Homework 03

Name: Jack Xie

Student ID: 3032163590

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 [28]:
```
import numpy as np

a = np.arange(25, 35)
b = np.arange(1, 6, 0.5)
print(a)
print(b)
```

```
[25 26 27 28 29 30 31 32 33 34]
[1.  1.5 2.  2.5 3.  3.5 4.  4.5 5.  5.5]
```

In [31]:
```
print(a**3)
print(b**3)
```

```
[15625 17576 19683 21952 24389 27000 29791 32768 35937 39304]
[  1.      3.375   8.     15.625  27.     42.875  64.     91.125 125.
 166.375]
```

In [32]:
```
c = a**3+b**3
print(c)
```

```
[15626.    17579.375 19691.    21967.625 24416.    27042.875 29855.
 32859.125 36062.    39470.375]
```

In [33]:
```
sum([c[i] for i in np.arange(10) if i%2==0])
```

Out[33]:
```
125650.0
```

In [61]:
```
print(np.sqrt(c))
```

```
[125.00399994 132.58723543 140.32462364 148.21479346 156.25619988
 164.44717997 172.7859948  181.27086087 189.89997367 198.67152539]
```

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 [35]:
```
m = np.hstack([a,b]).reshape(4, 5)
print(m)
```

```
[[25.  26.  27.  28.  29. ]
 [30.  31.  32.  33.  34. ]
 [ 1.   1.5  2.   2.5  3. ]
 [ 3.5  4.   4.5  5.   5.5]]
```

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 [38]:
```
m2 = np.hstack([m[:, 2], m[:, 3]]).reshape(4, 2)
print(m2)
```

```
[[27.  32. ]
 [ 2.   4.5]
 [28.  33. ]
 [ 2.5  5. ]]
```

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 [42]:
```
m3 = m.T.dot(m2)
print(m3)
```

```
[[ 771.75  985.5 ]
 [ 816.   1041.  ]
 [ 860.25 1096.5 ]
 [ 904.5  1152.  ]
 [ 948.75 1207.5 ]]
```

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 [45]:
```
f = [np.cos(x) for x in np.arange(0, np.pi, np.pi/50)]
print(f)
```

```
[1.0, 0.9980267284282716, 0.9921147013144779, 0.9822872507286887, 0.968
5831611286311, 0.9510565162951535, 0.9297764858882513, 0.90482705246601
95, 0.8763066800438636, 0.8443279255020151, 0.8090169943749475, 0.77051
32427757891, 0.7289686274214116, 0.6845471059286887, 0.637423989748689
6, 0.5877852522924731, 0.5358267949789967, 0.4817536741017152, 0.425779
29156507266, 0.36812455268467786, 0.30901699437494745, 0.24868988716485
474, 0.18738131458572452, 0.12533323356430426, 0.06279051952931329, -1.
6081226496766366e-16, -0.0627905195293134, -0.12533323356430437, -0.187
38131458572482, -0.24868988716485485, -0.3090169943749756, -0.36812455
2684678, -0.42577929156507277, -0.4817536741017154, -0.535826794978996
8, -0.587785252292473, -0.6374239897486896, -0.6845471059286887, -0.728
9686274214117, -0.7705132427757895, -0.8090169943749473, -0.84432792550
20151, -0.8763066800438637, -0.9048270524660196, -0.9297764858882515, -
0.9510565162951536, -0.9685831611286311, -0.9822872507286887, -0.992114
7013144779, -0.9980267284282716]
```

In [60]:
```
print([f[i]>=0.5 for i in np.arange(50)])
```

```
[True, True, True, True, True, True, True, True, True, True, True, Tru
e, True, True, True, True, True, False, False, False, False, False, Fal
se, False, False, False, False, False, False, False, False, False, Fals
e, False, False, False, False, False, False, False, False, False, Fals
e, False, False, False, False, False, False, False]
```

In [66]:
```
print([f[i] for i in np.arange(50) if f[i]>=1/2])
```

```
[1.0, 0.9980267284282716, 0.9921147013144779, 0.9822872507286887, 0.968
5831611286311, 0.9510565162951535, 0.9297764858882513, 0.90482705246601
95, 0.8763066800438636, 0.8443279255020151, 0.8090169943749475, 0.77051
32427757891, 0.7289686274214116, 0.6845471059286887, 0.637423989748689
6, 0.5877852522924731, 0.5358267949789967]
```

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 [71]:
```
# 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 [74]:
```
print(np.mean(x))
print(np.mean(y))
```

```
5.782532541587923
11.012981683344968
```

3b) Find variance of distributions of x and y

In [75]:
```
print(np.var(x))
print(np.var(y))
```

```
7.03332752947585
30.113903575509635
```

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

In [99]:
```
cov = np.mean(x*y) - np.mean(x)*np.mean(y)

print(cov)
```

```
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 [100]:
```
slope = cov/np.var(x)
y_intc = np.mean(y)- slope*np.mean(x)
y_predicted = np.array([x[i]*slope - y_intc for i in np.arange(100)])
print(y_predicted)
```

```
[ 3.69634436  3.23197529  4.04354848  4.12031556  3.81879927  4.3233824
6
  4.61891188  4.63633533  4.83441039  5.04600871  5.73388719  5.5066151
5
  5.5579814   5.7922365   6.07160155  6.7383907   6.45625697  6.4516550
6
  7.30338026  7.01850825  7.29381883  7.63209848  7.9642386   7.9507137
2
  8.15262477  8.65373429  8.44536569  8.70023834  8.27014241  9.1928910
9
  9.49283906  9.29596544  9.22228092  9.68964175  9.49834387  9.7240132
5
  9.84131263 10.172302   10.78976757 10.91612042 11.16543614 11.6204851
9
 11.84388348 11.89578883 11.37262877 12.21565391 12.2002125  12.5650843
6
 12.25777792 13.08537354 12.64784904 12.80727707 13.1856081  13.6709240
5
 13.22499164 13.35358351 14.41686351 14.32655925 14.8235941  14.9846484
1
 14.34097135 15.18537609 15.38575769 15.15167711 16.00602822 15.7823332
 15.57526856 16.35440162 16.57635454 16.63385856 16.80247809 17.1226817
4
 17.49558385 17.02676269 17.88551724 17.41007374 18.2604188  18.0196988
8
 18.37163394 18.54196414 18.84808983 18.69937322 19.14571326 18.7912506
6
 19.37984736 19.91735726 19.40686504 19.33384431 19.7593407  20.6394707
8
 20.75498021 20.60569788 20.35321396 21.36288409 21.56315943 21.6012187
7
 21.7008027  21.41523464 21.73365717 22.30282516]
```

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

In [102]:
```
y_error = y_predicted - y
print(y_error)
```

```
[2.03284316 1.21051609 1.93538795 1.8601506  1.85592369 1.36791896
 1.59009301 1.30068237 2.07975261 1.62099801 2.33718571 2.11283748
 1.77294797 1.40930601 1.74196569 2.6991403  1.71933829 2.15067108
 2.46174736 2.23674868 2.29616096 2.31463031 2.19579189 2.01347623
 2.42450835 1.94399814 1.76393202 2.12541103 1.09276638 1.64425856
 2.19062487 1.97570815 1.44204208 1.77830809 1.22180217 1.03198044
 1.05911398 1.71332654 1.89882042 2.09892121 2.29436644 1.95855957
 2.44182099 2.03588101 1.76903099 2.14179125 1.504413   1.89786519
 1.07521508 2.51105517 1.18040188 1.86328791 1.92115551 1.57337577
 1.10511127 1.23202651 2.24072658 1.88905732 1.81447038 2.12057647
 1.09456269 2.42417524 2.26852707 1.07326612 1.80781115 1.50944319
 1.26901915 1.72379327 2.29926275 1.55936626 2.3498619  2.00370861
 2.25771714 1.75473145 2.56059831 1.39912103 2.54791322 1.72481382
 1.66544461 1.9764102  2.42429526 1.51792578 2.0075735  1.09511441
 2.00221717 2.00792887 1.50342771 1.31433262 1.40206156 2.47105809
 2.13684272 1.94507034 1.54103413 1.91293216 1.84177273 1.88155151
 1.91118366 1.77138376 1.03645908 2.22308197]
```

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

In [104]:
```
np.sqrt(np.mean(y_error**2))
```

Out[104]: 1.882020108128031

Pandas Introduction

Reading File

In [105]:
```
# 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 [346]:
```
df = pd.read_csv("data3.csv")
```

4a) Display the first 10 rows of the dataframe

In [347]:
```
df.head(10)
```

Out[347]:

	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 [348]:
```
list(df)
```

Out[348]:
```
['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 [349]:
```
df.iloc[0 : 3, 0 : 4]
```

Out[349]:

	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 [350]:
```
df[df['Symbol'] != 'E'].head()
```

Out[350]:

	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 [351]:
```
pd.isna(df).iloc[390, 4] == False
```

Out[351]:
```
False
```

In [352]:
```
del df['Other']
```

In [353]:
```
df.index
```

Out[353]:
```
RangeIndex(start=0, stop=398, step=1)
```

In [354]:
```
df.iloc[390]
```

Out[354]:
```
Area             NaN
Area Id          NaN
Variable Name    NaN
Variable Id      NaN
Year             NaN
Value            NaN
Symbol           NaN
Name: 390, dtype: object
```

In [371]:
```
df2 = df.copy()
```

In [372]:
```
for i in np.arange(398):
    for j in np.arange(7):
            if pd.isna(df).iloc[i, j] == True:
                df2.drop(i, axis=0, inplace=True)
                break
```

In [378]:
```
df2.tail(5)
```

Out[378]:

	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 [380]:
```
df1 = df2.drop(['Area Id', 'Variable Id', 'Symbol'], axis=1)
df1.head()
```

Out[380]:

	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 [388]:
```
pd.unique(df1['Value'])
```

Out[388]:
```
array([2.78040000e+05, 2.12880000e+04, 2.29320000e+04, 2.47830000e+04,
       2.68790000e+04, 2.89940000e+04, 3.13260000e+04, 3.36550000e+04,
       3.58340000e+04, 3.78890000e+04, 3.99700000e+04, 4.20950000e+04,
       4.34170000e+04, 7.65600000e+00, 8.24800000e+00, 8.91300000e+00,
       9.66700000e+00, 1.04300000e+01, 1.12700000e+01, 1.21000000e+01,
       1.28900000e+01, 1.36300000e+01, 1.43800000e+01, 1.51400000e+01,
       1.56200000e+01, 2.44506049e+10, 2.42566676e+10, 3.47330005e+10,
       5.67810001e+10, 8.43074868e+10, 1.11000000e+11, 2.29000000e+11,
       2.93000000e+11, 9.77245135e+10, 3.29000000e+11, 6.04000000e+11,
       5.48000000e+11, 7.81600000e+02, 8.03200000e+02, 8.48900000e+02,
       8.91000000e+02, 9.68600000e+02, 9.58800000e+02, 8.87300000e+02,
       1.06200000e+03, 7.74122000e+05, 1.06910000e+04, 1.19760000e+04,
       1.33640000e+04, 1.42120000e+04, 1.51010000e+04, 1.63150000e+04,
       1.75380000e+04, 1.85130000e+04, 1.95140000e+04, 2.09760000e+04,
       2.29110000e+04, 2.39690000e+04, 1.38100000e+00, 1.54700000e+00,
       1.72600000e+00, 1.83600000e+00, 1.95100000e+00, 2.10800000e+00,
       2.26600000e+00, 2.39100000e+00, 2.52100000e+00, 2.71000000e+00,
       2.96000000e+00, 3.09600000e+00, 1.98835256e+10, 3.03785418e+10,
       5.19287383e+10, 1.10000000e+11, 1.94000000e+11, 1.89000000e+11,
       3.25000000e+11, 4.36000000e+11, 3.94000000e+11, 8.53000000e+11,
       1.54000000e+12, 1.34000000e+12, 5.96400000e+02, 6.72100000e+02,
       7.68500000e+02, 6.93400000e+02, 6.37000000e+02, 6.94700000e+02,
       6.35500000e+02, 7.17100000e+02, 3.56970000e+04, 3.57030000e+04,
       3.57040000e+04, 3.57100000e+04, 3.57170000e+04, 3.57380000e+04,
       7.42380000e+04, 7.71070000e+04, 7.87000000e+04, 7.85250000e+04,
       7.78120000e+04, 7.78640000e+04, 8.00760000e+04, 8.19940000e+04,
       8.17000000e+04, 8.08550000e+04, 8.04780000e+04, 8.06890000e+04,
       2.08000000e+02, 2.16000000e+02, 2.20500000e+02, 2.20000000e+02,
       2.18000000e+02, 2.18100000e+02, 2.24300000e+02, 2.29700000e+02,
       2.28800000e+02, 2.26400000e+02, 2.25300000e+02, 2.25800000e+02,
       2.99000000e+11, 5.98000000e+11, 7.74000000e+11, 1.29000000e+12,
       2.12000000e+12, 2.22000000e+12, 2.08000000e+12, 3.44000000e+12,
       3.54000000e+12, 3.36000000e+12, 7.85800000e+02, 6.78200000e+02,
       6.71900000e+02, 7.75100000e+02, 7.55600000e+02, 7.39400000e+02,
       7.90900000e+02, 8.35500000e+02, 1.03000000e+04, 1.82600000e+02,
       1.97400000e+02, 2.09900000e+02, 2.22100000e+02, 2.33100000e+02,
       2.46900000e+02, 2.59900000e+02, 2.72800000e+02, 2.86900000e+02,
       3.05400000e+02, 3.23400000e+02, 3.29400000e+02, 1.77300000e+00,
       1.91700000e+00, 2.03800000e+00, 2.15600000e+00, 2.26300000e+00,
       2.39700000e+00, 2.52300000e+00, 2.64900000e+00, 2.78500000e+00,
       2.96500000e+00, 3.14000000e+00, 3.19800000e+00, 2.84916516e+08,
       6.21225962e+08, 8.46506911e+08, 2.22653869e+09, 3.23280422e+09,
       5.56538403e+09, 7.13878800e+09, 7.59612605e+09, 9.16179822e+09,
       2.12938412e+10, 1.41945190e+10, 1.65984948e+10, 8.16000000e+02,
       9.63200000e+02, 1.01000000e+03, 9.32600000e+02, 9.68500000e+02,
       1.09500000e+03, 9.93200000e+02, 9.23400000e+02, 7.02800000e+03,
       2.82200000e+03, 2.88200000e+03, 3.02800000e+03, 3.28800000e+03,
       3.50100000e+03, 3.56600000e+03, 3.58200000e+03, 3.70600000e+03,
       3.96800000e+03, 4.38900000e+03, 4.66800000e+03, 4.68800000e+03,
       4.01500000e+01, 4.10100000e+01, 4.30800000e+01, 4.67800000e+01,
       4.98200000e+01, 5.07400000e+01, 5.09700000e+01, 5.27300000e+01,
       5.64600000e+01, 6.24500000e+01, 6.64200000e+01, 6.67000000e+01,
       2.26034968e+09, 3.34363677e+09, 6.32515932e+09, 1.12609772e+10,
       2.14988799e+10, 3.39586277e+10, 5.59813610e+10, 8.28167796e+10,
       1.28000000e+11, 2.70000000e+11, 2.25000000e+11, 2.38000000e+11,
       1.09800000e+03, 1.03000000e+03, 9.91600000e+02, 1.15400000e+03,
       1.07100000e+03, 1.12800000e+03, 1.12200000e+03, 1.20200000e+03,
```

```
4.50300000e+04, 4.47420000e+04, 7.57600000e+03, 7.87100000e+03,
8.12200000e+03, 8.24600000e+03, 8.32300000e+03, 8.41400000e+03,
8.67800000e+03, 8.85900000e+03, 8.91200000e+03, 9.15300000e+03,
9.54300000e+03, 9.77900000e+03, 1.68200000e+01, 1.74800000e+01,
1.80400000e+01, 1.83100000e+01, 1.84800000e+01, 1.86900000e+01,
1.92700000e+01, 1.96700000e+01, 1.97900000e+01, 2.03300000e+01,
2.13300000e+01, 2.18600000e+01, 1.75114773e+10, 2.74634092e+10,
4.82639150e+10, 9.31367751e+10, 1.13000000e+11, 1.80000000e+11,
2.80000000e+11, 2.64000000e+11, 4.88000000e+11, 5.44000000e+11,
4.93000000e+11, 6.14100000e+02, 5.81700000e+02, 6.14800000e+02,
6.15900000e+02, 6.78600000e+02, 6.76500000e+02, 6.46600000e+02,
7.00400000e+02, 9.62909000e+05, 9.63203000e+05, 9.83151000e+05,
1.91861000e+05, 2.03713000e+05, 2.13220000e+05, 2.23091000e+05,
2.33954000e+05, 2.45425000e+05, 2.57908000e+05, 2.72883000e+05,
2.88471000e+05, 3.01656000e+05, 3.14799000e+05, 3.21774000e+05,
1.99300000e+01, 2.11600000e+01, 2.21400000e+01, 2.31700000e+01,
2.43000000e+01, 2.54900000e+01, 2.67800000e+01, 2.83400000e+01,
2.99500000e+01, 3.13200000e+01, 3.20200000e+01, 3.27300000e+01,
6.05000000e+11, 8.62000000e+11, 1.28000000e+12, 2.09000000e+12,
3.34000000e+12, 4.87000000e+12, 6.54000000e+12, 8.61000000e+12,
1.10000000e+13, 1.45000000e+13, 1.62000000e+13, 1.79000000e+13,
9.28500000e+02, 9.52200000e+02, 1.00800000e+03, 9.49200000e+02,
9.74600000e+02, 1.02000000e+03, 1.00500000e+03, 9.38700000e+02])
```

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 [421]:
```
df1['Year'] = pd.to_datetime(df1['Year'].astype(int), format='%Y')
df1['Year']
```

Out[421]:

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 [422]:
```
dftemp = df1[df1['Area']=='Iceland']
dftemp
```

Out[422]:

	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 [431]:
```
dftemp[(dftemp['Variable Name']=='National Rainfall Index (NRI)') & (dftemp['Value']>900) & (dftemp['Value']<950)]
```

Out[431]:

	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

In [434]:
```
print('1981 , 1998')
```

```
1981 , 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 [441]:
```
df_usa = df1[df1['Area']=='United States of America'].set_index('Year')
df_usa
```

Out[441]:

Year	Area	Variable Name	Value
1962-01-01	United States of America	Total area of the country	9.629090e+05
1967-01-01	United States of America	Total area of the country	9.629090e+05
1972-01-01	United States of America	Total area of the country	9.629090e+05
1977-01-01	United States of America	Total area of the country	9.629090e+05
1982-01-01	United States of America	Total area of the country	9.629090e+05
1987-01-01	United States of America	Total area of the country	9.629090e+05
1992-01-01	United States of America	Total area of the country	9.629090e+05
1997-01-01	United States of America	Total area of the country	9.629090e+05
2002-01-01	United States of America	Total area of the country	9.632030e+05
2007-01-01	United States of America	Total area of the country	9.632030e+05
2012-01-01	United States of America	Total area of the country	9.831510e+05
2014-01-01	United States of America	Total area of the country	9.831510e+05
1962-01-01	United States of America	Total population	1.918610e+05
1967-01-01	United States of America	Total population	2.037130e+05
1972-01-01	United States of America	Total population	2.132200e+05
1977-01-01	United States of America	Total population	2.230910e+05
1982-01-01	United States of America	Total population	2.339540e+05
1987-01-01	United States of America	Total population	2.454250e+05
1992-01-01	United States of America	Total population	2.579080e+05
1997-01-01	United States of America	Total population	2.728830e+05
2002-01-01	United States of America	Total population	2.884710e+05
2007-01-01	United States of America	Total population	3.016560e+05
2012-01-01	United States of America	Total population	3.147990e+05
2015-01-01	United States of America	Total population	3.217740e+05
1962-01-01	United States of America	Population density	1.993000e+01
1967-01-01	United States of America	Population density	2.116000e+01
1972-01-01	United States of America	Population density	2.214000e+01
1977-01-01	United States of America	Population density	2.317000e+01
1982-01-01	United States of America	Population density	2.430000e+01
1987-01-01	United States of America	Population density	2.549000e+01
1992-01-01	United States of America	Population density	2.678000e+01
1997-01-01	United States of America	Population density	2.834000e+01
2002-01-01	United States of America	Population density	2.995000e+01

Year	Area	Variable Name	Value
2007-01-01	United States of America	Population density	3.132000e+01
2012-01-01	United States of America	Population density	3.202000e+01
2015-01-01	United States of America	Population density	3.273000e+01
1962-01-01	United States of America	Gross Domestic Product (GDP)	6.050000e+11
1967-01-01	United States of America	Gross Domestic Product (GDP)	8.620000e+11
1972-01-01	United States of America	Gross Domestic Product (GDP)	1.280000e+12
1977-01-01	United States of America	Gross Domestic Product (GDP)	2.090000e+12
1982-01-01	United States of America	Gross Domestic Product (GDP)	3.340000e+12
1987-01-01	United States of America	Gross Domestic Product (GDP)	4.870000e+12
1992-01-01	United States of America	Gross Domestic Product (GDP)	6.540000e+12
1997-01-01	United States of America	Gross Domestic Product (GDP)	8.610000e+12
2002-01-01	United States of America	Gross Domestic Product (GDP)	1.100000e+13
2007-01-01	United States of America	Gross Domestic Product (GDP)	1.450000e+13
2012-01-01	United States of America	Gross Domestic Product (GDP)	1.620000e+13
2015-01-01	United States of America	Gross Domestic Product (GDP)	1.790000e+13
1965-01-01	United States of America	National Rainfall Index (NRI)	9.285000e+02
1969-01-01	United States of America	National Rainfall Index (NRI)	9.522000e+02
1974-01-01	United States of America	National Rainfall Index (NRI)	1.008000e+03
1981-01-01	United States of America	National Rainfall Index (NRI)	9.492000e+02
1984-01-01	United States of America	National Rainfall Index (NRI)	9.746000e+02
1992-01-01	United States of America	National Rainfall Index (NRI)	1.020000e+03
1996-01-01	United States of America	National Rainfall Index (NRI)	1.005000e+03
2002-01-01	United States of America	National Rainfall Index (NRI)	9.387000e+02

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 [442]:
```
df_usa = df_usa.pivot_table(
    index='Year', # the rows (turned into index)
    columns='Variable Name', # the column values
    values='Value', # the field(s) to processed in each group
)
```

In [446]:
```
df_usa.head()
```

Out[446]:

Variable Name / Year	Gross Domestic Product (GDP)	National Rainfall Index (NRI)	Population density	Total area of the country	Total population
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 [451]:
```
df_usa.rename(columns={'Gross Domestic Product (GDP)':'GDP','National Rainfall Index (NRI)':'NRI',
                       'Population density':'PD','Total area of the country':'Area',
                       'Total population':'Population'
                      },
              inplace=True)
```

In [452]:
```
df_usa.head()
```

Out[452]:

Variable Name / Year	GDP	NRI	PD	Area	Population
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 [454]:
```
df_usa.fillna(0, inplace=True)
df_usa.head()
```

Out[454]:

Variable Name	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 [457]:
```
df_usa['Area'] = df_usa['Area']*10
df_usa.head()
```

Out[457]:

Variable Name	GDP	NRI	PD	Area	Population
Year					
1962-01-01	6.050000e+11	0.0	19.93	96290900.0	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	96290900.0	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	96290900.0	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 [459]:
```
df_usa['GDP/capita'] = df_usa['GDP']/df_usa['Population']
df_usa.head()
```

Out[459]:

Variable Name	GDP	NRI	PD	Area	Population	GDP/capita
Year						
1962-01-01	6.050000e+11	0.0	19.93	96290900.0	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	96290900.0	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	96290900.0	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 [466]:
```
df_usa[df_usa['NRI']==df_usa['NRI'].max()]
```

Out[466]:

Variable Name	GDP	NRI	PD	Area	Population	GDP/capita
Year						
1992-01-01	6.540000e+12	1020.0	26.78	96290900.0	257908.0	2.535788e+07

In [474]:
```
print('The max NRI value is 1020.0 and the year is 1992')
```

```
The max NRI value is 1020.0 and the year is 1992
```

In [ ]:
```

```