Homework 03

Name: Shun Lin

Student ID: 26636176

Note: Please print the output of each question in a new cell below your code

Numpy Introduction

1a) Create two numpy arrays (a and b). a should be all integers between 25-34 (inclusive), and b should be ten evenly spaced numbers between 1-6. Print all the results below and store them seperately:

i) Cube (i.e. raise to the power of 3) all the elements in both arrays (element-wise)
ii) Add both the cubed arrays (e.g., [1,2] + [3,4] = [4,6])
iii) Sum the elements with even indices of the added array.
iv) Take the square root of the added array (element-wise square root)

In [1]:
```
import numpy as np
# your code here
a = np.arange(25, 35)
b = np.linspace(1,6, 10)
print('i)')
a_cubed = a ** 3
print('a to the cube')
print(a_cubed)
b_cubed = b ** 3
print('b to the cube')
print(b_cubed)
print()
print('ii)')
summed = b_cubed + a_cubed
print(summed)
print()
print('iii)')
even_sum = 0
for i in range(0, len(summed), 2):
    even_sum += summed[i]
print(even_sum)
print()
print('iv)')
sqrt_summed = np.sqrt(summed)
print(sqrt_summed)
```

```
i)
a to the cube
[15625 17576 19683 21952 24389 27000 29791 32768 35937 39304]
b to the cube
[  1.           3.76406036   9.40877915  18.96296296  33.45541838
  53.91495199  81.37037037 116.85048011 161.38408779 216.        ]

ii)
[15626.         17579.76406036 19692.40877915 21970.96296296
 24422.45541838 27053.91495199 29872.37037037 32884.85048011
 36098.38408779 39520.        ]

iii)
125711.61865569273

iv)
[125.00399994 132.58870261 140.32964327 148.22605359 156.27685503
 164.48074341 172.83625306 181.34180566 189.99574755 198.79637824]
```

1b) Append b to a, reshape the appended array so that it is a 4x5, 2d array and store the results in a variable called m. Print m.

In [2]:
```
# your code here
appended = np.append(a, b)
m = np.reshape(appended, (4,5))
print(m)
```

```
[[25.         26.         27.         28.         29.        ]
 [30.         31.         32.         33.         34.        ]
 [ 1.          1.55555556  2.11111111  2.66666667  3.22222222]
 [ 3.77777778  4.33333333  4.88888889  5.44444444  6.        ]]
```

1c) Extract the third and the fourth column of the m matrix. Store the resulting 4x2 matrix in a new variable called m2. Print m2.

In [3]:
```
# your code here
m2 = m[:, 3:]
print(m2)
```

```
[[28.         29.        ]
 [33.         34.        ]
 [ 2.66666667  3.22222222]
 [ 5.44444444  6.        ]]
```

1d) Take the dot product of m2 and m store the results in a matrix called m3. Print m3. Note that Dot product of two matrices A.B =A^TB

In [4]:
```
# your code here
m3 = np.dot(m2.T, m)
print(m3)
```

```
[[1713.2345679  1778.74074074 1844.24691358 1909.75308642 1975.25925926]
 [1770.88888889 1839.01234568 1907.13580247 1975.25925926 2043.38271605]]
```

Numpy conditions

2a) Create a numpy array called 'f' where the values are cosine(x) for x from 0 to pi with 50 equally spaced values in f

- Print f
- Use condition on the array and print an array that is True when f >= 1/2 and False when f < 1/2
- Create and print an array sequence that has only those values where f>= 1/2

In [5]:
```
from math import *
# your code here
x = np.linspace(0, pi, 50)
f = np.cos(x)
print("i)")
print(f)
print()
print("ii)")
boolean_array = []
for num in f:
    boolean_array.append(num >= 1/2)
print(boolean_array)
print()
print("iii)")
f_greater_than_half = f[f >= 0.5]
print(f_greater_than_half)
```

```
i)
[ 1.          0.99794539  0.99179001  0.98155916  0.96729486  0.94905575
  0.92691676  0.90096887  0.8713187   0.8380881   0.80141362  0.76144596
  0.71834935  0.67230089  0.6234898   0.57211666  0.51839257  0.46253829
  0.40478334  0.34536505  0.28452759  0.22252093  0.1595999   0.09602303
  0.03205158 -0.03205158 -0.09602303 -0.1595999  -0.22252093 -0.28452759
 -0.34536505 -0.40478334 -0.46253829 -0.51839257 -0.57211666 -0.6234898
 -0.67230089 -0.71834935 -0.76144596 -0.80141362 -0.8380881  -0.8713187
 -0.90096887 -0.92691676 -0.94905575 -0.96729486 -0.98155916 -0.99179001
 -0.99794539 -1.        ]

ii)
[True, True, True, True, True, True, True, True, True, True, True, True,
True, True, True, True, False, False, False, False, False, False, F
alse, False, False, False, False, False, False, False, False, False, Fals
e, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False]

iii)
[1.         0.99794539 0.99179001 0.98155916 0.96729486 0.94905575
 0.92691676 0.90096887 0.8713187  0.8380881  0.80141362 0.76144596
 0.71834935 0.67230089 0.6234898  0.57211666 0.51839257]
```

NumPy and 2 Variable Prediction

Let 'x' be the number of miles a person drives per day and 'y' be the dollars spent on buying car fuel (per day).

We have created 2 numpy arrays each of size 100 that represent x and y.
x (number of miles) ranges from 1 to 10 with a uniform noise of (0,1/2)
y (money spent in dollars) will be from 1 to 20 with a uniform noise (0,1)

In [6]:
```
# seed the random number generator with a fixed value
import numpy as np
np.random.seed(500)

x = np.linspace(1,10,100)+ np.random.uniform(low=0,high=.5,size=100)
y = np.linspace(1,20,100)+ np.random.uniform(low=0,high=1,size=100)
print('x = ',x)
print('y = ',y)
```

```
x =  [ 1.34683976  1.12176759  1.51512398  1.55233174  1.40619168  1.6507
5498
  1.79399331  1.80243817  1.89844195  2.00100023  2.3344038   2.22424872
  2.24914511  2.36268477  2.49808849  2.8212704   2.68452475  2.68229427
  3.09511169  2.95703884  3.09047742  3.2544361   3.41541904  3.40886375
  3.50672677  3.74960644  3.64861355  3.7721462   3.56368566  4.01092701
  4.15630694  4.06088549  4.02517179  4.25169402  4.15897504  4.26835333
  4.32520644  4.48563164  4.78490721  4.84614839  4.96698768  5.18754259
  5.29582013  5.32097781  5.0674106   5.47601124  5.46852704  5.64537452
  5.49642807  5.89755027  5.68548923  5.76276141  5.94613234  6.18135713
  5.96522091  6.0275473   6.54290191  6.4991329   6.74003765  6.81809807
  6.50611821  6.91538752  7.01250925  6.89905417  7.31314433  7.20472297
  7.1043621   7.48199528  7.58957227  7.61744354  7.6991707   7.85436822
  8.03510784  7.80787781  8.22410224  7.99366248  8.40581097  8.28913792
  8.45971515  8.54227144  8.6906456   8.61856507  8.83489887  8.66309658
  8.94837987  9.20890222  8.9614749   8.92608294  9.13231416  9.55889896
  9.61488451  9.54252979  9.42015491  9.90952569 10.00659591 10.02504265
 10.07330937  9.93489915 10.0892334  10.36509991]
y =  [ 1.6635012   2.0214592   2.10816052  2.26016496  1.96287558  2.9554
635
  3.02881887  3.33565296  2.75465779  3.4250107   3.39670148  3.39377767
  3.78503343  4.38293049  4.32963586  4.03925039  4.73691868  4.30098399
  4.8416329   4.78175957  4.99765787  5.31746817  5.76844671  5.93723749
  5.72811642  6.70973615  6.68143367  6.57482731  7.17737603  7.54863252
  7.30221419  7.3202573   7.78023884  7.91133365  8.2765417   8.69203281
  8.78219865  8.45897546  8.89094715  8.81719921  8.87106971  9.66192562
  9.4020625   9.85990783  9.60359778 10.07386266 10.6957995  10.66721916
 11.18256285 10.57431836 11.46744716 10.94398916 11.26445259 12.09754828
 12.11988037 12.121557   12.17613693 12.43750193 13.00912372 12.86407194
 13.24640866 12.76120085 13.11723062 14.07841099 14.19821707 14.27289001
 14.30624942 14.63060835 14.2770918  15.0744923  14.45261619 15.11897313
 15.2378667  15.27203124 15.32491892 16.01095271 15.71250558 16.29488506
 16.70618934 16.56555394 16.42379457 17.18144744 17.13813976 17.69613625
 17.37763019 17.90942839 17.90343733 18.01951169 18.35727914 18.16841269
 18.61813748 18.66062754 18.81217983 19.44995194 19.7213867  19.71966726
 19.78961904 19.64385088 20.69719809 20.07974319]
```

3a) Find Expected value of x and the expected value of y

In [7]:
```
# your code here
def mean(data):
    return np.mean(data)
mean_x = mean(x)
print("Expected value of x = " + str(mean_x))
mean_y = mean(y)
print("Expected value of y = " + str(mean_y))
```

```
Expected value of x = 5.782532541587923
Expected value of y = 11.012981683344968
```

3b) Find variance of distributions of x and y

In [8]:
```
# your code here
def variance(data):
    return np.var(data)

var_x = variance(x)
print("Variance of x = " + str(var_x))
var_y = variance(y)
print("Variance of y = " + str(var_y))
```

```
Variance of x = 7.03332752947585
Variance of y = 30.113903575509635
```

3c) Find co-variance of x and y.

In [9]:
```
# your code here
def covariance(data1, data2):
    return np.cov(data1, data2)[0][1]

cov_xy = covariance(x, y)
print("Covariance of x and y = " + str(cov_xy))
```

```
Covariance of x and y = 14.657743832803437
```

3d) Assuming that number of dollars spent in car fuel is only dependant on the miles driven, by a linear relationship.
Write code that uses a linear predictor to calculate a predicted value of y for each x i.e y_predicted = f(x) = y0+mx. (Do not use Machine learning libraries)

In [10]:
```
# your code here

# this returns a prediction function
def linear_prediction(x, y):
    mean_y = mean(y)
    mean_x = mean(x)
    cov_xy = covariance(x, y)
    var_x = variance(x)

    def prediction(x):
        return mean_y + (cov_xy / var_x) * (x - mean_x)
    return prediction

y_predicted_function = linear_prediction(x, y)
y_predicted = [y_predicted_function(elem) for elem in x]
print("predicted y's")
print(y_predicted)
```

```
predicted y's
[1.7688155059969297, 1.299755833964003, 2.1195267351983116, 2.19706923872
19657, 1.8925073307368123, 2.4021873223788415, 2.700701885313027, 2.71830
13302844508, 2.918377155794003, 3.1321128293907545, 3.8269395755894307,
3.597371860515052, 3.649256956274452, 3.885878270536657, 4.16806519328280
1, 4.8415895827784805, 4.556606022970095, 4.551957630446149, 5.4122861125
23979, 5.124536601427354, 5.402628099723187, 5.744324715261142, 6.0798197
89507803, 6.066158291960435, 6.270108849268548, 6.7762800760815445, 6.565
806742761493, 6.823253871113012, 6.388813536173846, 7.320882904156916, 7.
623860657107462, 7.424998416424305, 7.350569607437052, 7.822651248071887,
7.629421068494954, 7.857369939133491, 7.975854161986549, 8.31018686281506
7, 8.933889453218617, 9.061518600971068, 9.313352660625903, 9.77299815976
0558, 9.998653002972214, 10.051082650343155, 9.522638138048974, 10.374178
682870525, 10.358581303248963, 10.727138732414964, 10.41672819325845, 11.
252683358836833, 10.810739425764545, 10.971777837836639, 11.3539303947543
15, 11.844148522062419, 11.393711747903128, 11.523602523629854, 12.597622
726310888, 12.506406305192352, 13.008461710228552, 13.171142830597185, 1
2.52096397836108, 13.373898055638552, 13.576303721505058, 13.339858684583
007, 14.202839601918168, 13.976885036472414, 13.767728840969179, 14.55473
1931382912, 14.778926800361141, 14.837011668016164, 15.007334422320346, 1
5.3307724517342, 15.70744124820664, 15.233884524110863, 16.1013133596612
2, 15.62106740389384, 16.48000180661897, 16.236850374881776, 16.592340337
599357, 16.7643910411251, 17.07360890700889, 16.923390110248295, 17.37423
8639698127, 17.01619560870849, 17.610737726524846, 18.153677017708517, 1
7.63802830912916, 17.564269996519084, 17.994064337247536, 18.883084617990
19, 18.9997608038146, 18.848970577986016, 18.593936312892442, 19.61380513
5089265, 19.81610345812138, 19.85454723187822, 19.955137057246038, 19.666
68447445487, 19.988323394891008, 20.56324055552777]
```

3e) Predict y for each value in x, put the error into an array called y_error

In [11]:
```
# your code here
y_error = y - y_predicted
print("y error array")
print(y_error)
```

```
y error array
[-1.05314309e-01  7.21703366e-01 -1.13662110e-02  6.30957167e-02
  7.03682516e-02  5.53276173e-01  3.28116980e-01  6.17351628e-01
 -1.63719369e-01  2.92897867e-01 -4.30238098e-01 -2.03594191e-01
  1.35776473e-01  4.97052216e-01  1.61570667e-01 -8.02339188e-01
  1.80312658e-01 -2.50973643e-01 -5.70653214e-01 -3.42777034e-01
 -4.04970232e-01 -4.26856544e-01 -3.11373081e-01 -1.28920801e-01
 -5.41992427e-01 -6.65439261e-02  1.15626927e-01 -2.48426563e-01
  7.88562495e-01  2.27749619e-01 -3.21646464e-01 -1.04741121e-01
  4.29669229e-01  8.86824064e-02  6.47120628e-01  8.34662873e-01
  8.06344483e-01  1.48788597e-01 -4.29423037e-02 -2.44319388e-01
 -4.42282956e-01 -1.11072540e-01 -5.96590507e-01 -1.91174825e-01
  8.09596391e-02 -3.00316025e-01  3.37218195e-01 -5.99195710e-02
  7.65834652e-01 -6.78364995e-01  6.56707732e-01 -2.77886742e-02
 -8.94778085e-02  2.53399759e-01  7.26168620e-01  5.97954474e-01
 -4.21485798e-01 -6.89043727e-02  6.62014456e-04 -3.07070891e-01
  7.25444679e-01 -6.12697206e-01 -4.59073099e-01  7.38552302e-01
 -4.62253673e-03  2.96004972e-01  5.38520575e-01  7.58764193e-02
 -5.01835004e-01  2.37480633e-01 -5.54718230e-01 -2.11799323e-01
 -4.69574546e-01  3.81467141e-02 -7.76394438e-01  3.89885301e-01
 -7.67496225e-01  5.80346879e-02  1.13849001e-01 -1.98837099e-01
 -6.49814339e-01  2.58057329e-01 -2.36098875e-01  6.79940645e-01
 -2.33107534e-01 -2.44248626e-01  2.65409018e-01  4.55241691e-01
  3.63214807e-01 -7.14671927e-01 -3.81623320e-01 -1.88343033e-01
  2.18243517e-01 -1.63853198e-01 -9.47167582e-02 -1.34879969e-01
 -1.65518016e-01 -2.28335961e-02  7.08874698e-01 -4.83497362e-01]
```

3f) Write code that calculates the root mean square error(RMSE), that is root of average of y-error squared

In [12]:
```
# your code here
def RMSE(y_error):
    result = sum(y_error ** 2) / len(y_error)
    return result

mean_sqr_error = RMSE(y_error)
print("The root mean square error is:")
print(mean_sqr_error)
```

```
The root mean square error is:
0.17750941494696743
```

Pandas Introduction

Reading File

In [13]:
```
# Load required modules
import pandas as pd
import numpy as np
```

Read the CSV file called 'data3.csv' into a dataframe called df.

Data description

- File location: https://bcourses.berkeley.edu/files/74463396/download?download_frd=1 (https://bcourses.berkeley.edu/files/74463396/download?download_frd=1)
- Data source: http://www.fao.org/nr/water/aquastat/data/query/index.html?*lang=en (http://www.fao.org/nr/water/aquastat/data/query/index.html?*lang=en)
- Data, units:
- GDP, current USD (CPI adjusted)
- NRI, mm/yr
- Population density, inhab/km^2
- Total area of the country, 1000 ha = 10km^2
- Total Population, unit 1000 inhabitants

In [14]:
```
# your code here
# import code to dataframe
df=pd.read_csv('data3.csv')
```

4a) Display the first 10 rows of the dataframe

In [15]:
```
# your code here
df.head(10)
```

Out[15]:

	Area	Area Id	Variable Name	Variable Id	Year	Value	Symbol	Other
0	Argentina	9.0	Total area of the country	4100.0	1962.0	278040.0	E	NaN
1	Argentina	9.0	Total area of the country	4100.0	1967.0	278040.0	E	NaN
2	Argentina	9.0	Total area of the country	4100.0	1972.0	278040.0	E	NaN
3	Argentina	9.0	Total area of the country	4100.0	1977.0	278040.0	E	NaN
4	Argentina	9.0	Total area of the country	4100.0	1982.0	278040.0	E	NaN
5	Argentina	9.0	Total area of the country	4100.0	1987.0	278040.0	E	NaN
6	Argentina	9.0	Total area of the country	4100.0	1992.0	278040.0	E	NaN
7	Argentina	9.0	Total area of the country	4100.0	1997.0	278040.0	E	NaN
8	Argentina	9.0	Total area of the country	4100.0	2002.0	278040.0	E	NaN
9	Argentina	9.0	Total area of the country	4100.0	2007.0	278040.0	E	NaN

4b) Display the column names.

In [16]:
```
# your code here
print(list(df.columns.values))
```

```
['Area', 'Area Id', 'Variable Name', 'Variable Id', 'Year', 'Value', 'Sym
bol', 'Other']
```

4c) Use iloc to display the first 3 rows and first 4 columns.

In [17]:
```
# your code here
df.iloc[0:3,0:4]
```

Out[17]:

	Area	Area Id	Variable Name	Variable Id
0	Argentina	9.0	Total area of the country	4100.0
1	Argentina	9.0	Total area of the country	4100.0
2	Argentina	9.0	Total area of the country	4100.0

Data Preprocessing

5a) Find all the rows that have 'NaN' in the 'Symbol' column. Display first 5 rows.

Hint : You might have to use a condition (mask)

In [18]:
```
# your code here
symbol = df['Symbol']
mask = symbol.isnull()
has_nan = df[mask]
has_nan.head(5)
```

Out[18]:

	Area	Area Id	Variable Name	Variable Id	Year	Value	Symbol	Other
390	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
391	E - External data	NaN	NaN	NaN	NaN	NaN	NaN	NaN
392	I - AQUASTAT estimate	NaN	NaN	NaN	NaN	NaN	NaN	NaN
393	K - Aggregate data	NaN	NaN	NaN	NaN	NaN	NaN	NaN
394	L - Modelled data	NaN	NaN	NaN	NaN	NaN	NaN	NaN

5b) Now, we will try to get rid of the NaN valued rows and columns. First, drop the column 'Other' which only has 'NaN' values. Then drop all other rows that have any column with a value 'NaN'. Then display the last 5 rows of the dataframe.

In [19]:
```
# your code here
df = df.drop(columns=['Other'])
df.dropna(inplace=True)
df.tail(5)
```

Out[19]:

	Area	Area Id	Variable Name	Variable Id	Year	Value	Symbol
385	United States of America	231.0	National Rainfall Index (NRI)	4472.0	1981.0	949.2	E
386	United States of America	231.0	National Rainfall Index (NRI)	4472.0	1984.0	974.6	E
387	United States of America	231.0	National Rainfall Index (NRI)	4472.0	1992.0	1020.0	E
388	United States of America	231.0	National Rainfall Index (NRI)	4472.0	1996.0	1005.0	E
389	United States of America	231.0	National Rainfall Index (NRI)	4472.0	2002.0	938.7	E

6a) For our analysis we do not want all the columns in our dataframe. Lets drop all the redundant columns/ features.

Drop columns: Area Id, Variable Id, Symbol. Save the new dataframe as df1. Display the first 5 rows of the new dataframe.

In [20]:
```
# your code here
df1 = df.drop(columns=['Area Id', 'Variable Id', 'Symbol'])
df1.head(5)
```

Out[20]:

	Area	Variable Name	Year	Value
0	Argentina	Total area of the country	1962.0	278040.0
1	Argentina	Total area of the country	1967.0	278040.0
2	Argentina	Total area of the country	1972.0	278040.0
3	Argentina	Total area of the country	1977.0	278040.0
4	Argentina	Total area of the country	1982.0	278040.0

6b) Display all the unique values in your new dataframe for columns: Area, Variable Name, Year.

In [21]:
```
# your code here
def display_unique(column_name):
    print("unique values in " + column_name + " column")
    print(df1[column_name].unique())
    print()
display_unique('Area')
display_unique('Variable Name')
display_unique('Year')
```

```
unique values in Area column
['Argentina' 'Australia' 'Germany' 'Iceland' 'Ireland' 'Sweden'
 'United States of America']

unique values in Variable Name column
['Total area of the country' 'Total population' 'Population density'
 'Gross Domestic Product (GDP)' 'National Rainfall Index (NRI)']

unique values in Year column
[1962. 1967. 1972. 1977. 1982. 1987. 1992. 1997. 2002. 2007. 2012. 2014.
 2015. 1963. 1970. 1974. 1978. 1984. 1990. 1964. 1981. 1985. 1996. 2001.
 1969. 1973. 1979. 1993. 1971. 1975. 1986. 1991. 1998. 2000. 1965. 1983.
 1988. 1995.]
```

6c) Convert the year column to pandas datetime. Convert the 'Year' column float values to pandas datetime objects, where each year is represented as the first day of that year. Also display the column and datatype for 'Year' after conversion. For eg: 1962.0 will be represented as 1962-01-01¶

In [22]:
```
df1["Year"] = pd.to_datetime(df1["Year"], format="%Y.0")
df1.head()
```

Out[22]:

	Area	Variable Name	Year	Value
0	Argentina	Total area of the country	1962-01-01	278040.0
1	Argentina	Total area of the country	1967-01-01	278040.0
2	Argentina	Total area of the country	1972-01-01	278040.0
3	Argentina	Total area of the country	1977-01-01	278040.0
4	Argentina	Total area of the country	1982-01-01	278040.0

Extract specific statistics from the preprocessed data:

7a) Create a dataframe 'dftemp' to store rows where Area is 'Iceland'. Display the dataframe.

In [23]:
```
# your code here
dftemp = df1[df1["Area"] == 'Iceland']
dftemp
```

Out[23]:

	Area	Variable Name	Year	Value
166	Iceland	Total area of the country	1962-01-01	1.030000e+04
167	Iceland	Total area of the country	1967-01-01	1.030000e+04
168	Iceland	Total area of the country	1972-01-01	1.030000e+04
169	Iceland	Total area of the country	1977-01-01	1.030000e+04
170	Iceland	Total area of the country	1982-01-01	1.030000e+04
171	Iceland	Total area of the country	1987-01-01	1.030000e+04
172	Iceland	Total area of the country	1992-01-01	1.030000e+04
173	Iceland	Total area of the country	1997-01-01	1.030000e+04
174	Iceland	Total area of the country	2002-01-01	1.030000e+04
175	Iceland	Total area of the country	2007-01-01	1.030000e+04
176	Iceland	Total area of the country	2012-01-01	1.030000e+04
177	Iceland	Total area of the country	2014-01-01	1.030000e+04
178	Iceland	Total population	1962-01-01	1.826000e+02
179	Iceland	Total population	1967-01-01	1.974000e+02
180	Iceland	Total population	1972-01-01	2.099000e+02
181	Iceland	Total population	1977-01-01	2.221000e+02
182	Iceland	Total population	1982-01-01	2.331000e+02
183	Iceland	Total population	1987-01-01	2.469000e+02
184	Iceland	Total population	1992-01-01	2.599000e+02
185	Iceland	Total population	1997-01-01	2.728000e+02
186	Iceland	Total population	2002-01-01	2.869000e+02
187	Iceland	Total population	2007-01-01	3.054000e+02
188	Iceland	Total population	2012-01-01	3.234000e+02
189	Iceland	Total population	2015-01-01	3.294000e+02
190	Iceland	Population density	1962-01-01	1.773000e+00
191	Iceland	Population density	1967-01-01	1.917000e+00
192	Iceland	Population density	1972-01-01	2.038000e+00
193	Iceland	Population density	1977-01-01	2.156000e+00
194	Iceland	Population density	1982-01-01	2.263000e+00
195	Iceland	Population density	1987-01-01	2.397000e+00
196	Iceland	Population density	1992-01-01	2.523000e+00
197	Iceland	Population density	1997-01-01	2.649000e+00

	Area	Variable Name	Year	Value
198	Iceland	Population density	2002-01-01	2.785000e+00
199	Iceland	Population density	2007-01-01	2.965000e+00
200	Iceland	Population density	2012-01-01	3.140000e+00
201	Iceland	Population density	2015-01-01	3.198000e+00
202	Iceland	Gross Domestic Product (GDP)	1962-01-01	2.849165e+08
203	Iceland	Gross Domestic Product (GDP)	1967-01-01	6.212260e+08
204	Iceland	Gross Domestic Product (GDP)	1972-01-01	8.465069e+08
205	Iceland	Gross Domestic Product (GDP)	1977-01-01	2.226539e+09
206	Iceland	Gross Domestic Product (GDP)	1982-01-01	3.232804e+09
207	Iceland	Gross Domestic Product (GDP)	1987-01-01	5.565384e+09
208	Iceland	Gross Domestic Product (GDP)	1992-01-01	7.138788e+09
209	Iceland	Gross Domestic Product (GDP)	1997-01-01	7.596126e+09
210	Iceland	Gross Domestic Product (GDP)	2002-01-01	9.161798e+09
211	Iceland	Gross Domestic Product (GDP)	2007-01-01	2.129384e+10
212	Iceland	Gross Domestic Product (GDP)	2012-01-01	1.419452e+10
213	Iceland	Gross Domestic Product (GDP)	2015-01-01	1.659849e+10
214	Iceland	National Rainfall Index (NRI)	1967-01-01	8.160000e+02
215	Iceland	National Rainfall Index (NRI)	1971-01-01	9.632000e+02
216	Iceland	National Rainfall Index (NRI)	1975-01-01	1.010000e+03
217	Iceland	National Rainfall Index (NRI)	1981-01-01	9.326000e+02
218	Iceland	National Rainfall Index (NRI)	1986-01-01	9.685000e+02
219	Iceland	National Rainfall Index (NRI)	1991-01-01	1.095000e+03
220	Iceland	National Rainfall Index (NRI)	1997-01-01	9.932000e+02
221	Iceland	National Rainfall Index (NRI)	1998-01-01	9.234000e+02

7b) Print the years when the National Rainfall Index (NRI) was greater than 900 and less than 950 in Iceland. Use the dataframe you created in the previous question 'dftemp'.

In [24]:
```
# your code here
NRI_frame = dftemp[dftemp["Variable Name"] == 'National Rainfall Index (NRI
years_datetime = NRI_frame[(NRI_frame["Value"] > 900) & (NRI_frame["Value"]
years = [datetime.year for datetime in years_datetime]
print(years)
```

```
[1981, 1998]
```

In [25]:
```
print("The years that Iceland's NRI is greater than 900 and less than 950 a
```

```
The years that Iceland's NRI is greater than 900 and less than 950 are ye
ars 1981 and years 1998
```

US statistics:

8a) Create a new DataFrame called `df_usa` that only contains values where 'Area' is equal to 'United States of America'. Set the indices to be the 'Year' column (Use .set_index( )). Display the dataframe head.

In [26]:
```
# your code here
df_usa = df1[df1["Area"] == "United States of America"]
df_usa = df_usa.set_index("Year")
df_usa.head()
```

Out[26]:

Year	Area	Variable Name	Value
1962-01-01	United States of America	Total area of the country	962909.0
1967-01-01	United States of America	Total area of the country	962909.0
1972-01-01	United States of America	Total area of the country	962909.0
1977-01-01	United States of America	Total area of the country	962909.0
1982-01-01	United States of America	Total area of the country	962909.0

8b) Pivot the DataFrame so that the unique values in the column 'Variable Name' becomes the columns. The DataFrame values should be the ones in the the 'Value' column. Save it in df_usa. Display the dataframe head.

In [27]:
```
# your code here
df_usa = df_usa.pivot(columns="Variable Name", values="Value")
df_usa.head()
```

Out[27]:

Variable Name	Gross Domestic Product (GDP)	National Rainfall Index (NRI)	Population density	Total area of the country	Total population
Year					
1962-01-01	6.050000e+11	NaN	19.93	962909.0	191861.0
1965-01-01	NaN	928.5	NaN	NaN	NaN
1967-01-01	8.620000e+11	NaN	21.16	962909.0	203713.0
1969-01-01	NaN	952.2	NaN	NaN	NaN
1972-01-01	1.280000e+12	NaN	22.14	962909.0	213220.0

8c) Rename new columns to ['GDP','NRI','PD','Area','Population'] and display the head.

In [28]:
```
# your code here
df_usa.rename(index=str, columns={"Gross Domestic Product (GDP)": "GDP",
                                  "National Rainfall Index (NRI)": "NRI",
                                  "Population density": "PD",
                                  "Total area of the country": "Area",
                                  "Total population": "Population"}, inplace
df_usa.head()
```

Out[28]:

Variable Name	GDP	NRI	PD	Area	Population
Year					
1962-01-01 00:00:00	6.050000e+11	NaN	19.93	962909.0	191861.0
1965-01-01 00:00:00	NaN	928.5	NaN	NaN	NaN
1967-01-01 00:00:00	8.620000e+11	NaN	21.16	962909.0	203713.0
1969-01-01 00:00:00	NaN	952.2	NaN	NaN	NaN
1972-01-01 00:00:00	1.280000e+12	NaN	22.14	962909.0	213220.0

8d) Replace all 'Nan' values in df_usa with 0. Display the head of the dataframe.

In [29]:
```
# your code here
df_usa.fillna(0, inplace=True)
df_usa.head()
```

Out[29]:

Variable Name	GDP	NRI	PD	Area	Population
Year					
1962-01-01 00:00:00	6.050000e+11	0.0	19.93	962909.0	191861.0
1965-01-01 00:00:00	0.000000e+00	928.5	0.00	0.0	0.0
1967-01-01 00:00:00	8.620000e+11	0.0	21.16	962909.0	203713.0
1969-01-01 00:00:00	0.000000e+00	952.2	0.00	0.0	0.0
1972-01-01 00:00:00	1.280000e+12	0.0	22.14	962909.0	213220.0

Note: Use df_usa

9a) Multiply the 'Area' column for all countries by 10 (so instead of 1000 ha, the unit becomes 100 ha = 1km^2). Display the dataframe head.

In [30]:
```
# your code here
df_usa["Area"] = df_usa["Area"] * 10
df_usa.head()
```

Out[30]:

Variable Name	GDP	NRI	PD	Area	Population
Year					
1962-01-01 00:00:00	6.050000e+11	0.0	19.93	9629090.0	191861.0
1965-01-01 00:00:00	0.000000e+00	928.5	0.00	0.0	0.0
1967-01-01 00:00:00	8.620000e+11	0.0	21.16	9629090.0	203713.0
1969-01-01 00:00:00	0.000000e+00	952.2	0.00	0.0	0.0
1972-01-01 00:00:00	1.280000e+12	0.0	22.14	9629090.0	213220.0

9b) Create a new column in df_usa called 'GDP/capita' and populate it with the calculated GDP per capita. Round the results to two decimal points. Display the dataframe head.

GDP per capita = (GDP / Population)

In [31]:
```
# your code here
df_usa["GDP/capita"] = df_usa["GDP"] / df_usa["Population"]
df_usa.head()
```

Out[31]:

Variable Name	GDP	NRI	PD	Area	Population	GDP/capita
Year						
1962-01-01 00:00:00	6.050000e+11	0.0	19.93	9629090.0	191861.0	3.153325e+06
1965-01-01 00:00:00	0.000000e+00	928.5	0.00	0.0	0.0	NaN
1967-01-01 00:00:00	8.620000e+11	0.0	21.16	9629090.0	203713.0	4.231443e+06
1969-01-01 00:00:00	0.000000e+00	952.2	0.00	0.0	0.0	NaN
1972-01-01 00:00:00	1.280000e+12	0.0	22.14	9629090.0	213220.0	6.003189e+06

9c) Find the maximum value of the 'NRI' column in the US (using pandas methods). What year does the max value occur? Display the values.

In [32]:
```
# your code here
max_row = df_usa.loc[df_usa['NRI'].idxmax()]
print(max_row)
print()
max_value = max_row["NRI"]
print("The max value is " + str(max_value))
print("The year that this max value occurs is " + str(1992))
```

```
Variable Name
GDP            6.540000e+12
NRI            1.020000e+03
PD             2.678000e+01
Area           9.629090e+06
Population     2.579080e+05
GDP/capita     2.535788e+07
Name: 1992-01-01 00:00:00, dtype: float64

The max value is 1020.0
The year that this max value occurs is 1992
```

In [ ]:
```

```