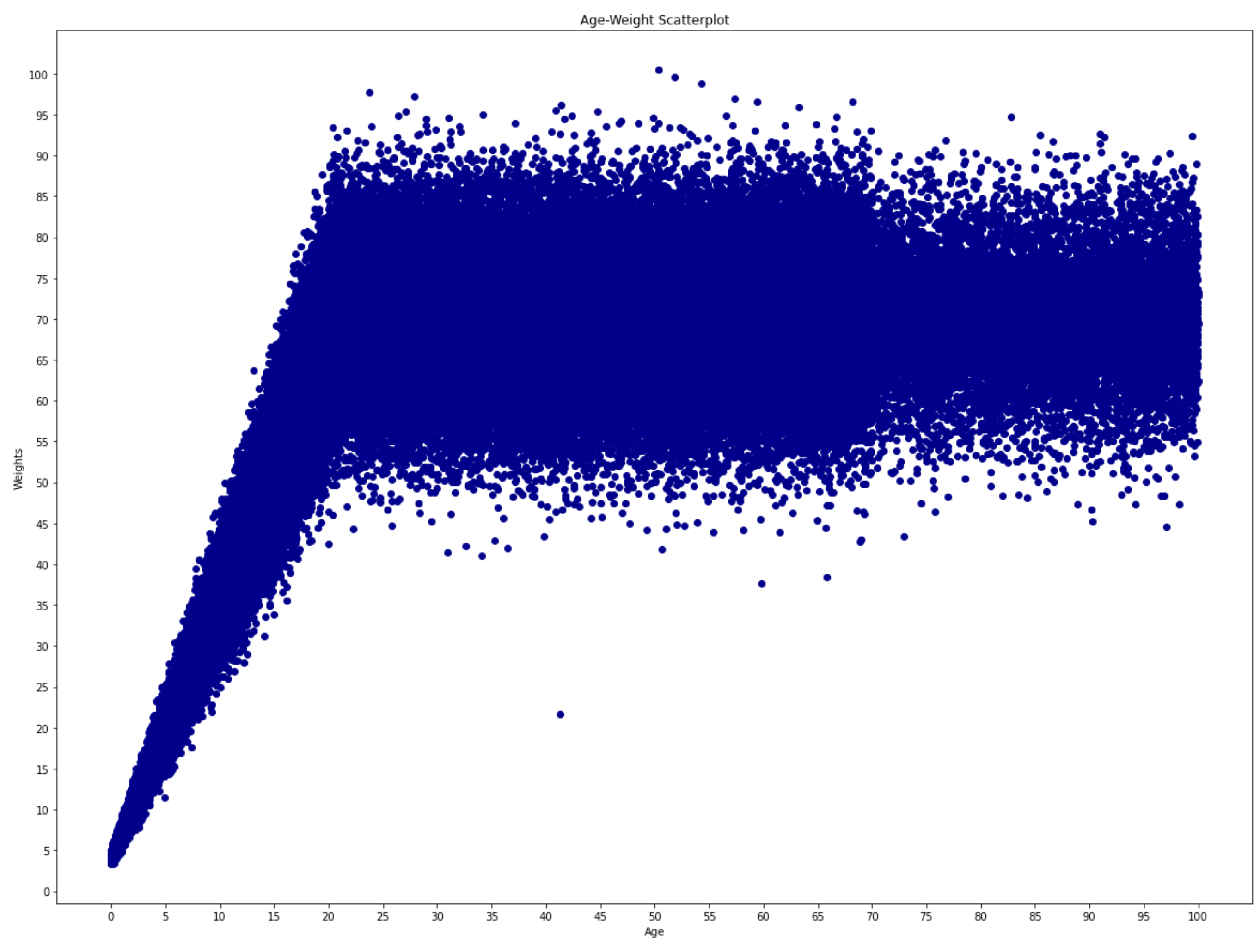
```
import matplotlib.pyplot as plt

plt.figure(figsize=(15,10))
weight_hist = plt.hist(weight, bins = 20)

plt.title('Weight Histogram')
plt.xlabel('Weights')
plt.ylabel('Population Count')
plt.xticks(np.arange(0, 105, 5))
plt.show()
```



Make a scatterplot of the weights vs the ages.

In [9]:

```
import matplotlib.pyplot as plt

plt.figure(figsize=(20,15))
plt.scatter(age, weight, c='DarkBlue')
plt.title('Age-Weight Scatterplot')
plt.xlabel('Age')
plt.ylabel('Weights')
plt.xticks(np.arange(0, 105, 5))
plt.yticks(np.arange(0, 105, 5))
plt.show()
```



The relationship between weight and age is approximately increasing linearly uptil the age of about 20.
This makes sense as growing children will have increasing weight with increasing age.

After the age of approximately 20, the weight remains non-increasingly linear with age around a mean weight of 70.

You should notice at least one outlier that does not follow the general relationship. What is the name of the person?

In [10]:

```
data[(data['age'] > 25) & (data['weight'] < 35)].name.squeeze()
```

Out[10]:

```
'Anthony Freeman'
```

Be sure to explain your process for identifying the person whose values don't follow the usual relationship in the readme.

Visually, the outlier is apparent in the scatter plot between age and weight.
The outlier was picked up by filtering for age values that are more than 25 and weight values that are less than 35.

Exercise 3

COVID-19 Case Analysis

Dataset retrieved on 17th September, 2022 at 7:19 PM.

Reference:
The New York Times. (2021). Coronavirus (Covid-19) Data in the United States.
Retrieved 17th September 2022, from https://github.com/nytimes/covid-19-data.

```
In [1]:
#Downloaded on 17-09-2022 at 7:19 PM
import pandas as pd

data = pd.read_csv("us-states.csv")
```

```
In [2]:
data.head()
```

Out[2]:

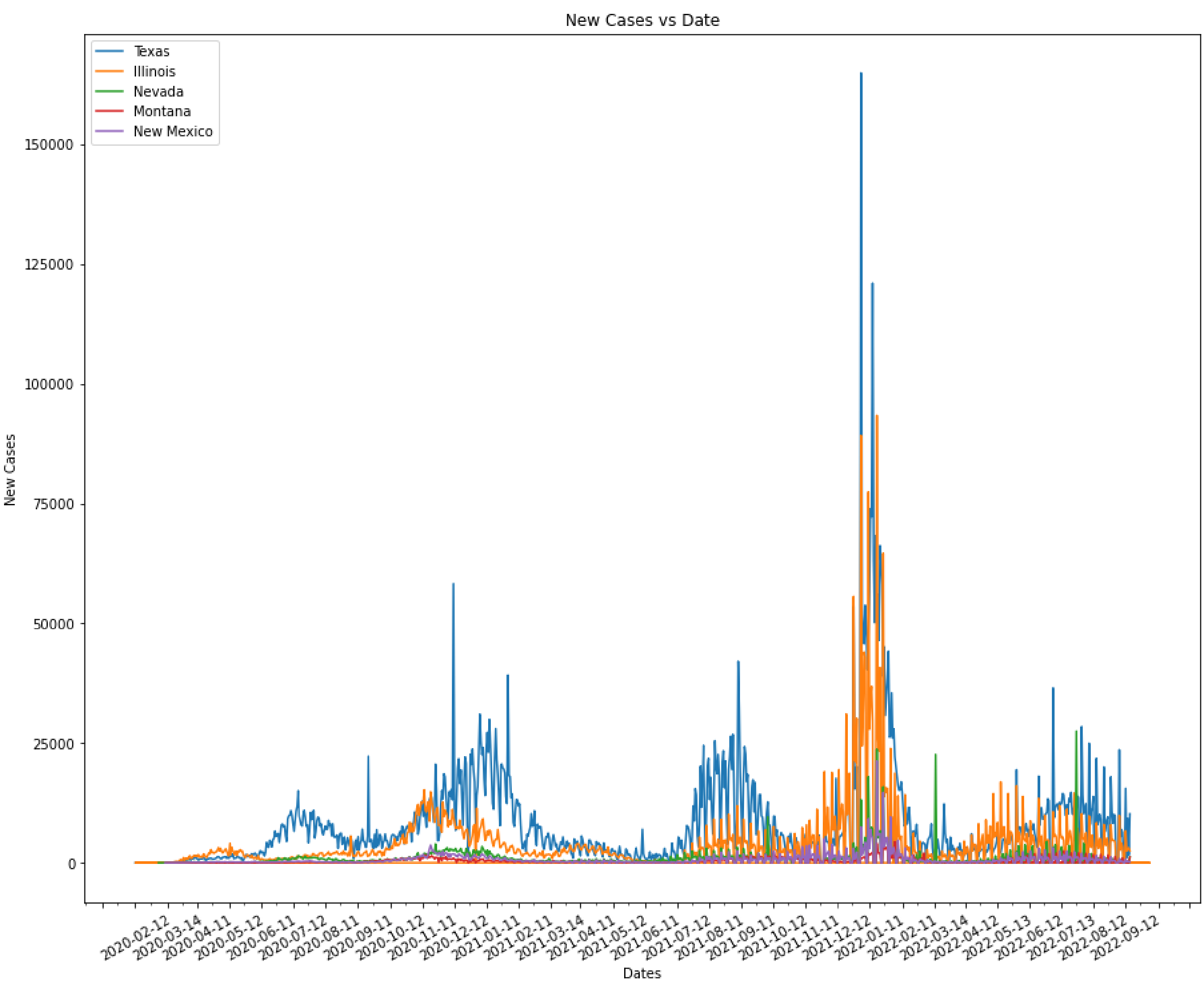
	date	state	fips	cases	deaths
0	2020-01-21	Washington	53	1	0
1	2020-01-22	Washington	53	1	0
2	2020-01-23	Washington	53	1	0
3	2020-01-24	Illinois	17	1	0
4	2020-01-24	Washington	53	1	0

Make a function that takes a list of state names and plots their new cases vs date

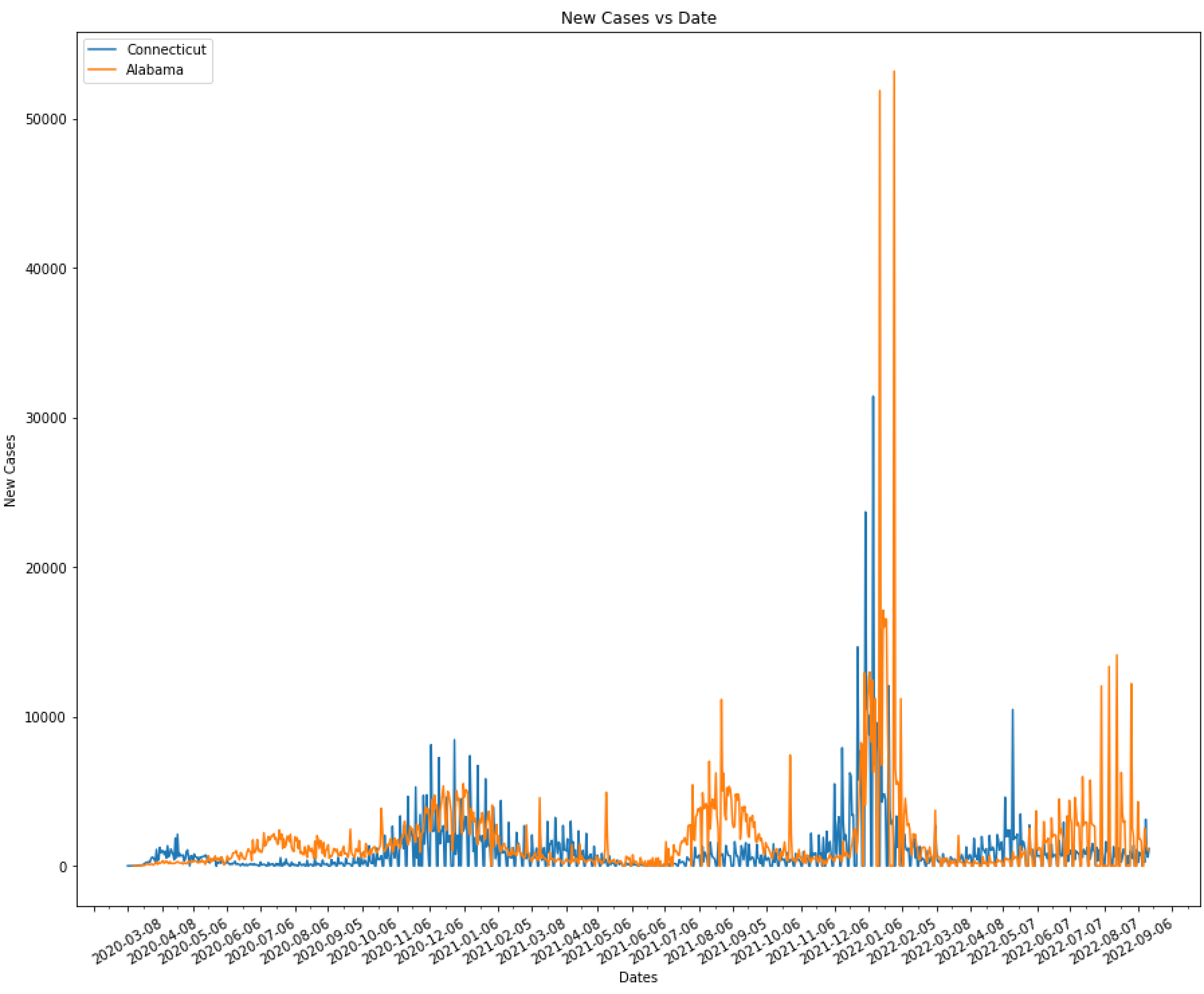
```
In [3]:
import matplotlib.pyplot as plt
from matplotlib.dates import MonthLocator, DateFormatter
import numpy as np

def newcases_plotter(states):

    #totals = []*len(states)
    totals = list()
    plt.figure(figsize=(15, 12))

    for x in states:
        state_data = data[data['state'] == x]

        #Converting cumulative cases to daily cases
        daily_state_data = state_data['cases'].diff().fillna(state_data['cases'])

        state_data.insert(1, "cases_daily", daily_state_data, True)


        plt.plot(state_data['date'], state_data['cases_daily'], label = x)

    plt.xticks(rotation=30)
    plt.gca().xaxis.set_major_locator(MonthLocator())
    plt.gca().xaxis.set_minor_locator(MonthLocator(bymonthday=15))
    plt.title('New Cases vs Date')
    plt.xlabel('Dates')
    plt.ylabel('New Cases')
    plt.legend(loc="upper left")
    plt.show()

    return
```

Test the above function. (X axis labelling changed to month-wise labels for better reading)

```
In [4]:
states = ['Texas', 'Illinois', 'Nevada', 'Montana', 'New Mexico']
newcases_plotter(states)
```



```
In [5]:
states2 = ['Connecticut', 'Alabama']
newcases_plotter(states2)
```



Make a function that takes the name of a state and returns the date of its highest number of new cases.

```
In [6]:
def max_date_finder(state):
    state_data = data[data['state'] == state]

    daily_state_data = state_data['cases'].diff().fillna(state_data['cases'])

    max_cases_index = daily_state_data.idxmax()

    max_date = state_data['date'].loc[max_cases_index]

    return max_date
```

```
In [7]:
max_date_finder('Washington')
```

```
Out[7]:
'2022-01-18'
```

```
In [8]:
max_date_finder('California')
```

```
Out[8]:
'2022-01-10'
```

Make a function that takes the names of two states and reports which one had its highest number of daily new cases first and how many days separate their maximums.

```
In [9]:
from dateutil.parser import parse as parse_date

def highest_daily_compare(states):

    dates = list()

    for x in states:
        dates.append(max_date_finder(x))

    date_difference = (parse_date(dates[0]) - parse_date(dates[1])).days

    if date_difference > 0:
        print("%s had its highest number of daily new cases first by %d day(s)." % (st
    elif date_difference < 0:
        print("%s had highest number of daily new cases first by %d day(s)." % (states
    elif date_difference == 0:
        print("Both states had their highest number of daily new cases on the same day

    return
```

Testing the above function

```
In [10]:
highest_daily_compare(['Washington', 'California'])
```

```
Washington had its highest number of daily new cases first by 8 day(s).
```

```
In [11]:
highest_daily_compare(['Kentucky', 'Minnesota'])
```

```
Minnesota had highest number of daily new cases first by 1 day(s).
```

Exercise 4

XML MeSH Data Analysis

Importing data

In [1]:

```
import xml.etree.ElementTree as ET
from pprint import pprint as pp
import pandas as pd

tree = ET.parse("desc2022.xml")
root = tree.getroot()
```

In [2]:

```
#Function to make pandas dataframe of DescriptorName and DescriptorUI
def descriptor_record():

    root = tree.getroot()
    descriptor_data = []

    for descriptor_record in root:

        descriptor_data_dict = {
            'DescriptorUI': descriptor_record.find('DescriptorUI').text,
            'DescriptorName': descriptor_record.find('DescriptorName/String').text,
        }

        descriptor_data.append(descriptor_data_dict)

    descriptor_data = pd.DataFrame(descriptor_data)

    return descriptor_data
```

Function to extract DescriptorName associated with DescriptorUI

In [3]:

```
def ui_to_name(string):

    root = tree.getroot()
    descriptor_data = []

    descriptor_data = descriptor_record()

    output = descriptor_data[descriptor_data['DescriptorUI'] == string]['DescriptorNa

    return output
```

In [4]:

```
ui_to_name('D007154')
```

Out[4]:

```
'Immune System Diseases'
```

In [5]:

```
ui_to_name('D006090')
```

Out[5]:

```
'Gram-Negative Bacteria'
```

Function to extract DescriptorUI associated with DescriptorName

In [6]:

```
def name_to_ui(string):

    root = tree.getroot()
    descriptor_data = []

    descriptor_data = descriptor_record()

    output = descriptor_data[descriptor_data['DescriptorName'] == string]['Descriptor

    return output
```

In [7]:

```
name_to_ui('Nervous System Diseases')
```

Out[7]:

```
'D009422'
```

In [8]:

```
name_to_ui('Calcium Ionophores')
```

Out[8]:

```
'D061207'
```

Function to find common descendants from an input of DescriptorName and DescriptorUI

In [9]:

```
def common_descendants(descriptor_name, descriptor_ui):

    root = tree.getroot()
    descriptor_data = []
    concept_list1 = []
    concept_list2 = []
    term_list1 = []
    term_list2 = []
    qualifier_list1 = []
    qualifier_list2 = []
    related_list1 = []
    related_list2 = []

    descriptor_data = descriptor_record()

    index1 = descriptor_data[descriptor_data['DescriptorName'] == descriptor_name].inc
    index2 = descriptor_data[descriptor_data['DescriptorUI'] == descriptor_ui].index

    index1 = index1[0]
    index2 = index2[0]

    #Common Concepts
    concept_list_temp1 = root[index1].findall('ConceptList/Concept/ConceptName/String
    for concept in concept_list_temp1:
        concept_list1.append(concept.text)

    concept_list_temp2 = root[index2].findall('ConceptList/Concept/ConceptName/String
    for concept in concept_list_temp2:
        concept_list2.append(concept.text)

    common_concepts = list(set(concept_list1).intersection(concept_list2))

    print('\nDescendant Concepts in both are: ')
    print(common_concepts)

    #Common Terms
    term_list_temp1 = root[index1].findall('ConceptList/Concept/TermList/Term/String')
    for term in term_list_temp1:
        term_list1.append(term.text)

    term_list_temp2 = root[index2].findall('ConceptList/Concept/TermList/Term/String')
    for term in term_list_temp2:
        term_list2.append(term.text)

    common_terms = list(set(term_list1).intersection(term_list2))

    print('\nDescendant Terms in both are: ')
    print(common_terms)

    #Common Qualifiers
    qualifier_list_temp1 = root[index1].findall('AllowableQualifiersList/AllowableQua
    for qualifier in qualifier_list_temp1:
        qualifier_list1.append(qualifier.text)

    qualifier_list_temp2 = root[index2].findall('AllowableQualifiersList/AllowableQua
    for qualifier in qualifier_list_temp2:
        qualifier_list2.append(qualifier.text)

    common_qualifiers = list(set(qualifier_list1).intersection(qualifier_list2))

    print('\nDescendant Qualifiers in both are: ')
    print(common_qualifiers)

    #Common Related Descriptor
    related_list_temp1 = root[index1].findall('SeeRelatedList/SeeRelatedDescriptor/De
    for related in related_list_temp1:
        related_list1.append(related.text)

    related_list_temp2 = root[index2].findall('SeeRelatedList/SeeRelatedDescriptor/De
    for related in related_list_temp2:
        related_list2.append(related.text)

    common_related = list(set(related_list1).intersection(related_list2))

    print('\nDescendant Related Descriptors in both are: ')
    print(common_related)

    return
```

In [10]:

```
common_descendants('Nervous System Diseases', 'D007154')
```

```
Descendant Concepts in both are:
[]

Descendant Terms in both are:
[]

Descendant Qualifiers in both are:
['diet therapy', 'parasitology', 'diagnostic imaging', 'diagnosis', 'radiotherapy', 'p
revention & control', 'history', 'complications', 'psychology', 'therapy', 'immunolog
y', 'genetics', 'epidemiology', 'pathology', 'drug therapy', 'embryology', 'urine', 'p
hysiopathology', 'etiology', 'mortality', 'cerebrospinal fluid', 'economics', 'congeni
tal', 'virology', 'surgery', 'blood', 'classification', 'veterinary', 'ethnology', 'en
zymology', 'metabolism', 'nursing', 'chemically induced', 'microbiology', 'rehabilitat
ion']

Descendant Related Descriptors in both are:
[]
```

Explain briefly in terms of biology/medicine what the above search has found.

The search above has demonstrated that the two topics - 'Nervous System Diseases' and 'Immune System Diseases' - have no direct connections in the Medical Subject Heading database. This makes sense since they are different disease conditions.

They have common qualifiers which are mainly broader treatment methods and disciplines that involve both diseases.