Machine Learning LAB Assignment 1

20BCE0083 - Jeevan Yohan Varghese

Slot : L43+L44

Numpy Problems

In [2]:
```
import numpy as np
```

1. Create an array of 6 zeros

In [3]:
```
data=np.zeros((1,6),dtype=np.int64 )
print(data)
```

```
[[0 0 0 0 0 0]]
```

2. . Create an array of 6 ones

In [4]:
```
data=np.ones((1,6),dtype=np.int64)
print(data)
```

```
[[1 1 1 1 1 1]]
```

3. Create an array of 6 fives

In [5]:
```
data=np.ones(6,dtype=np.int64)*5
print(data)
```

```
[5 5 5 5 5 5]
```

4. Create an array of integers from 1 to 99

In [6]:
```
data=np.arange(1,100)
print(data)
```

```
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72
 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96
 97 98 99]
```

5. Create an array of all the odd integers ranging from 1 to 99

In [7]:
```
data=np.arange(1,100,2)
print(data)
```

```
[ 1  3  5  7  9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47
 49 51 53 55 57 59 61 63 65 67 69 71 73 75 77 79 81 83 85 87 89 91 93 95
 97 99]
```

6.Create a 2X2 matrix filled with values from 1 to 4

In [8]:
```
data=np.arange(1,5).reshape(2,2)
print(data)
```

```
[[1 2]
 [3 4]]
```

7. Create a 3X3 matrix filled with values from 9 to 17

In [9]:
```
data=np.arange(9,18).reshape(3,3)
print(data)
```

```
[[ 9 10 11]
 [12 13 14]
 [15 16 17]]
```

8. Make an identity matrix of 4X4

In [10]:
```
data=np.identity(4,dtype=np.int64)
print(data)
```

```
[[1 0 0 0]
 [0 1 0 0]
 [0 0 1 0]
 [0 0 0 1]]
```

9. With the help of Numpy generate a random nos in between 0 to 1

In [11]:
```
data=np.random.rand()
print(data)
```

```
0.08787494915513927
```

10. Create 10 points that are space linearly from each

In [12]:
```
data=np.linspace(1,10,10)
print(data)
```

```
[ 1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
```

11. Compare two 3d array and display the results in terms of true and false.

In [13]:
```
arr1=np.array(
    [
    [[1,2],[3,4],[4,5]],
    [[3,2],[5,7],[6,2]],
    [[2,5],[2,1],[3,8]],
    ]
        )
arr2=np.array(
```

```
    [
    [[1,4],[8,8],[4,5]],
    [[3,8],[5,7],[2,2]],
    [[2,3],[2,0],[9,8]],
    ])
print(arr1==arr2)
#np.where((arr1==arr2).all(axis=1,keepdims=True),[0,0],)
```

```
[[[ True False]
  [False False]
  [ True  True]]

 [[ True False]
  [ True  True]
  [False  True]]

 [[ True False]
  [ True False]
  [False  True]]]
```

12. Create a null vector of size 20 where the 6th value should be updated as 5.

In [14]:
```
data=np.zeros(20,dtype=np.int64)
print(data)
data[5]=5
print(data)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 5 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

13. Reverse an array of size 100 using numpy.

In [15]:
```
data=np.arange(0,100)
datarev=np.flip(data)
print(datarev)
```

```
[99 98 97 96 95 94 93 92 91 90 89 88 87 86 85 84 83 82 81 80 79 78 77 76
 75 74 73 72 71 70 69 68 67 66 65 64 63 62 61 60 59 58 57 56 55 54 53 52
 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32 31 30 29 28
 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10  9  8  7  6  5  4
  3  2  1  0]
```

14. Find the minimum and maximum values of a 20X20 array using numpy

In [16]:
```
data=np.random.randint(0,400,size=(20,20))#.reshape(20,20)
#print(data)
print("Max : ",np.max(data))
print("Min : ",np.min(data))
```

```
Max :  399
Min :  0
```

15. Find mean value of a randomly generated array of size 50.

In [17]:
```
data=np.random.randint(0,300,size=(50))
```

```
print(data)
print(np.mean(data))
```

```
[166 227 156 267 155  64 255 279  26  21  31 204  89  39 158 112  57  98
  55 141 282 127 213 170 195 173 275  25 248 100 100 113  83 175  49 297
 242  70 141 239  23 177  18 119 275 100 250 281 242 244]
152.92
```

16. “ A 20 X20 array filled with zeros at all borders and all 1’s inside”-create such array.

In [18]:
```
data=np.zeros((20,20),dtype=np.int64)
data[1:-1,1:-1]=1
print(data)
```

```
[[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0]
 [0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0]
 [0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0]
 [0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0]
 [0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0]
 [0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0]
 [0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0]
 [0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0]
 [0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0]
 [0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0]
 [0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0]
 [0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0]
 [0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0]
 [0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0]
 [0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0]
 [0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0]
 [0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0]
 [0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]]
```

17. Create an array of size 10X10 with 10 element valued as nan

In [19]:
```
data = np.ones((10, 10))
for i in range(10):
    data[i,i]=np.nan

print(data)
```

```
[[nan  1.  1.  1.  1.  1.  1.  1.  1.  1.]
 [ 1. nan  1.  1.  1.  1.  1.  1.  1.  1.]
 [ 1.  1. nan  1.  1.  1.  1.  1.  1.  1.]
 [ 1.  1.  1. nan  1.  1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1. nan  1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1. nan  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.  1. nan  1.  1.  1.]
 [ 1.  1.  1.  1.  1.  1.  1. nan  1.  1.]
 [ 1.  1.  1.  1.  1.  1.  1.  1. nan  1.]
 [ 1.  1.  1.  1.  1.  1.  1.  1.  1. nan]]
```

18. Create a 4X4 matrix and the values just below the diagonal is 9 8 7

In [20]:
```
data = np.diag(1+np.arange(6,9), k = -1)
print (data)
```

```
[[0 0 0 0]
 [7 0 0 0]
 [0 8 0 0]
 [0 0 9 0]]
```

19. Create a check board pattern using numpy

In [21]:
```
data = np.zeros ((8,8), dtype=int)
data[1::2, ::2]= 1
data[::2, 1::2] = 1
print (data)
```

```
[[0 1 0 1 0 1 0 1]
 [1 0 1 0 1 0 1 0]
 [0 1 0 1 0 1 0 1]
 [1 0 1 0 1 0 1 0]
 [0 1 0 1 0 1 0 1]
 [1 0 1 0 1 0 1 0]
 [0 1 0 1 0 1 0 1]
 [1 0 1 0 1 0 1 0]]
```

20. Print the dtype of int32 and float64 data type.

In [22]:
```
print(np.dtype(np.int32))
print(np.dtype(np.float64))
```

```
int32
float64
```

Pandas Problems

In [23]:
```
import pandas as pd
```

1. Create a pandas Series using lists and dictionaries

In [24]:
```
#using list
l=["Argentina","France","Brazil"]
data=pd.Series(l)
print(data)
```

```
0    Argentina
1       France
2       Brazil
dtype: object
```

In [25]:
```
#using dictionary
d={"Argentina":"Messi","France":"Mbappe","Brazil":"Neymar"}
data=pd.Series(d)
print(data)
```

```
Argentina     Messi
France       Mbappe
Brazil       Neymar
dtype: object
```

2. Create series using NumPy functions in Pandas

In [26]:
```
data=pd.Series(np.random.rand(10))
print(data)
print()
data=pd.Series(np.linspace(1,100,8))
print(data)
```

```
0    0.138944
1    0.206337
2    0.165384
3    0.612013
4    0.771923
5    0.215269
6    0.018310
7    0.667409
8    0.172723
9    0.607430
dtype: float64

0      1.000000
1     15.142857
2     29.285714
3     43.428571
4     57.571429
5     71.714286
6     85.857143
7    100.000000
dtype: float64
```

3. Find indices and values of series

In [27]:
```
d={"Argentina":"Messi","France":"Mbappe","Brazil":"Neymar"}
data=pd.Series(d)
print(data.index)
print(data.values)
```

```
Index(['Argentina', 'France', 'Brazil'], dtype='object')
['Messi' 'Mbappe' 'Neymar']
```

4. How to specify an index while creating Series in Pandas?

Find Length Size and Shape of a Series in Pandas?

In [28]:
```
l=["Argentina","France","Brazil"]
data=pd.Series(l,index=[10,11,12])
print(data)
```

```
10    Argentina
11       France
12       Brazil
dtype: object
```

In [29]:
```
print(data.size)
print(data.shape)
```

```
3
(3,)
```

5. Find first and last few values

In [30]:
```
data=pd.Series(["India","China","Pakistan","Sri Lanka","Bangladesh","UAE","Quatar",
print("First two values : ")
print(data.head(2))
print("\nLast two values : ")
print(data.tail(2))
```

```
First two values :
0    India
1    China
dtype: object

Last two values :
7    Kuwait
8     Saudi
dtype: object
```

6. Handling missing values

In [31]:
```
data=pd.read_csv('train.csv')

print(data.isnull().sum())
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

In [32]:
```
# 1. Deleting Row
df = data.copy()
df.dropna(inplace=True)
print("After deleting rows with missing values : ")
print(df.isnull().sum())
#print(df)
print()
```

```
After deleting rows with missing values :
PassengerId    0
Survived       0
Pclass         0
Name           0
Sex            0
Age            0
SibSp          0
Parch          0
Ticket         0
Fare           0
Cabin          0
Embarked       0
dtype: int64
```

In [33]:
```python
#2. Insert Mean



df=data.copy()
print("Age values before inserting mean")
print(df['Age'].head(10))
df['Age'] = df['Age'].fillna(df['Age'].mean())

print("\nAge values after inserting mean")
print(df['Age'].head(10))
print(df.isnull().sum())
print()
```

```
Age values before inserting mean
0    22.0
1    38.0
2    26.0
3    35.0
4    35.0
5     NaN
6    54.0
7     2.0
8    27.0
9    14.0
Name: Age, dtype: float64

Age values after inserting mean
0    22.000000
1    38.000000
2    26.000000
3    35.000000
4    35.000000
5    29.699118
6    54.000000
7     2.000000
8    27.000000
9    14.000000
Name: Age, dtype: float64
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age              0
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

In [34]:

```
#3. Insert Mode
df=data.copy()
print("\nAge values before inserting mode")
print(df['Age'].head(10))
df['Age'] = df['Age'].fillna(df['Age'].mode()[0])
print("\nAge values after inserting mode")
print(df['Age'].head(10))
print(df.isnull().sum())
#print(df)
```

```
Age values before inserting mode
0    22.0
1    38.0
2    26.0
3    35.0
4    35.0
5     NaN
6    54.0
7     2.0
8    27.0
9    14.0
Name: Age, dtype: float64

Age values after inserting mode
0    22.0
1    38.0
2    26.0
3    35.0
4    35.0
5    24.0
6    54.0
7     2.0
8    27.0
9    14.0
Name: Age, dtype: float64
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age              0
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

In [35]:

```
#4 Insert median
df=data.copy()
print("\nAge values before inserting median")
print(df['Age'].head(10))
df['Age'] = df['Age'].fillna(df['Age'].median())
print("\nAge values after inserting median")
print(df['Age'].head(10))
print(df.isnull().sum())
```

```
Age values before inserting median
0    22.0
1    38.0
2    26.0
3    35.0
4    35.0
5     NaN
6    54.0
7     2.0
8    27.0
9    14.0
Name: Age, dtype: float64

Age values after inserting median
0    22.0
1    38.0
2    26.0
3    35.0
4    35.0
5    28.0
6    54.0
7     2.0
8    27.0
9    14.0
Name: Age, dtype: float64
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age              0
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

In [36]:

```
#5. Last Observation Carried Forward
df=data.copy()
print("Age Values before LOCF")
print(df['Age'].head(10))
print("\nAge Values after LOCF")
df["Age"] = df["Age"].fillna(method='ffill')
print(df['Age'].head(10))
print(df.isnull().sum())
```

```
Age Values before LOCF
0    22.0
1    38.0
2    26.0
3    35.0
4    35.0
5     NaN
6    54.0
7     2.0
8    27.0
9    14.0
Name: Age, dtype: float64

Age Values after LOCF
0    22.0
1    38.0
2    26.0
3    35.0
4    35.0
5    35.0
6    54.0
7     2.0
8    27.0
9    14.0
Name: Age, dtype: float64
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age              0
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

In [37]:

```
#6. Imputation

df=data.copy()
print("\n Cabin values before imputation")
print(df['Cabin'].head(10))
df['Cabin'] = df['Cabin'].fillna('Z')
print("\n Cabin values after imputation")
print(df['Cabin'].head(10))
print(df.isnull().sum())
```

```
 Cabin values before imputation
0     NaN
1     C85
2     NaN
3    C123
4     NaN
5     NaN
6     E46
7     NaN
8     NaN
9     NaN
Name: Cabin, dtype: object

 Cabin values after imputation
0       Z
1     C85
2       Z
3    C123
4       Z
5       Z
6     E46
7       Z
8       Z
9       Z
Name: Cabin, dtype: object
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin            0
Embarked         2
dtype: int64
```

In [38]:

```
# 7. Interpolation
df=data.copy()
print("Age Values before interpolation")
print(df['Age'].head(10))
df["Age"] = df["Age"].interpolate(method='linear', limit_direction='forward', axis=
print("\nAge Values after interpolation")
print(df['Age'].head(10))
```

```
Age Values before interpolation
0    22.0
1    38.0
2    26.0
3    35.0
4    35.0
5     NaN
6    54.0
7     2.0
8    27.0
9    14.0
Name: Age, dtype: float64

Age Values after interpolation
0    22.0
1    38.0
2    26.0
3    35.0
4    35.0
5    44.5
6    54.0
7     2.0
8    27.0
9    14.0
Name: Age, dtype: float64
```

7. Checking the duplicate rows and columns using pandas.

In [39]:
```
stud=[("John",20,"A"),("Sam",21,"S"),("John",20,"A"),("Steve",20,"B"),("Tim",21,"B"
data=pd.DataFrame(stud,columns=["Student","Age","Grade"])
print(data[data.duplicated()])
```

```
  Student  Age Grade
2    John   20     A
```

8. Normalizing Data

In [44]:
```
#MinMaxScaler
from sklearn.preprocessing import MinMaxScaler

data=pd.DataFrame({"weight":np.random.randint(1,10,40),"height":np.random.randint(4
print(data)

scaler = MinMaxScaler()
df_norm = pd.DataFrame(scaler.fit_transform(data), columns=data.columns)

df_norm
```

	weight	height
0	3	75009
1	5	60037
2	6	68768
3	1	76793
4	6	70945
5	3	53909
6	1	54355
7	7	52246
8	3	42426
9	7	44320
10	4	54510
11	2	58027
12	7	73807
13	6	59017
14	1	66472
15	2	43752
16	8	58822
17	7	60341
18	6	46711
19	2	69764
20	5	62288
21	8	45874
22	2	41961
23	8	54463
24	6	79075
25	4	71072
26	5	48425
27	1	62616
28	2	73408
29	4	69822
30	3	52055
31	2	64700
32	8	45734
33	5	78058
34	8	61757
35	8	43918
36	1	46908
37	8	72557
38	9	61480
39	7	60166

Out[44]:

	weight	height
0	0.250	0.890446
1	0.500	0.487040
2	0.625	0.722288
3	0.000	0.938514
4	0.625	0.780945
5	0.250	0.321927
6	0.000	0.333944
7	0.750	0.277119
8	0.250	0.012529
9	0.750	0.063561
10	0.375	0.338120
11	0.125	0.432882
12	0.750	0.858059
13	0.625	0.459557
14	0.000	0.660425
15	0.125	0.048257
16	0.875	0.454303
17	0.750	0.495231
18	0.625	0.127984
19	0.125	0.749124
20	0.500	0.547691
21	0.875	0.105432
22	0.125	0.000000
23	0.875	0.336854
24	0.625	1.000000
25	0.375	0.784367
26	0.500	0.174166
27	0.000	0.556529
28	0.125	0.847308
29	0.375	0.750687
30	0.250	0.271973
31	0.125	0.612680
32	0.875	0.101660

	weight	height
33	0.500	0.972598
34	0.875	0.533384
35	0.875	0.052729
36	0.000	0.133292
37	0.875	0.824379
38	1.000	0.525920
39	0.750	0.490516

In [45]:

```
#Z Score
from sklearn.preprocessing import StandardScaler


std_scaler = StandardScaler()
std_scaler

df_std = pd.DataFrame(std_scaler.fit_transform(data), columns=data.columns)

df_std
```

Out[45]:

	weight	height
0	-0.703039	1.392526
1	0.089118	0.034274
2	0.485196	0.826346
3	-1.495196	1.554370
4	0.485196	1.023842
5	-0.703039	-0.521655
6	-1.495196	-0.481194
7	0.881275	-0.672522
8	-0.703039	-1.563388
9	0.881275	-1.391565
10	-0.306961	-0.467133
11	-1.099118	-0.148072
12	0.881275	1.283482
13	0.485196	-0.058260
14	-1.495196	0.618054
15	-1.099118	-1.443093
16	1.277353	-0.075950
17	0.881275	0.061853
18	0.485196	-1.174654
19	-1.099118	0.916703
20	0.089118	0.238483
21	1.277353	-1.250587
22	-1.099118	-1.605572
23	1.277353	-0.471397
24	0.485196	1.761392
25	-0.306961	1.035364
26	0.089118	-1.019161
27	-1.495196	0.268239
28	-1.099118	1.247284
29	-0.306961	0.921964
30	-0.703039	-0.689849
31	-1.099118	0.457299
32	1.277353	-1.263287

	weight	height
33	0.089118	1.669130
34	1.277353	0.190311
35	1.277353	-1.428034
36	-1.495196	-1.156783
37	1.277353	1.170082
38	1.673431	0.165182
39	0.881275	0.045977

In [41]:

```
def min_max_scaling(df):

    df_norm = df.copy()

    for column in df_norm.columns:
        df_norm[column] = (df_norm[column] - df_norm[column].min()) / (df_norm[colu

    return df_norm

def z_score(df):
    df_std = df.copy()
    for column in df_std.columns:
        df_std[column] = (df_std[column] - df_std[column].mean()) / df_std[column].

    return df_std
data=pd.DataFrame({"weight":np.random.randint(1,10,40),"height":np.random.randint(4
print(data)
data_normalized = min_max_scaling(data)
print("\nMin Max Normalized ")
print(data_normalized)
data_z_score=z_score(data)
print("\nZ Score ")
print(data_z_score)
```

```
    weight  height
0        3   53792
1        8   77189
2        1   50773
3        2   51202
4        6   43143
5        8   42461
6        6   48794
7        7   73447
8        8   75519
9        4   47201
10       6   57586
11       1   56375
12       5   56782
13       3   73283
14       5   63304
15       4   67127
16       8   67366
17       5   59018
18       3   57475
19       8   45832
20       7   78874
21       1   41107
22       4   66510
23       1   71488
24       2   46448
25       5   67577
26       5   47181
27       7   42001
28       8   45959
29       5   72187
30       7   43571
31       5   41265
32       8   79332
33       7   73609
34       4   65898
35       6   75693
36       8   51856
37       8   58460
38       4   47428
39       8   79528

Min Max Normalized
      weight    height
0   0.285714  0.330158
1   1.000000  0.939122
2   0.000000  0.251581
3   0.142857  0.262747
4   0.714286  0.052992
5   1.000000  0.035241
6   0.714286  0.200073
7   0.857143  0.841727
8   1.000000  0.895656
9   0.428571  0.158611
10  0.714286  0.428906
11  0.000000  0.397387
```

```
12  0.571429  0.407980
13  0.285714  0.837459
14  0.571429  0.577731
15  0.428571  0.677234
16  1.000000  0.683454
17  0.571429  0.466177
18  0.285714  0.426017
19  1.000000  0.122980
20  0.857143  0.982978
21  0.000000  0.000000
22  0.428571  0.661175
23  0.000000  0.790739
24  0.142857  0.139013
25  0.571429  0.688946
26  0.571429  0.158091
27  0.857143  0.023269
28  1.000000  0.126285
29  0.571429  0.808933
30  0.857143  0.064132
31  0.571429  0.004112
32  1.000000  0.994899
33  0.857143  0.845944
34  0.428571  0.645246
35  0.714286  0.900185
36  1.000000  0.279769
37  1.000000  0.451654
38  0.428571  0.164519
39  1.000000  1.000000

Z Score
      weight    height
0  -0.980460 -0.416012
1   1.174397  1.420824
2  -1.842403 -0.653026
3  -1.411432 -0.619346
4   0.312454 -1.252037
5   1.174397 -1.305579
6   0.312454 -0.808392
7   0.743426  1.127049
8   1.174397  1.289717
9  -0.549489 -0.933454
10  0.312454 -0.118156
11 -1.842403 -0.213228
12 -0.118517 -0.181275
13 -0.980460  1.114174
14 -0.118517  0.330749
15 -0.549489  0.630883
16  1.174397  0.649646
17 -0.118517 -0.005733
18 -0.980460 -0.126870
19  1.174397 -1.040931
20  0.743426  1.553109
21 -1.842403 -1.411878
22 -0.549489  0.582444
23 -1.842403  0.973254
24 -1.411432 -0.992570
```

```
25 -0.118517  0.666211
26 -0.118517 -0.935024
27  0.743426 -1.341692
28  1.174397 -1.030960
29 -0.118517  1.028130
30  0.743426 -1.218436
31 -0.118517 -1.399474
32  1.174397  1.589065
33  0.743426  1.139768
34 -0.549489  0.534397
35  0.312454  1.303377
36  1.174397 -0.568003
37  1.174397 -0.049540
38 -0.549489 -0.915633
39  1.174397  1.604453
```

In [ ]: