Homework 03

Name: Isha Mangal

Student ID: 3031911156

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 [119]:
```
import numpy as np
```

In [120]:
```
# a is an array of all integers between 25-34 (inclusive)
a = np.arange(25, 35, 1)
a
```

Out[120]:
```
array([25, 26, 27, 28, 29, 30, 31, 32, 33, 34])
```

In [121]:
```
# b is an array of ten evenly spaced numbers between 1-6
b = np.linspace(1, 6, 10)
b
```

Out[121]:
```
array([ 1.        ,  1.55555556,  2.11111111,  2.66666667,  3.22222222,
        3.77777778,  4.33333333,  4.88888889,  5.44444444,  6.        ])
```

In [122]:
```
a = a ** 3
a
```

Out[122]:
```
array([15625, 17576, 19683, 21952, 24389, 27000, 29791, 32768, 35937, 39304])
```

In [123]:
```
b = b ** 3
b
```

Out[123]:
```
array([   1.        ,    3.76406036,    9.40877915,   18.96296296,
         33.45541838,   53.91495199,   81.37037037,  116.85048011,
        161.38408779,  216.        ])
```

In [124]:
```
cube_sum = a + b
cube_sum
```

Out[124]:
```
array([ 15626.        ,  17579.76406036,  19692.40877915,  21970.96296296,
        24422.45541838,  27053.91495199,  29872.37037037,  32884.85048011,
        36098.38408779,  39520.        ])
```

In [125]:
```
even_sum = np.sum(cube_sum[::2])
even_sum
```

Out[125]:
```
125711.61865569273
```

In [126]:
```
square_root = np.sqrt(cube_sum)
square_root
```

Out[126]:
```
array([ 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 [127]:
```
a = np.arange(25, 35, 1)
b = np.linspace(1, 6, 10)
m = np.concatenate((a,b)).reshape(4,5)
m
```

Out[127]:
```
array([[ 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 [128]:
```
m2 = m[:, 2:4]
m2
```

Out[128]:
```
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 =A^TB

In [129]:
```
m3 = np.dot(m2.T, m)
m3
```

Out[129]:
```
array([[ 1655.58024691,  1718.4691358 ,  1781.35802469,  1844.24691358,
         1907.13580247],
       [ 1713.2345679 ,  1778.74074074,  1844.24691358,  1909.75308642,
         1975.25925926]])
```

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 [130]:
```
f = np.cos(np.linspace(0, np.pi, 50))
f
```

Out[130]:
```
array([ 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.        ])
```

In [131]:
```
f2 = 1/2 <= f
f2
```

Out[131]:
```
array([ 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], dtype=bool)
```

In [132]:
```
f3 = f[np.where(f2)]
f3
```

Out[132]:
```
array([ 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 [133]:
```
# 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 [134]:
```
exp_x = np.mean(x)
exp_x
```

Out[134]: 5.7825325415879227

In [135]:
```
exp_y = np.mean(y)
exp_y
```

Out[135]: 11.012981683344968

3b) Find variance of distributions of x and y

In [136]:
```
var_x = np.var(x)
var_x
```

Out[136]: 7.0333275294758497

In [137]:
```
var_y = np.var(y)
var_y
```

Out[137]: 30.113903575509635

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

In [138]:
```
covar_xy = np.mean(x*y) - (exp_x*exp_y)
covar_xy
```

Out[138]: 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 [139]:
```
m = covar_xy/var_x
b = exp_y - (covar_xy/var_x) * exp_x
# define linear predictor function
def linearPredictor(x):
    predicted = m * x + b
    return predicted
```

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

In [140]:
```
predict_y = linearPredictor(x)
predict_y
y_error = y - predict_y
y_error
```

Out[140]:
```
array([-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 [141]:
```
RMSE = np.sqrt(np.mean(y_error ** 2))
RMSE
```

Out[141]:
```
0.41767772366856115
```

Pandas Introduction

Reading File

In [142]:
```
# 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 [143]:
```
df = pd.read_csv('data3.csv')
```

4a) Display the first 10 rows of the dataframe

In [144]:
```
df.head(10)
```

Out[144]:

	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 [145]:
```
list(df)
```

Out[145]:
```
['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 [146]:
```
df.iloc[:3,:4]
```

Out[146]:

	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 [147]:
```
df[df['Symbol'].isnull()].head()
```

Out[147]:

	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 [148]:
```
df = df.drop(labels='Other',axis=1).dropna()
df.tail()
```

Out[148]:

	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 [155]:
```
df1 = df.drop(['Area Id', 'Variable Id', 'Symbol'], axis=1)
df1.head()
```

Out[155]:

	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 [157]:
```
print (df1['Area'].unique())
```

```
['Argentina' 'Australia' 'Germany' 'Iceland' 'Ireland' 'Sweden'
 'United States of America']
```

In [158]:
```
print (df1['Variable Name'].unique())
```

```
['Total area of the country' 'Total population' 'Population density'
 'Gross Domestic Product (GDP)' 'National Rainfall Index (NRI)']
```

In [159]:
```
print (df1['Year'].unique())
```

```
[ 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 [160]:
```
df1['Year'] = pd.to_datetime(df1['Year'].astype(int),
               format= '%Y')
df1.head()
```

Out[160]:

	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 [161]:
```
dftemp = df1[df1['Area'] =='Iceland']
dftemp
```

Out[161]:

	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
198	Iceland	Population density	2002-01-01	2.785000e+00
199	Iceland	Population density	2007-01-01	2.965000e+00

	Area	Variable Name	Year	Value
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 [162]:
```
dftemp[(dftemp['Variable Name']=='National Rainfall Index (NRI)')
      & (dftemp['Value'].between(900,950,inclusive=False))]
```

Out[162]:

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

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 [163]:
```
df_usa = df1[df1['Area']==
        'United States of America'].set_index('Year')
df_usa.head()
```

Out[163]:

	Area	Variable Name	Value
Year			
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 [164]:
```
df_usa = df_usa.pivot(columns='Variable Name')
df_usa.head()
```

Out[164]:

	Area					Value			
Variable Name	**Gross Domestic Product (GDP)**	**National Rainfall Index (NRI)**	**Population density**	**Total area of the country**	**Total population**	**Gross Domestic Product (GDP)**	**National Rainfall Index (NRI)**	**Population density**	**Total area of the country**
Year									
1962-01-01	United States of America	None	United States of America	United States of America	United States of America	6.050000e+11	NaN	19.93	962909
1965-01-01	None	United States of America	None	None	None	NaN	928.5	NaN	Na
1967-01-01	United States of America	None	United States of America	United States of America	United States of America	8.620000e+11	NaN	21.16	962909
1969-01-01	None	United States of America	None	None	None	NaN	952.2	NaN	Na
1972-01-01	United States of America	None	United States of America	United States of America	United States of America	1.280000e+12	NaN	22.14	962909

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

In [165]:
```
df_usa = 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'})
df_usa
```

Out[165]:

	Area					**Value**				
Variable Name	**GDP**	**NRI**	**PD**	**Area**	**Population**	**GDP**	**NRI**	**PD**	**Area**	**Populati**
Year										
1962-01-01 00:00:00	United States of America	None	United States of America	United States of America	United States of America	6.050000e+11	NaN	19.93	962909.0	191861
1965-01-01 00:00:00	None	United States of America	None	None	None	NaN	928.5	NaN	NaN	Na
1967-01-01 00:00:00	United States of America	None	United States of America	United States of America	United States of America	8.620000e+11	NaN	21.16	962909.0	203713
1969-01-01 00:00:00	None	United States of America	None	None	None	NaN	952.2	NaN	NaN	Na
1972-01-01 00:00:00	United States of America	None	United States of America	United States of America	United States of America	1.280000e+12	NaN	22.14	962909.0	213220
1974-01-01 00:00:00	None	United States of America	None	None	None	NaN	1008.0	NaN	NaN	Na
1977-01-01 00:00:00	United States of America	None	United States of America	United States of America	United States of America	2.090000e+12	NaN	23.17	962909.0	223091
1981-01-01 00:00:00	None	United States of America	None	None	None	NaN	949.2	NaN	NaN	Na
1982-01-01 00:00:00	United States of America	None	United States of America	United States of America	United States of America	3.340000e+12	NaN	24.30	962909.0	233954
1984-01-01 00:00:00	None	United States of America	None	None	None	NaN	974.6	NaN	NaN	Na
1987-01-01 00:00:00	United States of America	None	United States of America	United States of America	United States of America	4.870000e+12	NaN	25.49	962909.0	245425

	Area					Value				
Variable Name	GDP	NRI	PD	Area	Population	GDP	NRI	PD	Area	Populati
Year										
1992-01-01 00:00:00	United States of America	United States of America	United States of America	United States of America	United States of America	6.540000e+12	1020.0	26.78	962909.0	257908
1996-01-01 00:00:00	None	United States of America	None	None	None	NaN	1005.0	NaN	NaN	Na
1997-01-01 00:00:00	United States of America	None	United States of America	United States of America	United States of America	8.610000e+12	NaN	28.34	962909.0	272883
2002-01-01 00:00:00	United States of America	United States of America	United States of America	United States of America	United States of America	1.100000e+13	938.7	29.95	963203.0	288471
2007-01-01 00:00:00	United States of America	None	United States of America	United States of America	United States of America	1.450000e+13	NaN	31.32	963203.0	301656
2012-01-01 00:00:00	United States of America	None	United States of America	United States of America	United States of America	1.620000e+13	NaN	32.02	983151.0	314799
2014-01-01 00:00:00	None	None	None	United States of America	None	NaN	NaN	NaN	983151.0	Na
2015-01-01 00:00:00	United States of America	None	United States of America	None	United States of America	1.790000e+13	NaN	32.73	NaN	321774

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

In [166]:
```
df_usa = df_usa.fillna(0)
df_usa.head()
```

Out[166]:

	Area					Value				
Variable Name	GDP	NRI	PD	Area	Population	GDP	NRI	PD	Area	Populatio
Year										
1962-01-01 00:00:00	United States of America	0	United States of America	United States of America	United States of America	6.050000e+11	0.0	19.93	962909.0	191861.
1965-01-01 00:00:00	0	United States of America	0	0	0	0.000000e+00	928.5	0.00	0.0	0.
1967-01-01 00:00:00	United States of America	0	United States of America	United States of America	United States of America	8.620000e+11	0.0	21.16	962909.0	203713.
1969-01-01 00:00:00	0	United States of America	0	0	0	0.000000e+00	952.2	0.00	0.0	0.
1972-01-01 00:00:00	United States of America	0	United States of America	United States of America	United States of America	1.280000e+12	0.0	22.14	962909.0	213220.

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 [167]:
```
df_usa[('Value','Area')] = df_usa[('Value',
                    'Area')].apply(lambda x: x*10)
df_usa.head()
```

Out[167]:

	Area					Value				
Variable Name	**GDP**	**NRI**	**PD**	**Area**	**Population**	**GDP**	**NRI**	**PD**	**Area**	**Populati**
Year										
1962-01-01 00:00:00	United States of America	0	United States of America	United States of America	United States of America	6.050000e+11	0.0	19.93	9629090.0	191861
1965-01-01 00:00:00	0	United States of America	0	0	0	0.000000e+00	928.5	0.00	0.0	(
1967-01-01 00:00:00	United States of America	0	United States of America	United States of America	United States of America	8.620000e+11	0.0	21.16	9629090.0	203713
1969-01-01 00:00:00	0	United States of America	0	0	0	0.000000e+00	952.2	0.00	0.0	(
1972-01-01 00:00:00	United States of America	0	United States of America	United States of America	United States of America	1.280000e+12	0.0	22.14	9629090.0	213220

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 [168]:
```
df_usa['GDP/capita'] = 1000 * (df_usa[('Value',
                        'GDP')].values/df_usa[('Value',
                         'Population')].values)
df_usa.head()
```

```
/Users/ishamangal/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:
3: RuntimeWarning: invalid value encountered in true_divide
  This is separate from the ipykernel package so we can avoid doing imports un
til
```

Out[168]:

	Area					**Value**				
Variable Name	**GDP**	**NRI**	**PD**	**Area**	**Population**	**GDP**	**NRI**	**PD**	**Area**	**Populati**
Year										
1962-01-01 00:00:00	United States of America	0	United States of America	United States of America	United States of America	6.050000e+11	0.0	19.93	9629090.0	191861
1965-01-01 00:00:00	0	United States of America	0	0	0	0.000000e+00	928.5	0.00	0.0	(
1967-01-01 00:00:00	United States of America	0	United States of America	United States of America	United States of America	8.620000e+11	0.0	21.16	9629090.0	203713
1969-01-01 00:00:00	0	United States of America	0	0	0	0.000000e+00	952.2	0.00	0.0	(
1972-01-01 00:00:00	United States of America	0	United States of America	United States of America	United States of America	1.280000e+12	0.0	22.14	9629090.0	213220

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 [170]:
```
df_usa.sort_values(('Value','NRI'), ascending=False).head()
```

Out[170]:

	Area					Value				
Variable Name	**GDP**	**NRI**	**PD**	**Area**	**Population**	**GDP**	**NRI**	**PD**	**Area**	**Populat**
Year										
1992-01-01 00:00:00	United States of America	United States of America	United States of America	United States of America	United States of America	6.540000e+12	1020.0	26.78	9629090.0	2579C
1974-01-01 00:00:00	0	United States of America	0	0	0	0.000000e+00	1008.0	0.00	0.0	
1996-01-01 00:00:00	0	United States of America	0	0	0	0.000000e+00	1005.0	0.00	0.0	
1984-01-01 00:00:00	0	United States of America	0	0	0	0.000000e+00	974.6	0.00	0.0	
1969-01-01 00:00:00	0	United States of America	0	0	0	0.000000e+00	952.2	0.00	0.0	

In [ ]: