# Homework 1

In this homework, you will get a chance to do some exercises with Numpy, Pandas, and Matplotlib to show us your understanding with this libraries.

If you have questions, Google! Additionally you can ask your peers questions on Piazza and/or go to Office Hours.

This homework is due **Thursday Feb. 8th, 2018 at 11:59 PM**. Please upload your .ipynb to your private repo on Github. Additionally, submit a pdf on bCourses and in the comment section include a link to your private repo.

This homework is long, please start early!

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

## NumPy Basics

Create two numpy arrays (a and b). a should be all integers between 10-19 (inclusive), and b should be ten evenly spaced numbers between 1-7. Print the results below.

In [2]:

```
a = np.array(range(10,20))
b = np.linspace(1,7,10)
print("a: ", a)
print("b: ", b)
```

```
a:  [10 11 12 13 14 15 16 17 18 19]
b:  [ 1.          1.66666667  2.33333333  3.          3.66666667  4.33333333
  5.          5.66666667  6.33333333  7.        ]
```

For a and b above do the follow and print out the results.

1. Square all the elements in both arrays (element-wise).
2. Add both the squared arrays (e.g. [1,2] + [3,4] = [4,6]).
3. Sum the elements with even indices of the added array.
4. Take the square root of the added array (element-wise square root).

In [3]:

```
print("1. ",a*a,b*b)
print("2. ",a**2+b**2)
print("3. ", sum(i for i in a**2+b**2 if i % 2 == 0))
print("4. ", (a**2+b**2)**(0.5))
```

```
1.  [100 121 144 169 196 225 256 289 324 361] [  1.           2.77777778
5.44444444   9.          13.44444444
  18.77777778  25.          32.11111111  40.11111111  49.        ]
2.  [ 101.          123.77777778  149.44444444  178.          209.44444444
  243.77777778  281.          321.11111111  364.11111111  410.        ]
3.  588.0
4.  [ 10.04987562  11.12554618  12.22474721  13.34166406  14.47219556
  15.61338457  16.76305461  17.91957341  19.08169571  20.24845673]
```

Append b to a. Reshape the appended array so that it is a 5x4, 2D-array and store the results in a variable called m. Print m.

In [4]:

```
m = np.reshape(np.append(a,b),(5,4))
print("m: ", m)
```

```
m:  [[ 10.          11.          12.          13.        ]
 [ 14.          15.          16.          17.        ]
 [ 18.          19.           1.           1.66666667]
 [  2.33333333   3.           3.66666667   4.33333333]
 [  5.           5.66666667   6.33333333   7.        ]]
```

Extract the second and the third column of the matrix m. Store the resulting 5x2 matrix in a new variable called m2. Print m2.

In [5]:

```
m2 = m[:,[1,2]]

print("m2: ", m2)
```

```
m2:  [[ 11.          12.        ]
 [ 15.          16.        ]
 [ 19.           1.        ]
 [  3.           3.66666667]
 [  5.66666667   6.33333333]]
```

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 $A \cdot B = A^T B$

In [6]:

```
m3 = np.dot(m2.T,m)

print("m3: ", m3)
```

```
m3:  [[ 697.33333333  748.11111111  437.88888889  482.33333333]
 [ 402.22222222  437.88888889  454.55555556  489.88888889]]
```

Round the `m3` matrix to two decimal points. Store the result in place and print the new `m3`.

In [7]:

```
m3 = np.around(m3, decimals=2)

print("m3: ", m3)
```

```
m3:  [[ 697.33  748.11  437.89  482.33]
 [ 402.22  437.89  454.56  489.89]]
```

Sort the `m3` array so that the highest value is at the top left, the next highest value to the right of the highest, and the lowest value is at the bottom right. Print the sorted `m3` array.

In [8]:

```
sorted_m3 = np.reshape(np.sort(m3.ravel())[::-1],m3.shape)

print("sorted m3: ", sorted_m3)
```

```
sorted m3:  [[ 748.11  697.33  489.89  482.33]
 [ 454.56  437.89  437.89  402.22]]
```

# NumPy and Masks

Create an array called `f` where there are 100 equally-spaced values from 0 to pi, inclusive. Take the sin of the array `f` (element-wise) and store that in place. Print `f`.

In [9]:

```
f = np.sin(np.linspace(0,np.pi,100))

print("f: ", f)
```

```
f:  [  0.00000000e+00   3.17279335e-02   6.34239197e-02   9.50560433e-02
   1.26592454e-01   1.58001396e-01   1.89251244e-01   2.20310533e-01
   2.51147987e-01   2.81732557e-01   3.12033446e-01   3.42020143e-01
   3.71662456e-01   4.00930535e-01   4.29794912e-01   4.58226522e-01
   4.86196736e-01   5.13677392e-01   5.40640817e-01   5.67059864e-01
   5.92907929e-01   6.18158986e-01   6.42787610e-01   6.66769001e-01
   6.90079011e-01   7.12694171e-01   7.34591709e-01   7.55749574e-01
   7.76146464e-01   7.95761841e-01   8.14575952e-01   8.32569855e-01
   8.49725430e-01   8.66025404e-01   8.81453363e-01   8.95993774e-01
   9.09631995e-01   9.22354294e-01   9.34147860e-01   9.45000819e-01
   9.54902241e-01   9.63842159e-01   9.71811568e-01   9.78802446e-01
   9.84807753e-01   9.89821442e-01   9.93838464e-01   9.96854776e-01
   9.98867339e-01   9.99874128e-01   9.99874128e-01   9.98867339e-01
   9.96854776e-01   9.93838464e-01   9.89821442e-01   9.84807753e-01
   9.78802446e-01   9.71811568e-01   9.63842159e-01   9.54902241e-01
   9.45000819e-01   9.34147860e-01   9.22354294e-01   9.09631995e-01
   8.95993774e-01   8.81453363e-01   8.66025404e-01   8.49725430e-01
   8.32569855e-01   8.14575952e-01   7.95761841e-01   7.76146464e-01
   7.55749574e-01   7.34591709e-01   7.12694171e-01   6.90079011e-01
   6.66769001e-01   6.42787610e-01   6.18158986e-01   5.92907929e-01
   5.67059864e-01   5.40640817e-01   5.13677392e-01   4.86196736e-01
   4.58226522e-01   4.29794912e-01   4.00930535e-01   3.71662456e-01
   3.42020143e-01   3.12033446e-01   2.81732557e-01   2.51147987e-01
   2.20310533e-01   1.89251244e-01   1.58001396e-01   1.26592454e-01
   9.50560433e-02   6.34239197e-02   3.17279335e-02   1.22464680e-16]
```

Use a 'mask' and print an array that is True when `f` >= 1/2 and False when `f` < 1/2. Print an array sequence that has only those values where `f` >= 1/2.

In [10]:

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

```
[False False False False False False False False False False False False
 False False False False False  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  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]
[ 0.51367739  0.54064082  0.56705986  0.59290793  0.61815899  0.64278761
  0.666769    0.69007901  0.71269417  0.73459171  0.75574957  0.77614646
  0.79576184  0.81457595  0.83256985  0.84972543  0.8660254   0.88145336
  0.89599377  0.909632    0.92235429  0.93414786  0.94500082  0.95490224
  0.96384216  0.97181157  0.97880245  0.98480775  0.98982144  0.99383846
  0.99685478  0.99886734  0.99987413  0.99987413  0.99886734  0.99685478
  0.99383846  0.98982144  0.98480775  0.97880245  0.97181157  0.96384216
  0.95490224  0.94500082  0.93414786  0.92235429  0.909632    0.89599377
  0.88145336  0.8660254   0.84972543  0.83256985  0.81457595  0.79576184
  0.77614646  0.75574957  0.73459171  0.71269417  0.69007901  0.666769
  0.64278761  0.61815899  0.59290793  0.56705986  0.54064082  0.51367739]
```

# 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).

Run the cell below.

In [11]:

```
# seed the random number generator with a fixed value
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]
```

Find the expected value of x and the expected value of y.

In [12]:

```
print("The expected value of x : ",np.mean(x),"\nThe expected value of y : ",np.mean(y))
```

```
The expected value of x :  5.78253254159
The expected value of y :  11.0129816833
```

Find the variance for x and y.

In [13]:

```
print("The variance for x : ",np.var(x),"\nThe variance for y : ",np.var(y))
```

```
The variance for x :  7.03332752948
The variance for y :  30.1139035755
```

Find the co-variance of x and y.

In [14]:

```
print(np.cov(x,y))
```

```
[[  7.10437124  14.65774383]
 [ 14.65774383  30.41808442]]
```

Assume that the number of dollars spent on car fuel is only linearly dependent on the miles driven. Write code that uses a linear predictor to calculate a predicted value of y for each x.

i.e. $y_{predicted} = f(x) = mx + b$.

In [15]:

```
x_mean=np.mean(x)
y_mean=np.mean(y)
temp_x=[]
temp_y=[]
temp_xy=[]
for i in range(0,len(x)):
    temp_x.append(x[i]-x_mean)
    temp_y.append(y[i]-y_mean)
temp_x=np.array(temp_x)
temp_y=np.array(temp_y)
for i in range(0,len(x)):
    temp_xy.append(temp_x[i]*temp_y[i])
temp_xy=np.array(temp_xy)
m = sum(temp_xy)/(len(x)*np.var(x))
b = y_mean-(x_mean*m)
y_predicted = (m*x) + b
print(y_predicted)
```

```
[  1.86125717   1.39688809   2.20846128   2.28522836   1.98371207
   2.48829527   2.78382468   2.80124813   2.9993232    3.21092152   3.8988
   3.67152796   3.7228942    3.9571493    4.23651436   4.9033035
   4.62116978   4.61656787   5.46829307   5.18342105   5.45873164
   5.79701128   6.12915141   6.11562653   6.31753758   6.81864709
   6.61027849   6.86515115   6.43505522   7.35780389   7.65775187
   7.46087825   7.38719373   7.85455455   7.66325667   7.88892606
   8.00622544   8.33721481   8.95468038   9.08103323   9.33034895
   9.78539799  10.00879629  10.06070164   9.53754157  10.38056671
  10.36512531  10.72999716  10.42269073  11.25028634  10.81276185
  10.97218988  11.35052091  11.83583685  11.38990445  11.51849632
  12.58177632  12.49147206  12.98850691  13.14956122  12.50588416
  13.35028889  13.5506705   13.31658991  14.17094102  13.947246
  13.74018137  14.51931443  14.74126735  14.79877137  14.96739089
  15.28759454  15.66049665  15.1916755   16.05043004  15.57498655
  16.42533161  16.18461169  16.53654675  16.70687695  17.01300263
  16.86428603  17.31062607  16.95616347  17.54476017  18.08227006
  17.57177784  17.49875711  17.92425351  18.80438359  18.91989301
  18.77061069  18.51812677  19.5277969   19.72807224  19.76613158
  19.8657155   19.58014745  19.89856998  20.46773797]
```

Predict y for each value in x, put the error into an array called $y_{error}$.

In [16]:

```
y_error = y-y_predicted
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]
```

Write code that calculates the root mean square error (RMSE).

In [17]:

```
temp=0
for i in range (0,len(y_error)):
    temp=temp+(y_error[i]**2)
rmse = (temp/len(y_error))**(0.5)
print(rmse)
```

```
0.417677723669
```

# Pandas

## Reading a File

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

Data description

- Data source: http://www.fao.org/nr/water/aquastat/data/query/index.html (http://www.fao.org/nr/water/aquastat/data/query/index.html)
- 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

Display the first 10 lines of the dataframe.

In [18]:

```
df = pd.read_csv('data3.csv')
df.head(10)
```

Out[18]:

| | Area | Area Id | Variable Name | Variable Id | Year | Value | Symbol | Md |
|---|---|---|---|---|---|---|---|---|
| **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 |

Display the column names.

In [19]:

```
df.columns
```

Out[19]:

```
Index(['Area', 'Area Id', 'Variable Name', 'Variable Id', 'Year', 'Value',
       'Symbol', 'Md'],
      dtype='object')
```

## Data Preprocessing

Create a mask of NAN values (i.e. apply `.isnull` on the dataframe). Inspect the mask for 'True' values, they denote NANs.
*Hint: You will notice that the last 8 rows and the last column ('Other') have NAN values. You can also use df.tail() to see the last row.*

Remove the bottom 8 rows from the dataframe because they contain NAN values. Also remove the column 'Other'.

In [20]:

```
df.isnull()
k=range(len(df)-8,len(df))
df=df.drop(df.index[k])
df=df.drop('Md', axis=1)
df.head()
```

Out[20]:

| | Area | Area Id | Variable Name | Variable Id | Year | Value | Symbol |
|---|---|---|---|---|---|---|---|
| **0** | Argentina | 9.0 | Total area of the country | 4100.0 | 1962.0 | 278040.0 | E |
| **1** | Argentina | 9.0 | Total area of the country | 4100.0 | 1967.0 | 278040.0 | E |
| **2** | Argentina | 9.0 | Total area of the country | 4100.0 | 1972.0 | 278040.0 | E |
| **3** | Argentina | 9.0 | Total area of the country | 4100.0 | 1977.0 | 278040.0 | E |
| **4** | Argentina | 9.0 | Total area of the country | 4100.0 | 1982.0 | 278040.0 | E |

All the columns in our dataframe are not required for analysis. Drop these columns: `Area Id`, `Variable Id`, and `Symbol` and save the new dataframe as `df1`.

In [21]:

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

Out[21]:

| | 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 |

Display all the unique values in your new dataframe for these columns: `Area`, `Variable Name`, and `Year`.

Note the Countries and the Metrics (ie.recorded variables) represented in your dataset. *Hint: Use .unique( ) method.*

In [22]:

```
print(df.Area.unique(),"\n",df.Year.unique())
print(df["Variable Name"].unique())
```

```
['Argentina' 'Australia' 'Germany' 'Iceland' 'Ireland' 'Sweden'
 'United States of America']
 [ 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.]
['Total area of the country' 'Total population' 'Population density'
 'Gross Domestic Product (GDP)' 'National Rainfall Index (NRI)']
```

Convert the `Year` column string values to pandas datetime objects, where only the year is specified.
*Hint: df1['Year'] = pd.to_datetime(pd.Series(df1['Year']).astype(int),format='%Y').dt.year*

Run df1.tail() to see part of the result.

In [23]:

```
df1['Year'] = pd.to_datetime(pd.Series(df1['Year']).astype(int),format='%Y').dt.year
df1.tail()
```

Out[23]:

| | Area | Variable Name | Year | Value |
|---|---|---|---|---|
| **385** | United States of America | National Rainfall Index (NRI) | 1981 | 949.2 |
| **386** | United States of America | National Rainfall Index (NRI) | 1984 | 974.6 |
| **387** | United States of America | National Rainfall Index (NRI) | 1992 | 1020.0 |
| **388** | United States of America | National Rainfall Index (NRI) | 1996 | 1005.0 |
| **389** | United States of America | National Rainfall Index (NRI) | 2002 | 938.7 |

## Extracting Statistics

Create a dataframe 'dftemp' to store rows where the `Area` is `Iceland`.

In [24]:

```
dftemp=df1[df['Area']=="Iceland"]
dftemp
```

Out[24]:

| | Area | Variable Name | Year | Value |
|---|---|---|---|---|
| **166** | Iceland | Total area of the country | 1962 | 1.030000e+04 |
| **167** | Iceland | Total area of the country | 1967 | 1.030000e+04 |
| **168** | Iceland | Total area of the country | 1972 | 1.030000e+04 |
| **169** | Iceland | Total area of the country | 1977 | 1.030000e+04 |
| **170** | Iceland | Total area of the country | 1982 | 1.030000e+04 |
| **171** | Iceland | Total area of the country | 1987 | 1.030000e+04 |
| **172** | Iceland | Total area of the country | 1992 | 1.030000e+04 |
| **173** | Iceland | Total area of the country | 1997 | 1.030000e+04 |
| **174** | Iceland | Total area of the country | 2002 | 1.030000e+04 |
| **175** | Iceland | Total area of the country | 2007 | 1.030000e+04 |
| **176** | Iceland | Total area of the country | 2012 | 1.030000e+04 |
| **177** | Iceland | Total area of the country | 2014 | 1.030000e+04 |
| **178** | Iceland | Total population | 1962 | 1.826000e+02 |
| **179** | Iceland | Total population | 1967 | 1.974000e+02 |
| **180** | Iceland | Total population | 1972 | 2.099000e+02 |
| **181** | Iceland | Total population | 1977 | 2.221000e+02 |
| **182** | Iceland | Total population | 1982 | 2.331000e+02 |
| **183** | Iceland | Total population | 1987 | 2.469000e+02 |
| **184** | Iceland | Total population | 1992 | 2.599000e+02 |
| **185** | Iceland | Total population | 1997 | 2.728000e+02 |
| **186** | Iceland | Total population | 2002 | 2.869000e+02 |
| **187** | Iceland | Total population | 2007 | 3.054000e+02 |
| **188** | Iceland | Total population | 2012 | 3.234000e+02 |
| **189** | Iceland | Total population | 2015 | 3.294000e+02 |
| **190** | Iceland | Population density | 1962 | 1.773000e+00 |
| **191** | Iceland | Population density | 1967 | 1.917000e+00 |
| **192** | Iceland | Population density | 1972 | 2.038000e+00 |
| **193** | Iceland | Population density | 1977 | 2.156000e+00 |
| **194** | Iceland | Population density | 1982 | 2.263000e+00 |
| **195** | Iceland | Population density | 1987 | 2.397000e+00 |
| **196** | Iceland | Population density | 1992 | 2.523000e+00 |
| **197** | Iceland | Population density | 1997 | 2.649000e+00 |
| **198** | Iceland | Population density | 2002 | 2.785000e+00 |
| **199** | Iceland | Population density | 2007 | 2.965000e+00 |

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

Print the years when the National Rainfall Index (NRI) was `> 950` or `< 900` in Iceland using the dataframe you created in the previous question.

In [25]:

```
dftemp[["Year"]][(df['Variable Name'] == "National Rainfall Index (NRI)") & ((df['Value']>9
```

```
D:\python\lib\site-packages\ipykernel_launcher.py:1: UserWarning: Boolean Se
ries key will be reindexed to match DataFrame index.
  """Entry point for launching an IPython kernel.
```

Out[25]:

| | Year |
|---|---|
| **214** | 1967 |
| **215** | 1971 |
| **216** | 1975 |
| **218** | 1986 |
| **219** | 1991 |
| **220** | 1997 |

Get all the rows of df1 (from the preprocessed data section of this notebook) where the `Area` is `United States of America` and store that into a new dataframe called `df_usa`. Set the indices of the this dataframe to be the `Year` column.
*Hint: Use .set_index()*

In [26]:

```
df_usa = df1[df1["Area"]=="United States of America"]
df_usa.set_index("Year", inplace = True)
df_usa.head()
```

Out[26]:

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

Pivot the dataframe so that the unique `Variable Name` entries become the column entries. The dataframe values should be the ones in the `Value` column. Do this by running the lines of code below.

In [27]:

```
df_usa=df_usa.pivot(columns='Variable Name',values='Value')
df_usa.head()
```

Out[27]:

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

Rename the corresponding columns to ['GDP','NRI','PD','Area','Population'].

In [28]:

```
df_usa.rename(columns={'Gross Domestic Product (GDP)': 'GDP', 'National Rainfall Index (NRI
df_usa.head()
```

Out[28]:

| Variable Name<br>Year | GDP | NRI | PD | Area | Population |
|---|---|---|---|---|---|
| **1962** | 6.050000e+11 | NaN | 19.93 | 962909.0 | 191861.0 |
| **1965** | NaN | 928.5 | NaN | NaN | NaN |
| **1967** | 8.620000e+11 | NaN | 21.16 | 962909.0 | 203713.0 |
| **1969** | NaN | 952.2 | NaN | NaN | NaN |
| **1972** | 1.280000e+12 | NaN | 22.14 | 962909.0 | 213220.0 |

Print the output of `df_usa.isnull().sum()`. This gives us the number of NAN values in each column. Replace the NAN values by 0, using `df_usa=df_usa.fillna(0)`. Print the output of `df_usa.isnull().sum()` again.

In [29]:

```
print("Number of NAN values before: ", df_usa.isnull().sum())
df_usa=df_usa.fillna(0)
print("Number of NAN values after: ", df_usa.isnull().sum())
```

```
Number of NAN values before:  Variable Name
GDP            7
NRI           11
PD             7
Area           7
Population     7
dtype: int64
Number of NAN values after:  Variable Name
GDP           0
NRI           0
PD            0
Area          0
Population    0
dtype: int64
```

Calculate and print all the column averages and the column standard deviations.

In [30]:

```
print(np.mean(df_usa),"\n",np.std(df_usa))
```

```
Variable Name
GDP           4.620895e+12
NRI           4.092737e+02
PD            1.670158e+01
Area          6.103147e+05
Population    1.615134e+05
dtype: float64
 Variable Name
GDP           5.926262e+12
NRI           4.803878e+02
PD            1.319310e+01
Area          4.661739e+05
Population    1.278764e+05
dtype: float64
```

Using the `df_usa` dataframe, multiply the `Area` by 10 (so instead of 1000 ha, the unit becomes 100 ha = 1km^2). Store the result in place.

In [31]:

```
df_usa["Area"] = df_usa["Area"] * 10
df_usa.head()
```

Out[31]:

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

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. Store the result in place.

In [32]:

```
df_usa["GDP/capita"]= round(df_usa["GDP"]/df_usa["Population"],2)
df_usa.head()
```

Out[32]:

| Variable Name | GDP | NRI | PD | Area | Population | GDP/capita |
|---|---|---|---|---|---|---|
| **Year** | | | | | | |
| **1962** | 6.050000e+11 | 0.0 | 19.93 | 9629090.0 | 191861.0 | 3153324.54 |
| **1965** | 0.000000e+00 | 928.5 | 0.00 | 0.0 | 0.0 | NaN |
| **1967** | 8.620000e+11 | 0.0 | 21.16 | 9629090.0 | 203713.0 | 4231443.26 |
| **1969** | 0.000000e+00 | 952.2 | 0.00 | 0.0 | 0.0 | NaN |
| **1972** | 1.280000e+12 | 0.0 | 22.14 | 9629090.0 | 213220.0 | 6003189.19 |

Create a new column in `df_usa` called `PD2` (i.e. population density 2). Calculate the population density. **Note: the units should be inhab/km^2**. Round the reults to two decimal point. Store the result in place.

In [33]:

```
df_usa["PD2"] = round(df_usa["Population"]/df_usa["Area"],2)
df_usa.head()
```

Out[33]:

| Variable Name | GDP | NRI | PD | Area | Population | GDP/capita | PD2 |
|---|---|---|---|---|---|---|---|
| **Year** | | | | | | | |
| **1962** | 6.050000e+11 | 0.0 | 19.93 | 9629090.0 | 191861.0 | 3153324.54 | 0.02 |
| **1965** | 0.000000e+00 | 928.5 | 0.00 | 0.0 | 0.0 | NaN | NaN |
| **1967** | 8.620000e+11 | 0.0 | 21.16 | 9629090.0 | 203713.0 | 4231443.26 | 0.02 |
| **1969** | 0.000000e+00 | 952.2 | 0.00 | 0.0 | 0.0 | NaN | NaN |
| **1972** | 1.280000e+12 | 0.0 | 22.14 | 9629090.0 | 213220.0 | 6003189.19 | 0.02 |

Find the maximum value and minimum value of the 'NRI' column in the USA (using pandas methods). What years do the min and max values occur in?

In [34]:

```
print("Max NRI were ",df_usa['NRI'].max(), " in the year ",df_usa['NRI'].idxmax())
print("\nMin NRI were ",df_usa['NRI'][df_usa['NRI']>0].min(), " in the year ",df_usa['NRI']
```

```
Max NRI were  1020.0  in the year  1992

Min NRI were  928.5  in the year  1965
```

# Matplotlib

Create a dataframe called `icecream` that has column `Flavor` with entries `Strawberry`, `Vanilla`, and `Chocolate` and another column with `Price` with entries `3.50`, `3.00`, and `4.25`.

In [35]:

```
icecream = pd.DataFrame.from_items([('Flavor', ['Strawberry', 'Vanilla', 'Chocolate']), ('P
icecream
```

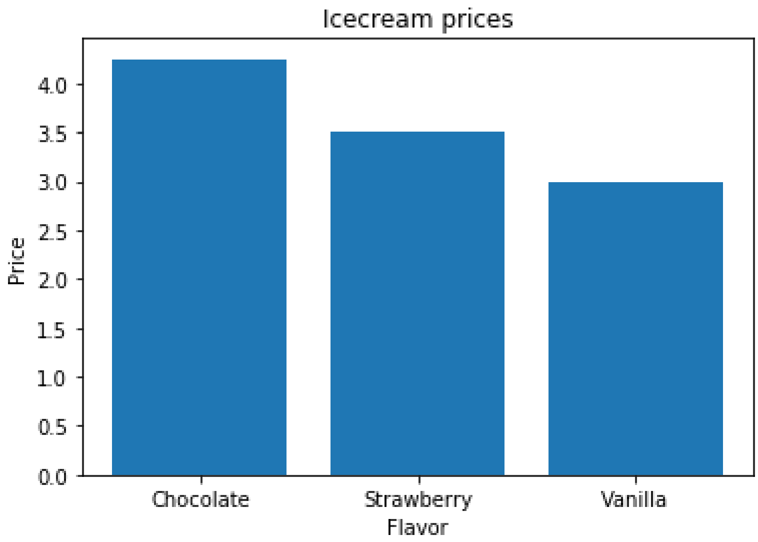
Out[35]:

| | Flavor | Price |
|---|---|---|
| **0** | Strawberry | 3.50 |
| **1** | Vanilla | 3.00 |
| **2** | Chocolate | 4.25 |

Create a bar chart representing the three flavors and their associated prices.

In [36]:

```
plt.bar(icecream['Flavor'], icecream['Price'])
plt.xlabel('Flavor')
plt.ylabel('Price')
plt.title('Icecream prices')
plt.show()
```



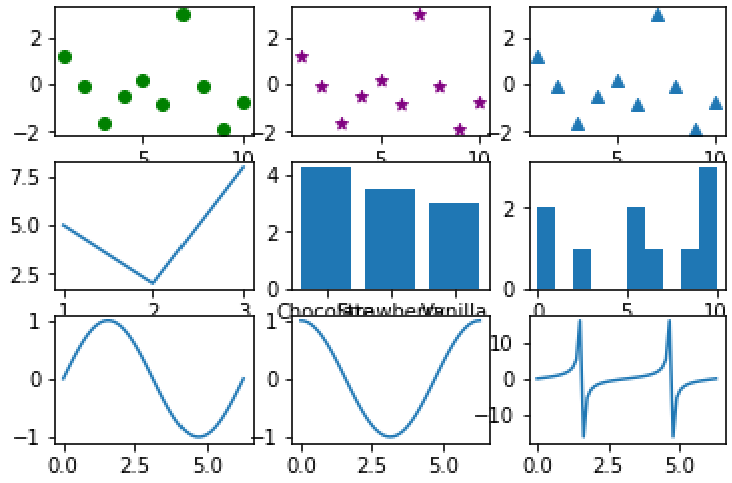

Create 9 random plots. The top three should be scatter plots (one with green dots, one with purple crosses, and one with blue triangles. The middle three graphs should be a line graph, a horizontal bar chart, and a histogram. The bottom three graphs should be trignometric functions (one sin, one cosine, one tangent).

In [37]:

```
N = 10
x = np.arange(1,N+1)
y = np.random.randn(N)
vals = np.random.randint(0,11,N)

plt.subplot2grid((3,3), (0,0)).scatter(x,y,color='green')
plt.subplot2grid((3,3), (0,1)).scatter(x,y,marker='*',color='purple')
plt.subplot2grid((3,3), (0,2)).scatter(x,y,marker='^')
plt.subplot2grid((3,3), (1,0)).plot([1,2,3],[5,2,8])
plt.subplot2grid((3,3), (1,1)).bar(icecream['Flavor'], icecream['Price'])
plt.subplot2grid((3,3), (1,2)).hist(vals)

x = np.linspace(0,2*np.pi,51)
y1 = np.sin(x)

plt.subplot2grid((3,3), (2,0)).plot(x,y1)

y1 = np.cos(x)

plt.subplot2grid((3,3), (2,1)).plot(x,y1)

y1 = np.tan(x)

plt.subplot2grid((3,3), (2,2)).plot(x,y1)
```

Out[37]:

```
[<matplotlib.lines.Line2D at 0x19618ca9240>]
```



## Extra Credit

Run the cell below to read in the data. See: [https://www.quantshare.com/sa-43-10-ways-to-download-historical-stock-quotes-data-for-free (https://www.quantshare.com/sa-43-10-ways-to-download-historical-stock-quotes-data-for-free)](https://www.quantshare.com/sa-43-10-ways-to-download-historical-stock-quotes-data-for-free)

In [38]:

```
df_google = pd.read_csv('https://finance.google.com/finance/historical?output=csv&q=goog')
df_apple = pd.read_csv('https://finance.google.com/finance/historical?output=csv&q=aapl')

df_disney = pd.read_csv('https://finance.google.com/finance/historical?output=csv&q=dis')
df_nike= pd.read_csv('https://finance.google.com/finance/historical?output=csv&q=nke')

df_google.head()
```

Out[38]:

| | Date | Open | High | Low | Close | Volume |
|---|---|---|---|---|---|---|
| **0** | 8-Feb-18 | 1055.41 | 1058.62 | 1000.66 | 1001.52 | 2859136 |
| **1** | 7-Feb-18 | 1081.54 | 1081.78 | 1048.26 | 1048.58 | 2369232 |
| **2** | 6-Feb-18 | 1027.18 | 1081.71 | 1023.14 | 1080.60 | 3447956 |
| **3** | 5-Feb-18 | 1090.60 | 1110.00 | 1052.03 | 1055.80 | 3798301 |
| **4** | 2-Feb-18 | 1122.00 | 1123.07 | 1107.28 | 1111.90 | 4857943 |

Show a 3 x 3 correlation matrix for Nike, Apple, and Disney stock prices for the month of July, 2017.

Hint: Convert `Date` to a pandas datetime object. Change the indices of all the dataframes to `Date`. Use `Date` indices to filter rows. Create a new dataframe that stores values of the `Close` column from each dataframe. Use the `Close` column of each company's stock data to find the correlation using df.corr().

In [39]:

```
df_google['Date'] = pd.to_datetime(df_google['Date'])
df_disney['Date'] = pd.to_datetime(df_disney['Date'])
df_nike['Date'] = pd.to_datetime(df_nike['Date'])
google = df_google[(df_google["Date"]>pd.to_datetime("2017-06-30")) & (df_google["Date"]<pd
disney = df_disney[(df_disney["Date"]>pd.to_datetime("2017-06-30")) & (df_disney["Date"]<pd
nike = df_nike[(df_nike["Date"]>pd.to_datetime("2017-06-30")) & (df_nike["Date"]<pd.to_date
s = {"Google" : google["Close"], "Disney" : disney["Close"], "Nike" : nike["Close"]}
stock = pd.DataFrame(data=s)
```

In [40]:

```
stock.corr()
```

Out[40]:

| | Disney | Google | Nike |
|---|---|---|---|
| **Disney** | 1.000000 | 0.188795 | 0.459045 |
| **Google** | 0.188795 | 1.000000 | 0.351716 |
| **Nike** | 0.459045 | 0.351716 | 1.000000 |

Show the same correlation matrix but over different time periods.

1. the last 20 days
2. the last 80 days

In [41]:

```
from datetime import datetime, timedelta
N = 21
date_N_days_ago = datetime.now() - timedelta(days=N)
google = df_google[(df_google["Date"]>date_N_days_ago)]
disney = df_disney[(df_disney["Date"]>date_N_days_ago)]
nike = df_nike[(df_nike["Date"]>date_N_days_ago)]
s = {"Google" : google["Close"], "Disney" : disney["Close"], "Nike" : nike["Close"]}
stock = pd.DataFrame(data=s)
stock.corr()
```

Out[41]:

| | Disney | Google | Nike |
|---|---|---|---|
| **Disney** | 1.000000 | 0.966388 | 0.900179 |
| **Google** | 0.966388 | 1.000000 | 0.929205 |
| **Nike** | 0.900179 | 0.929205 | 1.000000 |

In [42]:

```
N = 81
date_N_days_ago = datetime.now() - timedelta(days=N)
google = df_google[(df_google["Date"]>date_N_days_ago)]
disney = df_disney[(df_disney["Date"]>date_N_days_ago)]
nike = df_nike[(df_nike["Date"]>date_N_days_ago)]
s = {"Google" : google["Close"], "Disney" : disney["Close"], "Nike" : nike["Close"]}
stock = pd.DataFrame(data=s)
stock.corr()
```

Out[42]:

| | Disney | Google | Nike |
|---|---|---|---|
| **Disney** | 1.000000 | 0.650967 | 0.659349 |
| **Google** | 0.650967 | 1.000000 | 0.846589 |
| **Nike** | 0.659349 | 0.846589 | 1.000000 |

Change the code so that it accepts a list of any stock symbols (i.e. ['NKE', 'APPL', 'DIS', ... ]) and creates a correlation matrix for the past 100 days.

In [43]:

```
def Stock_corelation( list_stock_real ):
    list_stock = list_stock_real[:]
    N = 101
    date_N_days_ago = datetime.now() - timedelta(days=N)
    stock_dict = {}
    for l in range(0,len(list_stock)):
        list_stock[l] = pd.read_csv("https://finance.google.com/finance/historical?output=c
        list_stock[l]["Date"] = pd.to_datetime(list_stock[l]["Date"])
        list_stock[l] = list_stock[l][(list_stock[l]['Date']>date_N_days_ago)]
        stock_dict.update({list_stock_real[l][:]:list_stock[l]["Close"]})
    stock = pd.DataFrame(data=stock_dict)
    print(stock.corr())
Stock_corelation(["goog","aapl","dis","nke"])
Stock_corelation(["goog","aapl"])
```

```
          aapl       dis      goog       nke
aapl  1.000000  0.290307  0.203129 -0.076929
dis   0.290307  1.000000  0.692885  0.818333
goog  0.203129  0.692885  1.000000  0.791691
nke  -0.076929  0.818333  0.791691  1.000000
          aapl      goog
aapl  1.000000  0.203129
goog  0.203129  1.000000
```

In [ ]: