HW3 - Numpy & Pandas (updated)

February 7, 2019

1 Homework 03

1.1 Name: Shrey Samdani

1.2 Student ID: 3032000414

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

1.4 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]: # your code here
        import numpy as np
        a = np.arange(25,35)
        b = np.linspace(1,6,10)
        print("a",a)
        print("b",b)

        a_cubed = a**3
        print("a_cubed",a_cubed)

        b_cubed = b**3
        print("b_cubed",b_cubed)

        summed = a_cubed + b_cubed
        print("summed",summed)

        sumEven = sum(summed[::2])
```

```
        print("element sum with even indices", sumEven)

        sqrtSummed = summed**(1/2)
        print("sqrtSummed",sqrtSummed)
```

```
a [25 26 27 28 29 30 31 32 33 34]
b [1.         1.55555556 2.11111111 2.66666667 3.22222222 3.77777778
 4.33333333 4.88888889 5.44444444 6.        ]
a_cubed [15625 17576 19683 21952 24389 27000 29791 32768 35937 39304]
b_cubed [  1.           3.76406036   9.40877915  18.96296296  33.45541838
  53.91495199  81.37037037 116.85048011 161.38408779 216.        ]
summed [15626.         17579.76406036 19692.40877915 21970.96296296
 24422.45541838 27053.91495199 29872.37037037 32884.85048011
 36098.38408779 39520.        ]
element sum with even indices 125711.61865569273
sqrtSummed [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]: appended = np.concatenate((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]: m2 = m[:,2:4]
        m2
```

```
Out[3]: array([[27.        , 28.        ],
               [32.        , 33.        ],
               [ 2.11111111,  2.66666667],
               [ 4.88888889,  5.44444444]])
```

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 =ATB

```
In [4]: m3 = m2.T @ m
        m3
```

```
Out[4]: array([[1655.58024691, 1718.4691358 , 1781.35802469, 1844.24691358,
                1907.13580247],
               [1713.2345679 , 1778.74074074, 1844.24691358, 1909.75308642,
                1975.25925926]])
```

1.5 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]: # your code here
        f = np.linspace(0,np.pi,50)
        f = np.cos(f)

        print(f)
        print(f >= 1/2)

        f2 = f[f>=1/2]
        print(f2)
```

```
[ 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.        ]
[ True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 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]
```

1.6 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  1.65075498
  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  2.9554635
  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]: eX = np.mean(x)
        print(eX)

        eY = np.mean(y)
        print(eY)
```

```
5.782532541587923
```

```
11.012981683344968
```

3b) Find variance of distributions of x and y

```
In [8]: varX = np.var(x)
        print(varX)

        varY = np.var(y)
        print(varY)
```

```
7.03332752947585
30.113903575509635
```

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

```
In [9]: covar = np.cov(x,y, bias = True)[0][1]
        print(covar)
```

```
14.511166394475401
```

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]: def predict(x):
             return eY+covar/varX*(x-eX)
```

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

```
In [11]: y_pred = predict(x)
         y_error = y - y_pred
         print(y_error)
```

```
[-0.19775597  0.62457111 -0.10030076 -0.02506341 -0.02083649  0.46716823
  0.24499418  0.53440482 -0.24466541  0.21408918 -0.50209852 -0.27775029
  0.06213923  0.42578118  0.0931215  -0.86405311  0.1157489  -0.31558388
 -0.62666017 -0.40166149 -0.46107377 -0.47954311 -0.3607047  -0.17838904
 -0.58942116 -0.10891094  0.07115518 -0.29032384  0.74232081  0.19082863
 -0.35553767 -0.14062095  0.39304511  0.0567791   0.61328502  0.80310676
  0.77597321  0.12176065 -0.06373323 -0.26383402 -0.45927925 -0.12347238
 -0.60673379 -0.20079382  0.0660562  -0.30670405  0.33067419 -0.062778
  0.75987212 -0.67596798  0.65468531 -0.02820071 -0.08606832  0.26171143
  0.72997592  0.60306068 -0.40563939 -0.05397013  0.02061681 -0.28548928
  0.7405245  -0.58908804 -0.43343988  0.76182107  0.02727604  0.32564401
  0.56606805  0.11129392 -0.46417555  0.27572093 -0.5147747  -0.16862142
 -0.42262995  0.08035574 -0.72551112  0.43596616 -0.71282602  0.11027337
```

```
 0.16964259 -0.14132301 -0.58920807  0.31716141 -0.17248631  0.73997278
-0.16712997 -0.17284167  0.33165948  0.52075457  0.43302563 -0.6359709
-0.30175553 -0.10998314  0.29405306 -0.07784496 -0.00668554 -0.04646431
-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 [12]: rmse = np.sqrt(np.mean(np.power(y_error,2)))
         print(rmse)
```

```
0.4176777236685613
```

1.7 Pandas Introduction

1.8 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
- Data source: 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
         df = pd.read_csv("data3.csv")
```

4a) Display the first 10 rows of the dataframe

```
In [15]: df.head(10)
```

```
Out[15]:         Area  Area Id              Variable Name  Variable Id    Year  \
         0  Argentina      9.0  Total area of the country       4100.0  1962.0
         1  Argentina      9.0  Total area of the country       4100.0  1967.0
         2  Argentina      9.0  Total area of the country       4100.0  1972.0
         3  Argentina      9.0  Total area of the country       4100.0  1977.0
         4  Argentina      9.0  Total area of the country       4100.0  1982.0
```

```
5  Argentina      9.0  Total area of the country       4100.0  1987.0
6  Argentina      9.0  Total area of the country       4100.0  1992.0
7  Argentina      9.0  Total area of the country       4100.0  1997.0
8  Argentina      9.0  Total area of the country       4100.0  2002.0
9  Argentina      9.0  Total area of the country       4100.0  2007.0

      Value Symbol  Other
0  278040.0      E    NaN
1  278040.0      E    NaN
2  278040.0      E    NaN
3  278040.0      E    NaN
4  278040.0      E    NaN
5  278040.0      E    NaN
6  278040.0      E    NaN
7  278040.0      E    NaN
8  278040.0      E    NaN
9  278040.0      E    NaN
```

4b) Display the column names.

```
In [16]: list(df)

Out[16]: ['Area',
          'Area Id',
          'Variable Name',
          'Variable Id',
          'Year',
          'Value',
          'Symbol',
          'Other']
```

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

```
In [17]: # your code here
         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
```

1.9 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[df['Symbol'].isnull()].head()
```

```
Out[18]:                       Area  Area Id Variable Name  Variable Id  Year  Value  \
         390                    NaN      NaN           NaN          NaN   NaN    NaN
         391      E - External data      NaN           NaN          NaN   NaN    NaN
         392  I - AQUASTAT estimate      NaN           NaN          NaN   NaN    NaN
         393     K - Aggregate data      NaN           NaN          NaN   NaN    NaN
         394      L - Modelled data      NaN           NaN          NaN   NaN    NaN

             Symbol  Other
         390    NaN    NaN
         391    NaN    NaN
         392    NaN    NaN
         393    NaN    NaN
         394    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(['Other'], axis=1).dropna()
         df.tail()
```

```
Out[19]:                         Area  Area Id                 Variable Name  \
         385  United States of America    231.0  National Rainfall Index (NRI)
         386  United States of America    231.0  National Rainfall Index (NRI)
         387  United States of America    231.0  National Rainfall Index (NRI)
         388  United States of America    231.0  National Rainfall Index (NRI)
         389  United States of America    231.0  National Rainfall Index (NRI)

              Variable Id    Year   Value Symbol
         385       4472.0  1981.0   949.2      E
         386       4472.0  1984.0   974.6      E
         387       4472.0  1992.0  1020.0      E
         388       4472.0  1996.0  1005.0      E
         389       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(['Area Id', 'Variable Id', 'Symbol'],axis=1)
         df1.head()
```

```
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]: for col in ['Area', 'Variable Name', 'Year']:
             print(col)
             print(df[col].unique())
```

```
Area
['Argentina' 'Australia' 'Germany' 'Iceland' 'Ireland' 'Sweden'
 'United States of America']
Variable Name
['Total area of the country' 'Total population' 'Population density'
 'Gross Domestic Product (GDP)' 'National Rainfall Index (NRI)']
Year
[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 1962-01-01ű

```
In [22]: df['Year'] = pd.to_datetime(df['Year'], format="%Y.0")
         print(df.dtypes['Year'])
         print(df['Year'])
```

```
datetime64[ns]
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]
```

1.10 Extract specific statistics from the preprocessed data:

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

```
In [23]: dftemp = df[df['Area'] == 'Iceland']
         dftemp
```

Out[23]:

	Area	Area Id	Variable Name	Variable Id	Year
166	Iceland	99.0	Total area of the country	4100.0	1962-01-01
167	Iceland	99.0	Total area of the country	4100.0	1967-01-01
168	Iceland	99.0	Total area of the country	4100.0	1972-01-01
169	Iceland	99.0	Total area of the country	4100.0	1977-01-01
170	Iceland	99.0	Total area of the country	4100.0	1982-01-01
171	Iceland	99.0	Total area of the country	4100.0	1987-01-01
172	Iceland	99.0	Total area of the country	4100.0	1992-01-01
173	Iceland	99.0	Total area of the country	4100.0	1997-01-01
174	Iceland	99.0	Total area of the country	4100.0	2002-01-01
175	Iceland	99.0	Total area of the country	4100.0	2007-01-01
176	Iceland	99.0	Total area of the country	4100.0	2012-01-01
177	Iceland	99.0	Total area of the country	4100.0	2014-01-01
178	Iceland	99.0	Total population	4104.0	1962-01-01
179	Iceland	99.0	Total population	4104.0	1967-01-01
180	Iceland	99.0	Total population	4104.0	1972-01-01
181	Iceland	99.0	Total population	4104.0	1977-01-01
182	Iceland	99.0	Total population	4104.0	1982-01-01
183	Iceland	99.0	Total population	4104.0	1987-01-01
184	Iceland	99.0	Total population	4104.0	1992-01-01
185	Iceland	99.0	Total population	4104.0	1997-01-01
186	Iceland	99.0	Total population	4104.0	2002-01-01
187	Iceland	99.0	Total population	4104.0	2007-01-01
188	Iceland	99.0	Total population	4104.0	2012-01-01
189	Iceland	99.0	Total population	4104.0	2015-01-01
190	Iceland	99.0	Population density	4107.0	1962-01-01
191	Iceland	99.0	Population density	4107.0	1967-01-01
192	Iceland	99.0	Population density	4107.0	1972-01-01
193	Iceland	99.0	Population density	4107.0	1977-01-01
194	Iceland	99.0	Population density	4107.0	1982-01-01
195	Iceland	99.0	Population density	4107.0	1987-01-01
196	Iceland	99.0	Population density	4107.0	1992-01-01
197	Iceland	99.0	Population density	4107.0	1997-01-01
198	Iceland	99.0	Population density	4107.0	2002-01-01
199	Iceland	99.0	Population density	4107.0	2007-01-01
200	Iceland	99.0	Population density	4107.0	2012-01-01
201	Iceland	99.0	Population density	4107.0	2015-01-01
202	Iceland	99.0	Gross Domestic Product (GDP)	4112.0	1962-01-01
203	Iceland	99.0	Gross Domestic Product (GDP)	4112.0	1967-01-01
204	Iceland	99.0	Gross Domestic Product (GDP)	4112.0	1972-01-01
205	Iceland	99.0	Gross Domestic Product (GDP)	4112.0	1977-01-01
206	Iceland	99.0	Gross Domestic Product (GDP)	4112.0	1982-01-01

```
207  Iceland     99.0   Gross Domestic Product (GDP)       4112.0 1987-01-01
208  Iceland     99.0   Gross Domestic Product (GDP)       4112.0 1992-01-01
209  Iceland     99.0   Gross Domestic Product (GDP)       4112.0 1997-01-01
210  Iceland     99.0   Gross Domestic Product (GDP)       4112.0 2002-01-01
211  Iceland     99.0   Gross Domestic Product (GDP)       4112.0 2007-01-01
212  Iceland     99.0   Gross Domestic Product (GDP)       4112.0 2012-01-01
213  Iceland     99.0   Gross Domestic Product (GDP)       4112.0 2015-01-01
214  Iceland     99.0  National Rainfall Index (NRI)       4472.0 1967-01-01
215  Iceland     99.0  National Rainfall Index (NRI)       4472.0 1971-01-01
216  Iceland     99.0  National Rainfall Index (NRI)       4472.0 1975-01-01
217  Iceland     99.0  National Rainfall Index (NRI)       4472.0 1981-01-01
218  Iceland     99.0  National Rainfall Index (NRI)       4472.0 1986-01-01
219  Iceland     99.0  National Rainfall Index (NRI)       4472.0 1991-01-01
220  Iceland     99.0  National Rainfall Index (NRI)       4472.0 1997-01-01
221  Iceland     99.0  National Rainfall Index (NRI)       4472.0 1998-01-01
```

```
            Value Symbol
166  1.030000e+04      E
167  1.030000e+04      E
168  1.030000e+04      E
169  1.030000e+04      E
170  1.030000e+04      E
171  1.030000e+04      E
172  1.030000e+04      E
173  1.030000e+04      E
174  1.030000e+04      E
175  1.030000e+04      E
176  1.030000e+04      E
177  1.030000e+04      E
178  1.826000e+02      E
179  1.974000e+02      E
180  2.099000e+02      E
181  2.221000e+02      E
182  2.331000e+02      E
183  2.469000e+02      E
184  2.599000e+02      E
185  2.728000e+02      E
186  2.869000e+02      E
187  3.054000e+02      E
188  3.234000e+02      E
189  3.294000e+02      E
190  1.773000e+00      E
191  1.917000e+00      E
192  2.038000e+00      E
193  2.156000e+00      E
194  2.263000e+00      E
195  2.397000e+00      E
196  2.523000e+00      E
```

```
197  2.649000e+00      E
198  2.785000e+00      E
199  2.965000e+00      E
200  3.140000e+00      E
201  3.198000e+00      E
202  2.849165e+08      E
203  6.212260e+08      E
204  8.465069e+08      E
205  2.226539e+09      E
206  3.232804e+09      E
207  5.565384e+09      E
208  7.138788e+09      E
209  7.596126e+09      E
210  9.161798e+09      E
211  2.129384e+10      E
212  1.419452e+10      E
213  1.659849e+10      E
214  8.160000e+02      E
215  9.632000e+02      E
216  1.010000e+03      E
217  9.326000e+02      E
218  9.685000e+02      E
219  1.095000e+03      E
220  9.932000e+02      E
221  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]: temp = dftemp[dftemp['Variable Name'] == "National Rainfall Index (NRI)"]
         temp = temp[(temp['Value'] < 950) & (temp['Value'] > 900)]
         temp['Year'].unique()

Out[24]: array(['1981-01-01T00:00:00.000000000', '1998-01-01T00:00:00.000000000'],
               dtype='datetime64[ns]')

In [ ]:
```

1.11 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[df['Area'] == 'United States of America']
         df_usa = df_usa.set_index('Year')
         df_usa.head()
```

```
Out[25]:                                Area  Area Id             Variable Name  \
         Year
         1962-01-01  United States of America    231.0  Total area of the country
         1967-01-01  United States of America    231.0  Total area of the country
         1972-01-01  United States of America    231.0  Total area of the country
         1977-01-01  United States of America    231.0  Total area of the country
         1982-01-01  United States of America    231.0  Total area of the country

                     Variable Id     Value Symbol
         Year
         1962-01-01       4100.0  962909.0      E
         1967-01-01       4100.0  962909.0      E
         1972-01-01       4100.0  962909.0      E
         1977-01-01       4100.0  962909.0      E
         1982-01-01       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)  \
         Year
         1962-01-01                   6.050000e+11                            NaN
         1965-01-01                            NaN                          928.5
         1967-01-01                   8.620000e+11                            NaN
         1969-01-01                            NaN                          952.2
         1972-01-01                   1.280000e+12                            NaN

         Variable Name  Population density  Total area of the country  Total population
         Year
         1962-01-01                   19.93                   962909.0          191861.0
         1965-01-01                     NaN                        NaN               NaN
         1967-01-01                   21.16                   962909.0          203713.0
         1969-01-01                     NaN                        NaN               NaN
         1972-01-01                   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]:

Year	GDP	NRI	PD	Area	Population
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

1.12 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'] *= 10
         df_usa.head()
```

Out[29]:

Year	GDP	NRI	PD	Area	Population
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['GDP/capita'] = np.round(df_usa['GDP/capita'], 2)
         df_usa.head()
```

Out[30]:

Year	GDP	NRI	PD	Area	Population	GDP/capita
1962-01-01	6.050000e+11	0.0	19.93	9629090.0	191861.0	3153324.54
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	4231443.26
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	6003189.19

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]: # your code here
         maxVal = df_usa.max()['NRI']
         print(maxVal)
         df_usa['NRI'].idxmax()
```

```
1020.0
```

```
Out[31]: Timestamp('1992-01-01 00:00:00')
```