Homework 03

NAME: Jieun Hwang

STUDENT ID: 3033297165

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:

i) Cube (i.e. raise to the power of 3) all the elements in both arrays (element-wise)
ii) Add both the cubed arrays (e.g., [1,2] + [3,4] = [4,6])
iii) Sum the elements with even indices of the added array.
iv) Take the square root of the added array (element-wise square root)__

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

a = np.array(range(25,35))
b = np.linspace(1,6,10)

s = a**3 + b**3
s
```

Out[1]:
```
array([15626.        , 17579.76406036, 19692.40877915, 21970.96296296,
       24422.45541838, 27053.91495199, 29872.37037037, 32884.85048011,
       36098.38408779, 39520.        ])
```

In [2]:
```
print (sum(s[0::2]))
```

```
125711.61865569273
```

In [3]:
```
s**(1/2)
```

Out[3]:
```
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 [4]:
```
m = np.concatenate((a,b)).reshape(4,5)
print(m)
#m.ndim
```

```
[[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 [5]:
```
m2 = m[:, [2, 3]]
print(m2)
```

```
[[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 [6]:
```
m3 = np.dot(np.transpose(m2),m)
print(m3)
```

```
[[1655.58024691 1718.4691358  1781.35802469 1844.24691358 1907.13580
247]
 [1713.2345679  1778.74074074 1844.24691358 1909.75308642 1975.25925
926]]
```

1e) Round the m3 matrix to three decimal points. Store the result in place and print the new m3.

In [7]:
```
m3 = np.round(m3,3)
print(m3)
```

```
[[1655.58  1718.469 1781.358 1844.247 1907.136]
 [1713.235 1778.741 1844.247 1909.753 1975.259]]
```

1f) Sort the m3 array so that the highest value is at the bottom right and the lowest value is at the top left. Print the sorted m3 array.

In [8]:
```
np.sort(m3.ravel()).reshape(2,5)
```

Out[8]:
```
array([[1655.58 , 1713.235, 1718.469, 1778.741, 1781.358],
       [1844.247, 1844.247, 1907.136, 1909.753, 1975.259]])
```

NumPy and Masks

2a) create an 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 a 'mask' 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 [9]:
```
from numpy import pi

x = np.linspace(0,pi,50)
f = np.cos(x)
print(f)
```

```
[ 1.          0.99794539  0.99179001  0.98155916  0.96729486  0.9490
5575
  0.92691676  0.90096887  0.8713187   0.8380881   0.80141362  0.7614
4596
  0.71834935  0.67230089  0.6234898   0.57211666  0.51839257  0.4625
3829
  0.40478334  0.34536505  0.28452759  0.22252093  0.1595999   0.0960
2303
  0.03205158 -0.03205158 -0.09602303 -0.1595999  -0.22252093 -0.2845
2759
 -0.34536505 -0.40478334 -0.46253829 -0.51839257 -0.57211666 -0.6234
898
 -0.67230089 -0.71834935 -0.76144596 -0.80141362 -0.8380881  -0.8713
187
 -0.90096887 -0.92691676 -0.94905575 -0.96729486 -0.98155916 -0.9917
9001
 -0.99794539 -1.        ]
```

In [10]:
```
mask = f>=(1/2)
print(mask)
```

```
[ True  True  True  True  True  True  True  True  True  True  True
True
  True  True  True  True  True False False False False False False F
alse
 False False False False False False False False False False False F
alse
 False False False False False False False False False False False F
alse
 False False]
```

In [11]:
```
print(f[mask])
```

```
[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 [12]:
```
# 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   2.2242
4872
  2.24914511  2.36268477  2.49808849  2.8212704   2.68452475  2.6822
9427
  3.09511169  2.95703884  3.09047742  3.2544361   3.41541904  3.4088
6375
  3.50672677  3.74960644  3.64861355  3.7721462   3.56368566  4.0109
2701
  4.15630694  4.06088549  4.02517179  4.25169402  4.15897504  4.2683
5333
  4.32520644  4.48563164  4.78490721  4.84614839  4.96698768  5.1875
4259
  5.29582013  5.32097781  5.0674106   5.47601124  5.46852704  5.6453
7452
  5.49642807  5.89755027  5.68548923  5.76276141  5.94613234  6.1813
5713
  5.96522091  6.0275473   6.54290191  6.4991329   6.74003765  6.8180
9807
  6.50611821  6.91538752  7.01250925  6.89905417  7.31314433  7.2047
2297
  7.1043621   7.48199528  7.58957227  7.61744354  7.6991707   7.8543
6822
  8.03510784  7.80787781  8.22410224  7.99366248  8.40581097  8.2891
3792
  8.45971515  8.54227144  8.6906456   8.61856507  8.83489887  8.6630
9658
  8.94837987  9.20890222  8.9614749   8.92608294  9.13231416  9.5588
9896
  9.61488451  9.54252979  9.42015491  9.90952569 10.00659591 10.0250
4265
```

```
 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  3.3937
7767
  3.78503343  4.38293049  4.32963586  4.03925039  4.73691868  4.3009
8399
  4.8416329   4.78175957  4.99765787  5.31746817  5.76844671  5.9372
3749
  5.72811642  6.70973615  6.68143367  6.57482731  7.17737603  7.5486
3252
  7.30221419  7.3202573   7.78023884  7.91133365  8.2765417   8.6920
3281
  8.78219865  8.45897546  8.89094715  8.81719921  8.87106971  9.6619
2562
  9.4020625   9.85990783  9.60359778 10.07386266 10.6957995  10.6672
1916
 11.18256285 10.57431836 11.46744716 10.94398916 11.26445259 12.0975
4828
 12.11988037 12.121557   12.17613693 12.43750193 13.00912372 12.8640
7194
 13.24640866 12.76120085 13.11723062 14.07841099 14.19821707 14.2728
9001
 14.30624942 14.63060835 14.2770918  15.0744923  14.45261619 15.1189
7313
 15.2378667  15.27203124 15.32491892 16.01095271 15.71250558 16.2948
8506
 16.70618934 16.56555394 16.42379457 17.18144744 17.13813976 17.6961
3625
 17.37763019 17.90942839 17.90343733 18.01951169 18.35727914 18.1684
1269
 18.61813748 18.66062754 18.81217983 19.44995194 19.7213867  19.7196
6726
 19.78961904 19.64385088 20.69719809 20.07974319]
```

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

In [13]:
```
expect_x = np.mean(x)
expect_y = np.mean(y)

print(expect_x, expect_y)
```

```
5.782532541587923 11.012981683344968
```

3b) Find variance of distributions of x and y

In [14]:
```
np.var(x)
```

Out[14]: 7.03332752947585

In [15]:
```
np.var(y)
```

Out[15]: 30.113903575509635

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

In [16]:
```
np.cov(x,y)[0][1]
```

Out[16]: 14.657743832803437

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 ie y_predicted = f(x) = y0+mx.

In [17]:
```
m = np.cov(x,y)[0][1]/np.var(x) # slope
y_intercept = expect_y - np.cov(x,y)[0][1]*expect_x/np.var(x)
print(m,y_intercept)
```

```
2.0840411272437627 -1.0380539529496655
```

In [18]:
```
y_predicted = m*x + y_intercept
```

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

In [19]:
```
y_predicted
```

Out[19]:
```
array([ 1.76881551,  1.29975583,  2.11952674,  2.19706924,  1.892507
33,
        2.40218732,  2.70070189,  2.71830133,  2.91837716,  3.132112
83,
        3.82693958,  3.59737186,  3.64925696,  3.88587827,  4.168065
19,
        4.84158958,  4.55660602,  4.55195763,  5.41228611,  5.124536
6 ,
        5.4026281 ,  5.74432472,  6.07981979,  6.06615829,  6.270108
85,
        6.77628008,  6.56580674,  6.82325387,  6.38881354,  7.320882
9 ,
        7.62386066,  7.42499842,  7.35056961,  7.82265125,  7.629421
07,
        7.85736994,  7.97585416,  8.31018686,  8.93388945,  9.061518
6 ,
        9.31335266,  9.77299816,  9.998653  , 10.05108265,  9.522638
14,
       10.37417868, 10.3585813 , 10.72713873, 10.41672819, 11.252683
36,
       10.81073943, 10.97177784, 11.35393039, 11.84414852, 11.393711
75,
       11.52360252, 12.59762273, 12.50640631, 13.00846171, 13.171142
83,
       12.52096398, 13.37389806, 13.57630372, 13.33985868, 14.202839
6 ,
       13.97688504, 13.76772884, 14.55473193, 14.7789268 , 14.837011
67,
       15.00733442, 15.33077245, 15.70744125, 15.23388452, 16.101313
36,
       15.6210674 , 16.48000181, 16.23685037, 16.59234034, 16.764391
04,
       17.07360891, 16.92339011, 17.37423864, 17.01619561, 17.610737
73,
       18.15367702, 17.63802831, 17.56427   , 17.99406434, 18.883084
62,
       18.9997608 , 18.84897058, 18.59393631, 19.61380514, 19.816103
46,
       19.85454723, 19.95513706, 19.66668447, 19.98832339, 20.563240
56])
```

In [20]:
```
y_error = abs(y-y_predicted)
y_error
```

Out[20]:
```
array([1.05314309e-01, 7.21703366e-01, 1.13662110e-02, 6.30957167e-0
2,
       7.03682516e-02, 5.53276173e-01, 3.28116980e-01, 6.17351628e-0
1,
       1.63719369e-01, 2.92897867e-01, 4.30238098e-01, 2.03594191e-0
```

```
1,
       1.35776473e-01, 4.97052216e-01, 1.61570667e-01, 8.02339188e-0
1,
       1.80312658e-01, 2.50973643e-01, 5.70653214e-01, 3.42777034e-0
1,
       4.04970232e-01, 4.26856544e-01, 3.11373081e-01, 1.28920801e-0
1,
       5.41992427e-01, 6.65439261e-02, 1.15626927e-01, 2.48426563e-0
1,
       7.88562495e-01, 2.27749619e-01, 3.21646464e-01, 1.04741121e-0
1,
       4.29669229e-01, 8.86824064e-02, 6.47120628e-01, 8.34662873e-0
1,
       8.06344483e-01, 1.48788597e-01, 4.29423037e-02, 2.44319388e-0
1,
       4.42282956e-01, 1.11072540e-01, 5.96590507e-01, 1.91174825e-0
1,
       8.09596391e-02, 3.00316025e-01, 3.37218195e-01, 5.99195710e-0
2,
       7.65834652e-01, 6.78364995e-01, 6.56707732e-01, 2.77886742e-0
2,
       8.94778085e-02, 2.53399759e-01, 7.26168620e-01, 5.97954474e-0
1,
       4.21485798e-01, 6.89043727e-02, 6.62014456e-04, 3.07070891e-0
1,
       7.25444679e-01, 6.12697206e-01, 4.59073099e-01, 7.38552302e-0
1,
       4.62253673e-03, 2.96004972e-01, 5.38520575e-01, 7.58764193e-0
2,
       5.01835004e-01, 2.37480633e-01, 5.54718230e-01, 2.11799323e-0
1,
       4.69574546e-01, 3.81467141e-02, 7.76394438e-01, 3.89885301e-0
1,
       7.67496225e-01, 5.80346879e-02, 1.13849001e-01, 1.98837099e-0
1,
       6.49814339e-01, 2.58057329e-01, 2.36098875e-01, 6.79940645e-0
1,
       2.33107534e-01, 2.44248626e-01, 2.65409018e-01, 4.55241691e-0
1,
       3.63214807e-01, 7.14671927e-01, 3.81623320e-01, 1.88343033e-0
1,
       2.18243517e-01, 1.63853198e-01, 9.47167582e-02, 1.34879969e-0
1,
       1.65518016e-01, 2.28335961e-02, 7.08874698e-01, 4.83497362e-0
1])
```

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

In [21]:
```
RMSE = np.mean(np.square(y_predicted-y))
RMSE
```

Out[21]: 0.1775094149469674