Homework 03

Name:

Anish Saha

Student ID:

26071616

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

In [1]:
```
import numpy as np
import pandas as pd
```

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 [2]:
```
a = np.arange(25,35)
b = np.arange(25,35)

# i.
a = a**3
b = b**3
print("part i. \n")
print(a)
print(b)

# ii.
print("\n")
c = a + b
print("part ii. \n")
print(c)

# iii.
print("\n")
s = 0
for i in range(len(c)):
    if i % 2 == 0:
        s = s + c[i]
print("part iii. \n")
print(s)

# iv.
print("\n")
q = np.sqrt(c)
print("part iv. \n")
print(q)
```

```
part i.

[15625 17576 19683 21952 24389 27000 29791 32768 35937 39304]
[15625 17576 19683 21952 24389 27000 29791 32768 35937 39304]


part ii.

[31250 35152 39366 43904 48778 54000 59582 65536 71874 78608]


part iii.

250850


part iv.

[176.7766953  187.48866632 198.40866917 209.53281366 220.85742007
 232.37900077 244.09424409 256.         268.09326735 280.37118254]
```

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 [3]:
```
m = np.hstack((a, b))
m = m.reshape((4,5))
print(m)
```

```
[[15625 17576 19683 21952 24389]
 [27000 29791 32768 35937 39304]
 [15625 17576 19683 21952 24389]
 [27000 29791 32768 35937 39304]]
```

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 [4]:
```
m2 = m[:,3:5]
print(m2)
```

```
[[21952 24389]
 [35937 39304]
 [21952 24389]
 [35937 39304]]
```

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 =$A^T B$

In [5]:
```
m3 = np.dot(m2.T, m)
print(m3)
```

```
[[2626598000 2912855038 3219329664 3546716546 3895710352]
 [2884572250 3199133056 3535924318 3895710352 4279255474]]
```

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 [6]:
```
f = np.array([np.cos(i) for i in np.linspace(0, np.pi, 50)])
print(f)

tf = (f >= 0.5)
print(tf)

arr = f[tf]
print(arr)
```

```
[ 1.          0.99794539  0.99179001  0.98155916  0.96729486  0.9490557
5
  0.92691676  0.90096887  0.8713187   0.8380881   0.80141362  0.7614459
6
  0.71834935  0.67230089  0.6234898   0.57211666  0.51839257  0.4625382
9
  0.40478334  0.34536505  0.28452759  0.22252093  0.1595999   0.0960230
3
  0.03205158 -0.03205158 -0.09602303 -0.1595999  -0.22252093 -0.2845275
9
 -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 -0.9917900
1
 -0.99794539 -1.        ]
[ True  True  True  True  True  True  True  True  True  True  True  Tru
e
  True  True  True  True  True False False False False False False Fals
e
 False False False False False False False False False False False Fals
e
 False False False False False False False False False False False Fals
e
 False False]
[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 [7]:
```
# 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  1.65
075498
  1.79399331  1.80243817  1.89844195  2.00100023  2.3344038   2.2242487
2
  2.24914511  2.36268477  2.49808849  2.8212704   2.68452475  2.6822942
7
  3.09511169  2.95703884  3.09047742  3.2544361   3.41541904  3.4088637
5
  3.50672677  3.74960644  3.64861355  3.7721462   3.56368566  4.0109270
1
  4.15630694  4.06088549  4.02517179  4.25169402  4.15897504  4.2683533
3
  4.32520644  4.48563164  4.78490721  4.84614839  4.96698768  5.1875425
9
  5.29582013  5.32097781  5.0674106   5.47601124  5.46852704  5.6453745
2
  5.49642807  5.89755027  5.68548923  5.76276141  5.94613234  6.1813571
3
  5.96522091  6.0275473   6.54290191  6.4991329   6.74003765  6.8180980
7
  6.50611821  6.91538752  7.01250925  6.89905417  7.31314433  7.2047229
7
  7.1043621   7.48199528  7.58957227  7.61744354  7.6991707   7.8543682
2
  8.03510784  7.80787781  8.22410224  7.99366248  8.40581097  8.2891379
2
  8.45971515  8.54227144  8.6906456   8.61856507  8.83489887  8.6630965
8
  8.94837987  9.20890222  8.9614749   8.92608294  9.13231416  9.5588989
6
  9.61488451  9.54252979  9.42015491  9.90952569 10.00659591 10.0250426
5
 10.07330937  9.93489915 10.0892334  10.36509991]
y =  [ 1.6635012   2.0214592   2.10816052  2.26016496  1.96287558  2.95
54635
  3.02881887  3.33565296  2.75465779  3.4250107   3.39670148  3.3937776
7
  3.78503343  4.38293049  4.32963586  4.03925039  4.73691868  4.3009839
9
  4.8416329   4.78175957  4.99765787  5.31746817  5.76844671  5.9372374
9
  5.72811642  6.70973615  6.68143367  6.57482731  7.17737603  7.5486325
2
  7.30221419  7.3202573   7.78023884  7.91133365  8.2765417   8.6920328
1
  8.78219865  8.45897546  8.89094715  8.81719921  8.87106971  9.6619256
2
  9.4020625   9.85990783  9.60359778 10.07386266 10.6957995  10.6672191
6
 11.18256285 10.57431836 11.46744716 10.94398916 11.26445259 12.0975482
8
 12.11988037 12.121557   12.17613693 12.43750193 13.00912372 12.8640719
4
 13.24640866 12.76120085 13.11723062 14.07841099 14.19821707 14.2728900
1
 14.30624942 14.63060835 14.2770918  15.0744923  14.45261619 15.1189731
3
```

```
 15.2378667  15.27203124 15.32491892 16.01095271 15.71250558 16.2948850
6
 16.70618934 16.56555394 16.42379457 17.18144744 17.13813976 17.6961362
5
 17.37763019 17.90942839 17.90343733 18.01951169 18.35727914 18.1684126
9
 18.61813748 18.66062754 18.81217983 19.44995194 19.7213867  19.7196672
6
 19.78961904 19.64385088 20.69719809 20.07974319]
```

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

In [8]:
```
e_x = np.mean(x)
e_y = np.mean(y)
print(e_x)
print(e_y)
```

```
5.782532541587923
11.012981683344968
```

3b) Find variance of distributions of x and y

In [9]:
```
var_x = np.var(x)
var_y = np.var(y)
print(var_x)
print(var_y)
```

```
7.03332752947585
30.113903575509635
```

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

In [10]:
```
cov_xy = np.mean(x*y) - (e_x*e_y)
print(cov_xy)
```

```
14.511166394475424
```

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 [11]:
```
y_0 = e_y - (cov_xy / var_x) * e_x
m = cov_xy / var_x

print(y_0)
print(m)
```

```
-0.9175435965867358
2.0632007159713277
```

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

In [12]:
```
y_pred = y_0 + m*x
y_error = y - y_pred

print(y_pred)
print("\n")
print(y_error)
```

```
[ 1.86125717  1.39688809  2.20846128  2.28522836  1.98371207  2.4882952
7
  2.78382468  2.80124813  2.9993232   3.21092152  3.8988      3.6715279
6
  3.7228942   3.9571493   4.23651436  4.9033035   4.62116978  4.6165678
7
  5.46829307  5.18342105  5.45873164  5.79701128  6.12915141  6.1156265
3
  6.31753758  6.81864709  6.61027849  6.86515115  6.43505522  7.3578038
9
  7.65775187  7.46087825  7.38719373  7.85455455  7.66325667  7.8889260
6
  8.00622544  8.33721481  8.95468038  9.08103323  9.33034895  9.7853979
9
 10.00879629 10.06070164  9.53754157 10.38056671 10.36512531 10.7299971
6
 10.42269073 11.25028634 10.81276185 10.97218988 11.35052091 11.8358368
5
 11.38990445 11.51849632 12.58177632 12.49147206 12.98850691 13.1495612
2
 12.50588416 13.35028889 13.5506705  13.31658991 14.17094102 13.947246
 13.74018137 14.51931443 14.74126735 14.79877137 14.96739089 15.2875945
4
 15.66049665 15.1916755  16.05043004 15.57498655 16.42533161 16.1846116
9
 16.53654675 16.70687695 17.01300263 16.86428603 17.31062607 16.9561634
7
 17.54476017 18.08227006 17.57177784 17.49875711 17.92425351 18.8043835
9
 18.91989301 18.77061069 18.51812677 19.5277969  19.72807224 19.7661315
8
 19.8657155  19.58014745 19.89856998 20.46773797]


[-0.19775597  0.62457111 -0.10030076 -0.02506341 -0.02083649  0.4671682
3
  0.24499418  0.53440482 -0.24466541  0.21408918 -0.50209852 -0.2777502
9
  0.06213923  0.42578118  0.0931215  -0.86405311  0.1157489  -0.3155838
8
 -0.62666017 -0.40166149 -0.46107377 -0.47954311 -0.3607047  -0.1783890
4
 -0.58942116 -0.10891094  0.07115518 -0.29032384  0.74232081  0.1908286
3
 -0.35553767 -0.14062095  0.39304511  0.0567791   0.61328502  0.8031067
6
  0.77597321  0.12176065 -0.06373323 -0.26383402 -0.45927925 -0.1234723
8
 -0.60673379 -0.20079382  0.0660562  -0.30670405  0.33067419 -0.062778
  0.75987212 -0.67596798  0.65468531 -0.02820071 -0.08606832  0.2617114
3
  0.72997592  0.60306068 -0.40563939 -0.05397013  0.02061681 -0.2854892
8
  0.7405245  -0.58908804 -0.43343988  0.76182107  0.02727604  0.3256440
1
  0.56606805  0.11129392 -0.46417555  0.27572093 -0.5147747  -0.1686214
2
```

```
 -0.42262995  0.08035574 -0.72551112  0.43596616 -0.71282602  0.1102733
7
  0.16964259 -0.14132301 -0.58920807  0.31716141 -0.17248631  0.7399727
8
 -0.16712997 -0.17284167  0.33165948  0.52075457  0.43302563 -0.6359709
 -0.30175553 -0.10998314  0.29405306 -0.07784496 -0.00668554 -0.0464643
1
 -0.07609646  0.06370343  0.79862812 -0.38799477]
```

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

In [13]:
```
rmse = np.mean(np.sqrt(y_error**2))
print(rmse)
```

```
0.34160940705875104
```

Pandas Introduction

Reading File

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]:
```
df = pd.read_csv('data3.csv')
```

4a) Display the first 10 rows of the dataframe

In [15]:
```
df.head(n=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]:
```
df.columns
```

Out[16]:
```
Index(['Area', 'Area Id', 'Variable Name', 'Variable Id', 'Year', 'Valu
e',
       'Symbol', 'Other'],
      dtype='object')
```

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

In [17]:
```
df.iloc[:3,: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]:
```
df.loc[df['Symbol'].isnull()]
```

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
395	(c) FAO of the UN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
396	The information contained in AQUASTAT is provi...	NaN	NaN	NaN	NaN	NaN	NaN	NaN
397	FAO. 2016. AQUASTAT Main Database - Food and A...	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]:
```
df = df.drop(columns='Other')
df = df.dropna()
df.tail(n=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(n=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
print(df["Area"].unique())
print(df["Variable Name"].unique())
print(df["Year"].unique())
```

```
['Argentina' 'Australia' 'Germany' 'Iceland' 'Ireland' 'Sweden'
 'United States of America']
['Total area of the country' 'Total population' 'Population density'
 'Gross Domestic Product (GDP)' 'National Rainfall Index (NRI)']
[1962. 1967. 1972. 1977. 1982. 1987. 1992. 1997. 2002. 2007. 2012. 201
4.
 2015. 1963. 1970. 1974. 1978. 1984. 1990. 1964. 1981. 1985. 1996. 200
1.
 1969. 1973. 1979. 1993. 1971. 1975. 1986. 1991. 1998. 2000. 1965. 198
3.
 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]:
```
df['Year'] = df['Year'].apply(int)
df['Year'] = pd.to_datetime(df['Year'], format='%Y')
print(df['Year'])
```

```
0      1962-01-01
1      1967-01-01
2      1972-01-01
3      1977-01-01
4      1982-01-01
5      1987-01-01
6      1992-01-01
7      1997-01-01
8      2002-01-01
9      2007-01-01
10     2012-01-01
11     2014-01-01
12     1962-01-01
13     1967-01-01
14     1972-01-01
15     1977-01-01
16     1982-01-01
17     1987-01-01
18     1992-01-01
19     1997-01-01
20     2002-01-01
21     2007-01-01
22     2012-01-01
23     2015-01-01
24     1962-01-01
25     1967-01-01
26     1972-01-01
27     1977-01-01
28     1982-01-01
29     1987-01-01
          ...
360    1972-01-01
361    1977-01-01
362    1982-01-01
363    1987-01-01
364    1992-01-01
365    1997-01-01
366    2002-01-01
367    2007-01-01
368    2012-01-01
369    2015-01-01
370    1962-01-01
371    1967-01-01
372    1972-01-01
373    1977-01-01
374    1982-01-01
375    1987-01-01
376    1992-01-01
377    1997-01-01
378    2002-01-01
379    2007-01-01
380    2012-01-01
381    2015-01-01
382    1965-01-01
383    1969-01-01
384    1974-01-01
385    1981-01-01
```

```
386   1984-01-01
387   1992-01-01
388   1996-01-01
389   2002-01-01
Name: Year, Length: 390, dtype: datetime64[ns]
```

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 = df.loc[df['Area'] == 'Iceland']
dftemp
```

Out[23]:

	Area	Area Id	Variable Name	Variable Id	Year	Value	Symbol
166	Iceland	99.0	Total area of the country	4100.0	1962-01-01	1.030000e+04	E
167	Iceland	99.0	Total area of the country	4100.0	1967-01-01	1.030000e+04	E
168	Iceland	99.0	Total area of the country	4100.0	1972-01-01	1.030000e+04	E
169	Iceland	99.0	Total area of the country	4100.0	1977-01-01	1.030000e+04	E
170	Iceland	99.0	Total area of the country	4100.0	1982-01-01	1.030000e+04	E
171	Iceland	99.0	Total area of the country	4100.0	1987-01-01	1.030000e+04	E
172	Iceland	99.0	Total area of the country	4100.0	1992-01-01	1.030000e+04	E
173	Iceland	99.0	Total area of the country	4100.0	1997-01-01	1.030000e+04	E
174	Iceland	99.0	Total area of the country	4100.0	2002-01-01	1.030000e+04	E
175	Iceland	99.0	Total area of the country	4100.0	2007-01-01	1.030000e+04	E
176	Iceland	99.0	Total area of the country	4100.0	2012-01-01	1.030000e+04	E
177	Iceland	99.0	Total area of the country	4100.0	2014-01-01	1.030000e+04	E
178	Iceland	99.0	Total population	4104.0	1962-01-01	1.826000e+02	E
179	Iceland	99.0	Total population	4104.0	1967-01-01	1.974000e+02	E
180	Iceland	99.0	Total population	4104.0	1972-01-01	2.099000e+02	E
181	Iceland	99.0	Total population	4104.0	1977-01-01	2.221000e+02	E
182	Iceland	99.0	Total population	4104.0	1982-01-01	2.331000e+02	E
183	Iceland	99.0	Total population	4104.0	1987-01-01	2.469000e+02	E

	Area	Area Id	Variable Name	Variable Id	Year	Value	Symbol
184	Iceland	99.0	Total population	4104.0	1992-01-01	2.599000e+02	E
185	Iceland	99.0	Total population	4104.0	1997-01-01	2.728000e+02	E
186	Iceland	99.0	Total population	4104.0	2002-01-01	2.869000e+02	E
187	Iceland	99.0	Total population	4104.0	2007-01-01	3.054000e+02	E
188	Iceland	99.0	Total population	4104.0	2012-01-01	3.234000e+02	E
189	Iceland	99.0	Total population	4104.0	2015-01-01	3.294000e+02	E
190	Iceland	99.0	Population density	4107.0	1962-01-01	1.773000e+00	E
191	Iceland	99.0	Population density	4107.0	1967-01-01	1.917000e+00	E
192	Iceland	99.0	Population density	4107.0	1972-01-01	2.038000e+00	E
193	Iceland	99.0	Population density	4107.0	1977-01-01	2.156000e+00	E
194	Iceland	99.0	Population density	4107.0	1982-01-01	2.263000e+00	E
195	Iceland	99.0	Population density	4107.0	1987-01-01	2.397000e+00	E
196	Iceland	99.0	Population density	4107.0	1992-01-01	2.523000e+00	E
197	Iceland	99.0	Population density	4107.0	1997-01-01	2.649000e+00	E
198	Iceland	99.0	Population density	4107.0	2002-01-01	2.785000e+00	E
199	Iceland	99.0	Population density	4107.0	2007-01-01	2.965000e+00	E
200	Iceland	99.0	Population density	4107.0	2012-01-01	3.140000e+00	E
201	Iceland	99.0	Population density	4107.0	2015-01-01	3.198000e+00	E
202	Iceland	99.0	Gross Domestic Product (GDP)	4112.0	1962-01-01	2.849165e+08	E

	Area	Area Id	Variable Name	Variable Id	Year	Value	Symbol
203	Iceland	99.0	Gross Domestic Product (GDP)	4112.0	1967-01-01	6.212260e+08	E
204	Iceland	99.0	Gross Domestic Product (GDP)	4112.0	1972-01-01	8.465069e+08	E
205	Iceland	99.0	Gross Domestic Product (GDP)	4112.0	1977-01-01	2.226539e+09	E
206	Iceland	99.0	Gross Domestic Product (GDP)	4112.0	1982-01-01	3.232804e+09	E
207	Iceland	99.0	Gross Domestic Product (GDP)	4112.0	1987-01-01	5.565384e+09	E
208	Iceland	99.0	Gross Domestic Product (GDP)	4112.0	1992-01-01	7.138788e+09	E
209	Iceland	99.0	Gross Domestic Product (GDP)	4112.0	1997-01-01	7.596126e+09	E
210	Iceland	99.0	Gross Domestic Product (GDP)	4112.0	2002-01-01	9.161798e+09	E
211	Iceland	99.0	Gross Domestic Product (GDP)	4112.0	2007-01-01	2.129384e+10	E
212	Iceland	99.0	Gross Domestic Product (GDP)	4112.0	2012-01-01	1.419452e+10	E
213	Iceland	99.0	Gross Domestic Product (GDP)	4112.0	2015-01-01	1.659849e+10	E
214	Iceland	99.0	National Rainfall Index (NRI)	4472.0	1967-01-01	8.160000e+02	E
215	Iceland	99.0	National Rainfall Index (NRI)	4472.0	1971-01-01	9.632000e+02	E
216	Iceland	99.0	National Rainfall Index (NRI)	4472.0	1975-01-01	1.010000e+03	E
217	Iceland	99.0	National Rainfall Index (NRI)	4472.0	1981-01-01	9.326000e+02	E
218	Iceland	99.0	National Rainfall Index (NRI)	4472.0	1986-01-01	9.685000e+02	E
219	Iceland	99.0	National Rainfall Index (NRI)	4472.0	1991-01-01	1.095000e+03	E
220	Iceland	99.0	National Rainfall Index (NRI)	4472.0	1997-01-01	9.932000e+02	E
221	Iceland	99.0	National Rainfall Index (NRI)	4472.0	1998-01-01	9.234000e+02	E

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]:
```
x = dftemp.loc[ (dftemp['Variable Name'] == 'National Rainfall Index (NR
I)')
                  & (dftemp['Value'] > 900)
                  & (dftemp['Value'] < 950) ]
print(x['Year'])
print("\n")
x
```

```
217   1981-01-01
221   1998-01-01
Name: Year, dtype: datetime64[ns]
```

Out[24]:

	Area	Area Id	Variable Name	Variable Id	Year	Value	Symbol
217	Iceland	99.0	National Rainfall Index (NRI)	4472.0	1981-01-01	932.6	E
221	Iceland	99.0	National Rainfall Index (NRI)	4472.0	1998-01-01	923.4	E

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 [25]:
```
df_usa = df.loc[df['Area'] == 'United States of America'].set_index(keys
 = 'Year')
df_usa.head()
```

Out[25]:

	Area	Area Id	Variable Name	Variable Id	Value	Symbol
Year						
1962-01-01	United States of America	231.0	Total area of the country	4100.0	962909.0	E
1967-01-01	United States of America	231.0	Total area of the country	4100.0	962909.0	E
1972-01-01	United States of America	231.0	Total area of the country	4100.0	962909.0	E
1977-01-01	United States of America	231.0	Total area of the country	4100.0	962909.0	E
1982-01-01	United States of America	231.0	Total area of the country	4100.0	962909.0	E

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 [26]:
```
df_usa = df_usa.pivot(columns='Variable Name', values='Value')
df_usa.head()
```

Out[26]:

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 [27]:
```
df_usa.columns = ['GDP','NRI','PD','Area','Population']
df_usa.head()
```

Out[27]:

	GDP	NRI	PD	Area	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

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

In [28]:
```
df_usa = df_usa.fillna(0)
df_usa.head()
```

Out[28]:

	GDP	NRI	PD	Area	Population
Year					
1962-01-01	6.050000e+11	0.0	19.93	962909.0	191861.0
1965-01-01	0.000000e+00	928.5	0.00	0.0	0.0
1967-01-01	8.620000e+11	0.0	21.16	962909.0	203713.0
1969-01-01	0.000000e+00	952.2	0.00	0.0	0.0
1972-01-01	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 [29]:
```
df_usa['Area'] = df_usa['Area'] * 10
df_usa.head()
```

Out[29]:

	GDP	NRI	PD	Area	Population
Year					
1962-01-01	6.050000e+11	0.0	19.93	9629090.0	191861.0
1965-01-01	0.000000e+00	928.5	0.00	0.0	0.0
1967-01-01	8.620000e+11	0.0	21.16	9629090.0	203713.0
1969-01-01	0.000000e+00	952.2	0.00	0.0	0.0
1972-01-01	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 [30]:
```
df_usa['GDP/capita'] = df_usa['GDP'] / df_usa['Population']
df_usa.head()
```

Out[30]:

	GDP	NRI	PD	Area	Population	GDP/capita
Year						
1962-01-01	6.050000e+11	0.0	19.93	9629090.0	191861.0	3.153325e+06
1965-01-01	0.000000e+00	928.5	0.00	0.0	0.0	NaN
1967-01-01	8.620000e+11	0.0	21.16	9629090.0	203713.0	4.231443e+06
1969-01-01	0.000000e+00	952.2	0.00	0.0	0.0	NaN
1972-01-01	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 [31]:
```
df_usa.loc[df_usa['NRI'] == df_usa['NRI'].max()] # 1992
```

Out[31]:

	GDP	NRI	PD	Area	Population	GDP/capita
Year						
1992-01-01	6.540000e+12	1020.0	26.78	9629090.0	257908.0	2.535788e+07