Homework 03

NAME: Shubei Wang

STUDENT ID: 3034358656

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)
c = a+b

print('i)', a**3, b**3)
print('ii)', c)
print('iii)', c[::2].sum())
print('iv)', np.sqrt(c))
```

```
i) [15625 17576 19683 21952 24389 27000 29791 32768 35937 39304] [  1.
           3.76406036   9.40877915  18.96296296  33.45541838
  53.91495199  81.37037037 116.85048011 161.38408779 216.        ]
ii) [26.         27.55555556 29.11111111 30.66666667 32.22222222 33.77
777778
 35.33333333 36.88888889 38.44444444 40.        ]
iii) 161.11111111111114
iv) [5.09901951 5.24933858 5.39547135 5.53774924 5.67646212 5.81186526
 5.94418483 6.07362239 6.20035841 6.32455532]
```

1b) Append b to a, reshape the appended array so that it is a 4x5, 2d array and store the results in a variable called m. Print m.

In [2]:

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

```
[[25.         26.         27.         28.         29.        ]
 [30.         31.         32.         33.         34.        ]
 [ 1.          1.55555556  2.11111111  2.66666667  3.22222222]
 [ 3.77777778  4.33333333  4.88888889  5.44444444  6.        ]]
```

1c) Extract the third and the fourth column of the m matrix. Store the resulting 4x2 matrix in a new variable called m2. Print m2.

In [3]:

```
m2 = m[:,2:4]
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 [4]:

```
m3 = m2.T.dot(m)
print(m3)
```

```
[[1655.58024691 1718.4691358  1781.35802469 1844.24691358 1907.1358024
7]
 [1713.2345679  1778.74074074 1844.24691358 1909.75308642 1975.2592592
6]]
```

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

In [5]:

```
m3 = np.around(m3, decimals = 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 [6]:

```
m3.ravel().sort()
m3 = m3.reshape(2,5)
print(m3)
```

```
[[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 [7]:

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

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

f2 = f[mask]
print(f2)
```

```
[ 1.          0.99794539  0.99179001  0.98155916  0.96729486  0.949055
75
  0.92691676  0.90096887  0.8713187   0.8380881   0.80141362  0.761445
96
  0.71834935  0.67230089  0.6234898   0.57211666  0.51839257  0.462538
29
  0.40478334  0.34536505  0.28452759  0.22252093  0.1595999   0.096023
03
  0.03205158 -0.03205158 -0.09602303 -0.1595999  -0.22252093 -0.284527
59
 -0.34536505 -0.40478334 -0.46253829 -0.51839257 -0.57211666 -0.623489
8
 -0.67230089 -0.71834935 -0.76144596 -0.80141362 -0.8380881  -0.871318
7
 -0.90096887 -0.92691676 -0.94905575 -0.96729486 -0.98155916 -0.991790
01
 -0.99794539 -1.        ]
[ True  True  True  True  True  True  True  True  True  True  True  Tr
ue
  True  True  True  True  True False False False False False False Fal
se
 False False False False False False False False False False False Fal
se
 False False False False False False False False False False False Fal
se
 False False]
[1.         0.99794539 0.99179001 0.98155916 0.96729486 0.94905575
 0.92691676 0.90096887 0.8713187  0.8380881  0.80141362 0.76144596
 0.71834935 0.67230089 0.6234898  0.57211666 0.51839257]
```

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 [8]:

```
# 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  1.6
5075498
  1.79399331  1.80243817  1.89844195  2.00100023  2.3344038   2.224248
72
  2.24914511  2.36268477  2.49808849  2.8212704   2.68452475  2.682294
27
  3.09511169  2.95703884  3.09047742  3.2544361   3.41541904  3.408863
75
  3.50672677  3.74960644  3.64861355  3.7721462   3.56368566  4.010927
01
  4.15630694  4.06088549  4.02517179  4.25169402  4.15897504  4.268353
33
  4.32520644  4.48563164  4.78490721  4.84614839  4.96698768  5.187542
59
  5.29582013  5.32097781  5.0674106   5.47601124  5.46852704  5.645374
52
  5.49642807  5.89755027  5.68548923  5.76276141  5.94613234  6.181357
13
  5.96522091  6.0275473   6.54290191  6.4991329   6.74003765  6.818098
07
  6.50611821  6.91538752  7.01250925  6.89905417  7.31314433  7.204722
97
  7.1043621   7.48199528  7.58957227  7.61744354  7.6991707   7.854368
22
  8.03510784  7.80787781  8.22410224  7.99366248  8.40581097  8.289137
92
  8.45971515  8.54227144  8.6906456   8.61856507  8.83489887  8.663096
58
  8.94837987  9.20890222  8.9614749   8.92608294  9.13231416  9.558898
96
  9.61488451  9.54252979  9.42015491  9.90952569 10.00659591 10.025042
65
 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  3.393777
67
  3.78503343  4.38293049  4.32963586  4.03925039  4.73691868  4.300983
99
  4.8416329   4.78175957  4.99765787  5.31746817  5.76844671  5.937237
49
  5.72811642  6.70973615  6.68143367  6.57482731  7.17737603  7.548632
52
  7.30221419  7.3202573   7.78023884  7.91133365  8.2765417   8.692032
81
  8.78219865  8.45897546  8.89094715  8.81719921  8.87106971  9.661925
62
  9.4020625   9.85990783  9.60359778 10.07386266 10.6957995  10.667219
16
 11.18256285 10.57431836 11.46744716 10.94398916 11.26445259 12.097548
```

```
28
 12.11988037 12.121557   12.17613693 12.43750193 13.00912372 12.864071
94
 13.24640866 12.76120085 13.11723062 14.07841099 14.19821707 14.272890
01
 14.30624942 14.63060835 14.2770918  15.0744923  14.45261619 15.118973
13
 15.2378667  15.27203124 15.32491892 16.01095271 15.71250558 16.294885
06
 16.70618934 16.56555394 16.42379457 17.18144744 17.13813976 17.696136
25
 17.37763019 17.90942839 17.90343733 18.01951169 18.35727914 18.168412
69
 18.61813748 18.66062754 18.81217983 19.44995194 19.7213867  19.719667
26
 19.78961904 19.64385088 20.69719809 20.07974319]
```

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

In [9]:

```
Ex = x.sum()/100
Ey = y.sum()/100

print('Ex = ',Ex)
print('Ey = ',Ey)
```

```
Ex =  5.782532541587923
Ey =  11.012981683344968
```

3b) Find variance of distributions of x and y

In [10]:

```
Vx = (x**2).sum()/100 - Ex**2
print(Vx)
```

```
7.033327529475862
```

In [11]:

```
Vy = (y**2).sum()/100 - Ey**2
print(Vy)
```

```
30.113903575509667
```

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

In [12]:

```
Cov = (x*y).sum()/100 - Ex*Ey
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 ie y_predicted = f(x)

= y0+mx.

In [13]:

```
m = Cov/Vx
y0 = Ey-m*Ex
```

In [14]:

```
def predict(x):
    y = y0+m*x
    return y
```

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

In [15]:

```
y_p = predict(x)
print(y_p)
```

```
[ 1.86125717  1.39688809  2.20846128  2.28522836  1.98371207  2.488295
27
  2.78382468  2.80124813  2.9993232   3.21092152  3.8988      3.671527
96
  3.7228942   3.9571493   4.23651436  4.9033035   4.62116978  4.616567
87
  5.46829307  5.18342105  5.45873164  5.79701128  6.12915141  6.115626
53
  6.31753758  6.81864709  6.61027849  6.86515115  6.43505522  7.357803
89
  7.65775187  7.46087825  7.38719373  7.85455455  7.66325667  7.888926
06
  8.00622544  8.33721481  8.95468038  9.08103323  9.33034895  9.785397
99
 10.00879629 10.06070164  9.53754157 10.38056671 10.36512531 10.729997
16
 10.42269073 11.25028634 10.81276185 10.97218988 11.35052091 11.835836
85
 11.38990445 11.51849632 12.58177632 12.49147206 12.98850691 13.149561
22
 12.50588416 13.35028889 13.5506705  13.31658991 14.17094102 13.947246
 13.74018137 14.51931443 14.74126735 14.79877137 14.96739089 15.287594
54
 15.66049665 15.1916755  16.05043004 15.57498655 16.42533161 16.184611
69
 16.53654675 16.70687695 17.01300263 16.86428603 17.31062607 16.956163
47
 17.54476017 18.08227006 17.57177784 17.49875711 17.92425351 18.804383
59
 18.91989301 18.77061069 18.51812677 19.5277969  19.72807224 19.766131
58
 19.8657155  19.58014745 19.89856998 20.46773797]
```

In [16]:

```
y_error = y-y_p
print(y_error)
```

```
[-0.19775597  0.62457111 -0.10030076 -0.02506341 -0.02083649  0.467168
23
  0.24499418  0.53440482 -0.24466541  0.21408918 -0.50209852 -0.277750
29
  0.06213923  0.42578118  0.0931215  -0.86405311  0.1157489  -0.315583
88
 -0.62666017 -0.40166149 -0.46107377 -0.47954311 -0.3607047  -0.178389
04
 -0.58942116 -0.10891094  0.07115518 -0.29032384  0.74232081  0.190828
63
 -0.35553767 -0.14062095  0.39304511  0.0567791   0.61328502  0.803106
76
  0.77597321  0.12176065 -0.06373323 -0.26383402 -0.45927925 -0.123472
38
 -0.60673379 -0.20079382  0.0660562  -0.30670405  0.33067419 -0.062778
  0.75987212 -0.67596798  0.65468531 -0.02820071 -0.08606832  0.261711
43
  0.72997592  0.60306068 -0.40563939 -0.05397013  0.02061681 -0.285489
28
  0.7405245  -0.58908804 -0.43343988  0.76182107  0.02727604  0.325644
01
  0.56606805  0.11129392 -0.46417555  0.27572093 -0.5147747  -0.168621
42
 -0.42262995  0.08035574 -0.72551112  0.43596616 -0.71282602  0.110273
37
  0.16964259 -0.14132301 -0.58920807  0.31716141 -0.17248631  0.739972
78
 -0.16712997 -0.17284167  0.33165948  0.52075457  0.43302563 -0.635970
9
 -0.30175553 -0.10998314  0.29405306 -0.07784496 -0.00668554 -0.046464
31
 -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 [17]:

```
RMSE = (y_error**2).sum()/100
```

In [18]:

```
print(RMSE)
```

```
0.174454680848951
```