Python for Data Science (Lab Session-14)

EDA Process

Q-1) NumPy array manipulation

A => Creare a Numpy Array of dimension 5 x 4 using random number generator.

Explore about different distributions of random numbers

In [1]:
```
from numpy import random
import numpy as np
x = random.randint(100, size=(5, 4))

print(x)
```

```
[[35  7 59 18]
 [24 43 23 67]
 [ 4 11 79 13]
 [90 71 63 30]
 [10 44 81  7]]
```

B => Get the values of attributes like ndim, shape, size, itemsize, dtype, nbytes

In [2]:
```
#dimension
print(x.ndim)
```

```
2
```

In [3]:
```
#Shape of data
print(x.shape)
```

```
(5, 4)
```

In [4]:
```
#Size of it
print(x.size)
```

```
20
```

In [5]:
```
# item size or give column
print(x.itemsize)
```

```
4
```

In [6]:
```
#data type
print(x.dtype)
```

```
int32
```

In [7]:
```
#Give total byte
print(x.nbytes)
```

```
80
```

C => Sort the array using 2nd row and then sort it again using 4th column

In [8]:
```
snd_row = x[:,x[1].argsort()]
snd_row
# second tow sort
```

Out[8]:
```
array([[59, 35,  7, 18],
       [23, 24, 43, 67],
       [79,  4, 11, 13],
       [63, 90, 71, 30],
       [81, 10, 44,  7]])
```

In [9]:
```
col4 = snd_row[np.argsort(snd_row[:,3])]
col4
```

Out[9]:
```
array([[81, 10, 44,  7],
       [79,  4, 11, 13],
       [59, 35,  7, 18],
       [63, 90, 71, 30],
       [23, 24, 43, 67]])
```

D => Perform slicing of rows and columns and dicing

In [10]:
```
print("Extract Second Row to End\n")
print(x[2:])
```

```
Extract Second Row to End

[[ 4 11 79 13]
 [90 71 63 30]
 [10 44 81  7]]
```

In [11]:
```
print("Extract Second Row and All Column\n")
print(x[1])
```

```
Extract Second Row and All Column

[24 43 23 67]
```

In [12]:
```
print("Extract All Row with Second column.")
x[:,[2]]
```

```
Extract All Row with Second column.
```

Out[12]:
```
array([[59],
       [23],
       [79],
       [63],
       [81]])
```

In [13]:
```
print("Extract Third Row of fourth element")
x[3:4,[3]]
```

```
Extract Third Row of fourth element
```

Out[13]:
```
array([[30]])
```

E => Perform searching of any element from the array

In [14]:
```
result = np.where(x==74)
print("Row Index",result[0],"And Column Index",result[1],"Found Element 74")
```

```
Row Index [] And Column Index [] Found Element 74
```

Q-2) EDA Process

A => Create bag of words for any two sentences using CountVectorizer and Hashing Vectorizer.

Compare the performance considering execution time and memory usage

In [15]:
```
from nltk.tokenize import sent_tokenize
import pandas as pd
import time
from sklearn.feature_extraction.text import CountVectorizer,HashingVectorizer

st_time = time.time()
mydata = "I am Arjun Vankani.Here nowday's working with Ml. Wish to do bettter"
se = sent_tokenize(mydata)

for i in se:
    print(i)
cv = CountVectorizer()
cv_data = cv.fit([i for i in se])
print("Bag OF Words:")

cv_tr = cv.transform([i for i in se])
pd.DataFrame(cv_tr.toarray(),columns=cv.get_feature_names())
end_time = time.time()
tot_time = end_time - st_time
print("Totle Execution Time Of CountVectorizer",tot_time)
```

```
I am Arjun Vankani.Here nowday's working with Ml.
Wish to do bettter
Bag OF Words:
Totle Execution Time Of CountVectorizer 0.010001182556152344
```

In [16]:
```
hs_st_time = time.time()
text = ["This is Example sentence."]
vt = HashingVectorizer(n_features=10)
ans = vt.transform(text)
print(ans.toarray())
hs_end_time = time.time()
hs_tot_time=hs_end_time-hs_st_time
print("Totle Execution Time Of Hashing Vectorizer",hs_tot_time)
```

```
[[ 0.   0.   0.5  0.   0.   0.   0.   0.5 -0.5 -0.5]]
Totle Execution Time Of Hashing Vectorizer 0.002001047134399414
```

B => Measure central tendency,variance and range from iris dataset. Also print quantile valuesto display min, max, median, 25% and 75%value of the data distribution

In [17]:
```
from sklearn.datasets import load_iris
iris = load_iris()
iris_data = iris.data
print(iris_data)
```

```
 [6.1 3.  4.9 1.8]
 [6.4 2.8 5.6 2.1]
 [7.2 3.  5.8 1.6]
 [7.4 2.8 6.1 1.9]
 [7.9 3.8 6.4 2. ]
 [6.4 2.8 5.6 2.2]
 [6.3 2.8 5.1 1.5]
 [6.1 2.6 5.6 1.4]
 [7.7 3.  6.1 2.3]
 [6.3 3.4 5.6 2.4]
 [6.4 3.1 5.5 1.8]
 [6.  3.  4.8 1.8]
 [6.9 3.1 5.4 2.1]
 [6.7 3.1 5.6 2.4]
 [6.9 3.1 5.1 2.3]
 [5.8 2.7 5.1 1.9]
 [6.8 3.2 5.9 2.3]
 [6.7 3.3 5.7 2.5]
 [6.7 3.  5.2 2.3]
 [6.3 2.5 5.  1.9]
```

In [18]:
```
print("Varience Of Iris Data",np.var(iris_data))
print("Range Of Iris Data",np.ptp(iris_data))
print("Minimum Of Iris Dataset Element",np.amin(iris_data))
print("Maximum Of Iris Dataset Element:",np.amax(iris_data))
print("Median Of Iris Dataset Element:",np.median(iris_data))
```

```
Varience Of Iris Data 3.896056416666667
Range Of Iris Data 7.800000000000001
Minimum Of Iris Dataset Element 0.1
Maximum Of Iris Dataset Element: 7.9
Median Of Iris Dataset Element: 3.2
```

In [19]:
```
import pandas as pd
da = pd.DataFrame(iris_data)
da.describe()
(iris_data.sum()*25)/100
print("Covariance Of Iris Dataset Element:\n",np.cov(iris_data))
```

```
Covariance Of Iris Dataset Element:
 [[4.75       4.42166667 4.35333333 ... 2.915      2.475      2.6       ]
 [4.42166667 4.14916667 4.055      ... 2.95583333 2.50416667 2.62833333]
 [4.35333333 4.055      3.99       ... 2.68833333 2.28166667 2.39666667]
 ...
 [2.915      2.95583333 2.68833333 ... 4.18916667 3.65083333 3.835     ]
 [2.475      2.50416667 2.28166667 ... 3.65083333 3.20916667 3.375     ]
 [2.6        2.62833333 2.39666667 ... 3.835      3.375      3.55      ]]
```

C => Calculate covariance and correlation of the dataset

In [20]:
```
print("Covariance Of Iris Dataset Element:\n",np.cov(iris_data))
print("Correlation Of Iris Dataset Element:\n",np.corrcoef(iris_data))
```

```
Covariance Of Iris Dataset Element:
 [[4.75       4.42166667 4.35333333 ... 2.915      2.475      2.6       ]
 [4.42166667 4.14916667 4.055      ... 2.95583333 2.50416667 2.62833333]
 [4.35333333 4.055      3.99       ... 2.68833333 2.28166667 2.39666667]
 ...
 [2.915      2.95583333 2.68833333 ... 4.18916667 3.65083333 3.835     ]
 [2.475      2.50416667 2.28166667 ... 3.65083333 3.20916667 3.375     ]
 [2.6        2.62833333 2.39666667 ... 3.835      3.375      3.55      ]]
Correlation Of Iris Dataset Element:
 [[1.         0.99599866 0.99997391 ... 0.65347343 0.6339168  0.63315839]
 [0.99599866 1.         0.99660709 ... 0.70898277 0.68625679 0.68483481]
 [0.99997391 0.99660709 1.         ... 0.65755616 0.63763128 0.6368058 ]
 ...
 [0.65347343 0.70898277 0.65755616 ... 1.         0.99570813 0.99446012]
 [0.6339168  0.68625679 0.63763128 ... 0.99570813 1.         0.99991588]
 [0.63315839 0.68483481 0.6368058  ... 0.99446012 0.99991588 1.        ]]
```