NAME : kalpi
R.NO : 24
SUBJECT : DATA ANALYTICS (PRACTICAL)

TERM WORK

==========

*****
NumPy
*****

#Q-4 a

np.zeros(10)

======
output
======
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])

#Q-4 b

vowels = np.array([â€™aâ€™,â€™eâ€™,â€™iâ€™,â€™oâ€™,â€™uâ€™])
print(vowels)
======
output
======
array([â€™aâ€™, â€™eâ€™, â€™iâ€™, â€™oâ€™, â€™uâ€™], dtype=â€™<U1â€™)

#Q-4 c

np.ones((2,5),dtype=â€™intâ€™)
======
output
======
array([[1, 1, 1, 1, 1],
[1, 1, 1, 1, 1]])

#Q-4 d

list = [[2.7,-2,-19],[0,3.4,99.9],[10.6,0,13]]
arr1 = np.array(list)
print(arr1)
output:-
[[2.7 -2. -19.]
[0. 3.4 99.9]
[10.6 0. 13.]]

#Q-4 e

np.arange(4,5*3*4 + 1,4,dtype=â€™floatâ€™).reshape(3,5)
======

output
======
array([[4., 8., 12., 16., 20.],
[24., 28., 32., 36., 40.],
[44., 48., 52., 56., 60.]])

#Question 5
#(a)
zeros = np.zeros(10)
print(zeros.ndim)
print(zeros.shape)
print(zeros.size)
print(zeros.dtype)

======
output
======
1
(10,)
10
float64

#()
vowels = np.array([â€™aâ€™,â€™eâ€™,â€™iâ€™,â€™oâ€™,â€™uâ€™])
print(vowels.ndim)
print(vowels.shape)
print(vowels.size)
print(vowels.dtype)

======
output
======
1
(5,)
5
<U1

#
ones = np.ones(10)

list = [[2.7,-2,-19],[0,3.4,99.9],[10.6,0,13]]
myarr1 = np.array(list)

myarr2 = np.arange(9,dtype=â€™floatâ€™).reshape(3,3)

#(b)
print(ones.reshape((10,1)))

======
output
======

[[1.]
[1.]

[1.]
[1.]
[1.]
[1.]
[1.]
[1.]
[1.]
[1.]]

#(c)

print(vowels[1:3])

======
output
======
[â€™eâ€™ â€™iâ€™]

#(d)

print(myarr1[1:3])

======
output
======

[[0. 3.4 99.9]
[10.6 0. 13.]]

#(e)

print(myarr1[:,0:2])
======
output
======
[[2.7 -2.]
[0. 3.4]
[10.6 0.]]

#(f)

print(myarr1[2,0:2])
======
output
======
[10.6 0.]

#(g)

print(vowels[::-1])
======
output
======

[â€™uâ€™ â€™oâ€™ â€™iâ€™ â€™eâ€™ â€™aâ€™]

#6
#(a)
print(ones / 3)
======
output
======
[0.33333333 0.33333333 0.33333333 0.33333333 0.33333333 0.33333333
0.33333333 0.33333333 0.33333333 0.33333333]

#(b)

print(np.concatenate((myarr1, myarr2),axis = 1))

print("\n",np.add(myarr1, myarr2))
======
output
======
[[2.7 -2. -19. 0. 1. 2.]
[0. 3.4 99.9 3. 4. 5.]
[10.6 0. 13. 6. 7. 8.]]

[[2.7 -1. -17.]
[3. 7.4 104.9]
[16.6 7. 21.]]

#(c)

print(myarr1 - myarr2)
======
output
======
[[2.7 -3. -21.]
[-3. -0.6 94.9]
[4.6 -7. 5.]]

#(d)
List = []
for i in range(len(myarr1)):
List.append(myarr1[i] * myarr2[i])

print(np.array(List).reshape(3,3))

print("OR\n", myarr1 * myarr2)
======
output
======
[[0. -2. -38.]
[0. 13.6 499.5]
[63.6 0. 104.]]

#(e)

```
myarr3 = myarr1 @ myarr2
print(myarr3)
======
output
======
[[-120. -138.3 -156.6]
[ 609.6 712.9 816.2]
[ 78. 101.6 125.2]]

#(f)

print(myarr1 / myarr2)
======
output
======
[[ inf -2. -9.5 ]
[ 0. 0.85 19.98 ]
[ 1.76666667 0. 1.625 ]]

#(g)

print((myarr1 ** 3) / 2)
======
output
======
[[ 9.841500e+00 -4.000000e+00 -3.429500e+03]
[ 0.000000e+00 1.965200e+01 4.985015e+05]
[ 5.955080e+02 0.000000e+00 1.098500e+03]]

#(h)

print((np.round(((myarr1**(1/2)) / 2),decimals = 2)))
======
output
======
[[0.82 nan nan]
[0. 0.92 5. ]
[1.63 0. 1.8 ]]

#7
#(a)

print(ones.transpose())
print()
print(myarr2.transpose())
======
output
======
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]

[[0. 3. 6.]
[1. 4. 7.]
[2. 5. 8.]]
```

#(b)

print(vowels[::-1])
======
output
======
[â€™uâ€™ â€™oâ€™ â€™iâ€™ â€™eâ€™ â€™aâ€™]

#(c)

print(myarr1)
myarr1.sort(axis = 1)
print("\n",myarr1)
======
output
======
[[-19. -2. 2.7]
[0. 3.4 13.]
[0. 10.6 99.9]]

[[-19. -2. 2.7]
[0. 3.4 13.]
[0. 10.6 99.9]]

#8 (a)
myarray2A, myarray2B, myarray2C, myarray2D, myarray2E = np.split(myarray2, 5, axis=1)

print("myarray2A:\n", myarray2A)
print("myarray2B:\n", myarray2B)
print("myarray2C:\n", myarray2C)
print("myarray2D:\n", myarray2D)
print("myarray2E:\n", myarray2E)

======
output
======
myarray2A:
[[4.]
[8.]
[12.]]
myarray2B:
[[8.]
[12.]
[16.]]
myarray2C:
[[12.]
[16.]
[20.]]
myarray2D:
[[16.]
[20.]
[24.]]
myarray2E:

```
[[20.]
[24.]
[28.]]


#(b)
zerosA, zerosB, zerosC, zerosD = np.split(zeros, [2, 5, 7, 8])

print("zerosA:\n", zerosA)
print("zerosB:\n", zerosB)
print("zerosC:\n", zerosC)
print("zerosD:\n", zerosD)

======
output
======
zerosA:
[0. 0.]
zerosB:
[0. 0. 0.]
zerosC:
[0. 0.]
zerosD:
[0.]


#(c)

concatenated_array = np.concatenate((myarray2A, myarray2B, myarray2C), axis=1)

print("Concatenated Array:\n", concatenated_array)

======
output
======
Concatenated Array:
[[ 4. 8. 12.]
[ 8. 12. 16.]
[12. 16. 20.]]

#9

values = np.arange(-1, -1 + 14 * 3 * 0.25, 0.25)

myarray4 = values.reshape((14, 3))

split_arrays = np.array_split(myarray4, 3, axis=0)


for i, part in enumerate(split_arrays):
print(f"Part {i + 1}:")
print(part)
print()
```

======
output
======
Part 1:
[[-1. -0.75 -0.5]
[-0.25 0. 0.25]
[0.5 0.75 1.]
[1.25 1.5 1.75]
[2. 2.25 2.5]]

Part 2:
[[2.75 3. 3.25]
[3.5 3.75 4.]
[4.25 4.5 4.75]
[5. 5.25 5.5]
[5.75 6. 6.25]]

Part 3:
[[6.5 6.75 7.]
[7.25 7.5 7.75]
[8. 8.25 8.5]
[8.75 9. 9.25]]

#10

```
# a) Find the sum of all elements
sum_all_elements = np.sum(myarray4)

# b) Find the sum of all elements row-wise
sum_row_wise = np.sum(myarray4, axis=1)

# c) Find the sum of all elements column-wise
sum_column_wise = np.sum(myarray4, axis=0)

# d) Find the max of all elements
max_all_elements = np.max(myarray4)

# e) Find the min of all elements in each row
min_each_row = np.min(myarray4, axis=1)

# f) Find the mean of all elements in each row
mean_each_row = np.mean(myarray4, axis=1)

# g) Find the standard deviation column-wise
std_column_wise = np.std(myarray4, axis=0)

# Print results
print("a) Sum of all elements:", sum_all_elements)
print("b) Sum of all elements row-wise:", sum_row_wise)
print("c) Sum of all elements column-wise:", sum_column_wise)
print("d) Max of all elements:", max_all_elements)
print("e) Min of all elements in each row:", min_each_row)
print("f) Mean of all elements in each row:", mean_each_row)
print("g) Standard deviation column-wise:", std_column_wise)
```

======
output
======

a) Sum of all elements: 173.25
b) Sum of all elements row-wise: [-2.25 0. 2.25 4.5 6.75 9. 11.25 13.5 15.75 18. 20.25 22.5
24.75 27.]
c) Sum of all elements column-wise: [54.25 57.75 61.25]
d) Max of all elements: 9.25
e) Min of all elements in each row: [-1. -0.25 0.5 1.25 2. 2.75 3.5 4.25 5. 5.75 6.5 7.25
8. 8.75]
f) Mean of all elements in each row: [-0.75 0. 0.75 1.5 2.25 3. 3.75 4.5 5.25 6. 6.75 7.5
8.25 9.]
g) Standard deviation column-wise: [3.02334666 3.02334666 3.02334666]

==========

Pandas - 1

#Question 5

#(a)

```
import string
inde = list(range(1,27))

EngAlpha = pd.Series(list(string.ascii_uppercase))

#(b)

vowels = pd.Series([0,0,0,0,0],index=[â€™aâ€™,â€™eâ€™,â€™iâ€™,â€™oâ€™,â€™uâ€™])

print(vowels.empty)

#(c)

friends = pd.Series({â€™kalpiâ€™:1,â€™vedanshiâ€™:2,â€™yashviâ€™:3,â€™dhruviâ€™:4})

print(friends)

#(d)
MTSeries = pd.Series()

print(MTSeries.empty)

#(e)
MonthDays = [31,29,31,30,31,30,31,31,30,31,30,31]
ind = list(range(1,13))
MonthDaysS = pd.Series(MonthDays, index = ind)#index = np.arange(1,13)
```

```
print(MonthDaysS)

======
output
======

False
kalpi 1
vedanshi 2
yashvi 3
dhruvi 4
dtype: int64
True
1 31
2 29
3 31
4 30
5 31
6 30
7 31
8 31
9 30
10 31
11 30
12 31
dtype: int64

#Question 6

#(a)

# vowels[:] = 10
vowels.loc[â€™aâ€™:â€™uâ€™] = 10
print(vowels)

#(b)

vowels = vowels / 2
print(vowels)

#(C)

vowels1 = pd.Series([2,5,6,3,8],index = [â€™aâ€™,â€™eâ€™,â€™iâ€™,â€™oâ€™,â€™uâ€™])
print(vowels1)

#(d)

vowels2 = vowels + vowels1

print(vowels2)

#(e)

print("\nSubstract \n",vowels - vowels1)
```

```
print("\nMultiply \n",vowels * vowels1)
print("\nDivide \n",vowels / vowels1)

#(f)

vowels1 = vowels1.rename({â€™aâ€™:â€™Aâ€™,â€™eâ€™:â€™Eâ€™,â€™iâ€™:â€™Iâ€™,â€™oâ€™:â€™Oâ€™,â€™uâ€™:â€™Uâ€™})
print(vowels1)

vowels.index = (â€™aâ€™,â€™eâ€™,â€™iâ€™,â€™oâ€™,â€™uâ€™)
print(vowels)
======
output
======
a 10
e 10
i 10
o 10
u 10
dtype: int64
a 5.0
e 5.0
i 5.0
o 5.0
u 5.0
dtype: float64
a 2
e 5
i 6
o 3
u 8
dtype: int64
a 7.0
e 10.0
i 11.0
o 8.0
u 13.0
dtype: float64

Substract
a 3.0
e 0.0
i -1.0
o 2.0
u -3.0
dtype: float64

Multiply
a 10.0
e 25.0
i 30.0
o 15.0
u 40.0
dtype: float64
```

Divide
a 2.500000
e 1.000000
i 0.833333
o 1.666667
u 0.625000
dtype: float64
A 2
E 5
I 6
O 3
U 8
dtype: int64
a 5.0
e 5.0
i 5.0
o 5.0
u 5.0
dtype: float64

#Question 7

#(a)

```
print("\nDimension = ",EngAlpha.ndim,"\tSize = ",EngAlpha.size,"\tValues = ",EngAlpha.values)
print("\nDimension = ",vowels.ndim,"\tSize = ",vowels.size,"\tValues = ",vowels.values)
print("\nDimension = ",friends.ndim,"\tSize = ",friends.size,"\tValues = ",friends.values)
print("\nDimension = ",MonthDaysS.ndim,"\tSize = ",MonthDaysS.size,"\tValues = ",MonthDaysS.values)
print("\nDimension = ",MTSeries.ndim,"\tSize = ",MTSeries.size,"\tValues = ",MTSeries.values)
```

#(b)

```
MTSeries.name = "SeriesEmpty"

print(MTSeries.name)
```

#(c)

```
MonthDaysS.index.name = â€™monthnoâ€™
friends.index.name = â€™fnameâ€™
print(MonthDaysS.index.name)
print(friends)
```

#(d)

```
print(friends[[3,2]])
```

#(e)

```
print(EngAlpha[4:16])
```

#(f)

```
print(EngAlpha.head(10))

#(g)

print(EngAlpha.tail(10))

#(h)
print(MTSeries)

======
output
======
Dimension = 1 Size = 26 Values = [â€™Aâ€™ â€™Bâ€™ â€™Câ€™ â€™Dâ€™ â€™Eâ€™ â€™Fâ€™ â€™Gâ
€™ â€™Hâ€™ â€™Iâ€™ â€™Jâ€™ â€™Kâ€™ â€™Lâ€™ â€™Mâ€™ â€™Nâ€™ â€™Oâ€™ â€™Pâ€™ â
€™Qâ€™ â€™Râ€™
â€™Sâ€™ â€™Tâ€™ â€™Uâ€™ â€™Vâ€™ â€™Wâ€™ â€™Xâ€™ â€™Yâ€™ â€™Zâ€™]

Dimension = 1 Size = 5 Values = [5. 5. 5. 5. 5.]

Dimension = 1 Size = 5 Values = [ 8 14 27 35 36]

Dimension = 1 Size = 12 Values = [31 29 31 30 31 30 31 31 30 31 30 31]

Dimension = 1 Size = 0 Values = []
SeriesEmpty
monthno
fname
kalpi 1
vedandhi 2
yashvi 3
dhruvi 4
dtype: int64
fname
abc 5
def 7
dtype: int64
4 E
5 F
6 G
7 H
8 I
9 J
10 K
11 L
12 M
13 N
14 O
15 P
dtype: object
0 A
1 B
2 C
3 D
```

4 E
5 F
6 G
7 H
8 I
9 J
dtype: object
16 Q
17 R
18 S
19 T
20 U
21 V
22 W
23 X
24 Y
25 Z
dtype: object
Series([], Name: SeriesEmpty, dtype: object)

```
#Question 8
#(a)

print(MonthDaysS[2:7])

#(b)

print(MonthDaysS[::-1])

data =
{â€™2014â€™:[100.5,150.8,200.9,30000,40000],â€™2013â€™:[12000,18000,22000,30000,45000],â€™2016â
€™:[20000,50000,
70000,100000,125000],â€™2017â€™:[50000,60000,70000,80000,90000]}

df5 = pd.DataFrame(data)
df5.index = [â€™Madhuâ€™, â€™Kusumâ€™, â€™Kinshukâ€™, â€™Ankitâ€™, â€™Shrutiâ€™]

print(df5)
```

output:-

monthno
3 31
4 30
5 31
6 30
7 31
dtype: int64
monthno
12 31
11 30
10 31
9 30
8 31

7 31
6 30
5 31
4 30
3 31
2 29
1 31
dtype: int64
2014 2013 2016 2017
Madhu 100.5 12000 20000 50000
Kusum 150.8 18000 50000 60000
Kinshuk 200.9 22000 70000 70000
Ankit 30000.0 30000 100000 80000
Shruti 40000.0 45000 125000 90000

#9

```
dinc={"2014":[100.5,150.8,200.9,30000,40000],
"2015":[12000,18000,22000,30000,45000],
"2016":[20000,50000,70000,100000,125000],
"2017":[50000,60000,70000,80000,90000],
}
df=pd.DataFrame(dinc)
df.index=["Madhu","Kusum","Kinshuk","Ankit","Shruti"]
df.index.name = â€™Sales Personsâ€™
df.columns.name = â€™Yearsâ€™
print(df)
```

output:-

Years 2014 2015 2016 2017
Sales Persons
Madhu 100.5 12000 20000 50000
Kusum 150.8 18000 50000 60000
Kinshuk 200.9 22000 70000 70000
Ankit 30000.0 30000 100000 80000
Shruti 40000.0 45000 125000 90000

#10
#a

```
print(df.index)
#b
print("\n",df.columns)

#c
print(df.columns.dtype)

#d
print("Dimension : ",df.ndim,"\tShape : ",df.shape,"\tSize : ",df.size , "\tValues : ",df.values)

#e
print(df.tail(2))
```

```
#f
print(df.loc[:,"2014":"2015"])

#g

dict1={"2018":pd.Series([160000,110000,500000,340000,900000],index=["Madhu","Kusum","Kinshuk","A
nkit","Shruti"])}
df2=pd.DataFrame(dict1)
print(df2)

#h
print(df2.empty)

======
output
======

Index([â€™Madhuâ€™, â€™Kusumâ€™, â€™Kinshukâ€™, â€™Ankitâ€™, â€™Shrutiâ€™], dtype=â€™objectâ
€™, name=â€™Sales Personsâ€™)

Index([â€™2014â€™, â€™2015â€™, â€™2016â€™, â€™2017â€™], dtype=â€™objectâ€™, name=â€™Yearsâ€™)
object
Dimension : 2 Shape : (5, 4) Size : 20 Values : [[1.005e+02 1.200e+04 2.000e+04 5.000e+04]
[1.508e+02 1.800e+04 5.000e+04 6.000e+04]
[2.009e+02 2.200e+04 7.000e+04 7.000e+04]
[3.000e+04 3.000e+04 1.000e+05 8.000e+04]
[4.000e+04 4.500e+04 1.250e+05 9.000e+04]]
Years 2014 2015 2016 2017
Sales Persons
Ankit 30000.0 30000 100000 80000
Shruti 40000.0 45000 125000 90000
Years 2014 2015
Sales Persons
Madhu 100.5 12000
Kusum 150.8 18000
Kinshuk 200.9 22000
Ankit 30000.0 30000
Shruti 40000.0 45000
2018
Madhu 160000
Kusum 110000
Kinshuk 500000
Ankit 340000
Shruti 900000
False

#11
# a
df3=pd.concat([df,df2],axis=1)
print(df3)

#b
print("\n",df.transpose())
```

```
# c

print(df["2017"])

# d
print(df3.loc[["Madhu","Ankit"],"2017":"2018"])


# e
print(df3.loc[["Shruti"],"2016"])

#f
df3.loc[â€™Sumitâ€™]=[196.2,37800,52000,78438,38852]

print(df3)

#g

df3=df3.drop(columns="2014")
print(df3)


#h
df3=df3.drop("Kinshuk")
#i

df3=df3.rename({"Ankit":"Vivaan","Madhu":"Shailesh"})
print(df3)

#j
df3.loc["Shailesh","2018"]=100000
print(df3)

# k
df3.to_csv("data.csv",index=False,columns=None)

#l
df4 = pd.read_csv("data.csv")
df4.index = ["Shailesh","Kusum","Kinshuk","Vivaan","Shruti"]
print(df4)
======
output
======
2014 2015 2016 2017 2018
Madhu 100.5 12000 20000 50000 160000
Kusum 150.8 18000 50000 60000 110000
Kinshuk 200.9 22000 70000 70000 500000
Ankit 30000.0 30000 100000 80000 340000
Shruti 40000.0 45000 125000 90000 900000

Sales Persons Madhu Kusum Kinshuk Ankit Shruti
Years
2014 100.5 150.8 200.9 30000.0 40000.0
```

2015 12000.0 18000.0 22000.0 30000.0 45000.0
2016 20000.0 50000.0 70000.0 100000.0 125000.0
2017 50000.0 60000.0 70000.0 80000.0 90000.0
Sales Persons
Madhu 50000
Kusum 60000
Kinshuk 70000
Ankit 80000
Shruti 90000
Name: 2017, dtype: int64
2017 2018
Madhu 50000 160000
Ankit 80000 340000
Shruti 125000
Name: 2016, dtype: int64
2014 2015 2016 2017 2018
Madhu 100.5 12000.0 20000.0 50000.0 160000.0
Kusum 150.8 18000.0 50000.0 60000.0 110000.0
Kinshuk 200.9 22000.0 70000.0 70000.0 500000.0
Ankit 30000.0 30000.0 100000.0 80000.0 340000.0
Shruti 40000.0 45000.0 125000.0 90000.0 900000.0
Sumit 196.2 37800.0 52000.0 78438.0 38852.0
2015 2016 2017 2018
Madhu 12000.0 20000.0 50000.0 160000.0
Kusum 18000.0 50000.0 60000.0 110000.0
Kinshuk 22000.0 70000.0 70000.0 500000.0
Ankit 30000.0 100000.0 80000.0 340000.0
Shruti 45000.0 125000.0 90000.0 900000.0
Sumit 37800.0 52000.0 78438.0 38852.0
2015 2016 2017 2018
Shailesh 12000.0 20000.0 50000.0 160000.0
Kusum 18000.0 50000.0 60000.0 110000.0
Vivaan 30000.0 100000.0 80000.0 340000.0
Shruti 45000.0 125000.0 90000.0 900000.0
Sumit 37800.0 52000.0 78438.0 38852.0
2015 2016 2017 2018
Shailesh 12000.0 20000.0 50000.0 100000.0
Kusum 18000.0 50000.0 60000.0 110000.0
Vivaan 30000.0 100000.0 80000.0 340000.0
Shruti 45000.0 125000.0 90000.0 900000.0
Sumit 37800.0 52000.0 78438.0 38852.0
2015 2016 2017 2018
Shailesh 12000.0 20000.0 50000.0 100000.0
Kusum 18000.0 50000.0 60000.0 110000.0
Kinshuk 30000.0 100000.0 80000.0 340000.0
Vivaan 45000.0 125000.0 90000.0 900000.0
Shruti 37800.0 52000.0 78438.0 38852.0

==========
=====
Pandas 2

```
#13
#(a)
Product = pd.DataFrame({â€™Itemâ€™:[â€™TVâ€™,â€™TVâ€™,â€™TVâ€™,â€™ACâ€™], â€™Companyâ
€™:[â€™LGâ€™,â€™VIDEOCONâ€™,â€™LGâ€™,â€™SONYâ€™],
â€™Rupeesâ€™:[12000, 10000, 15000, 14000], â€™USDâ€™:[700, 650, 800, 750]})
print(Product)

#(b)
Product.loc[5] = [â€™ACâ€™,â€™SONYâ€™, 32000, 2200]
print(Product)

#(c)
print(Product[Product[â€™Companyâ€™] == â€™LGâ€™].max())

#(d)
print(Product.groupby("Item").sum())

#(e)
ProductTemp = Product[Product[â€™Companyâ€™] == â€™SONYâ€™]
print(ProductTemp[â€™USDâ€™].median())

#(f)
Product_sorted_by_ruppee = Product.sort_values(â€™Rupeesâ€™, ascending = True, axis = 0)
print(Product_sorted_by_ruppee)

#(g)
engine=sqlalchemy.create_engine(â€™mysql+pymysql://root:@localhost:3306/practiceâ€™)
Product.to_sql(â€™productâ€™, if_exists = â€™appendâ€™, con = engine, index = False)

======
output
======

Item Company Rupees USD
0 TV LG 12000 700
1 TV VIDEOCON 10000 650
2 TV LG 15000 800
3 AC SONY 14000 750
Item Company Rupees USD
0 TV LG 12000 700
1 TV VIDEOCON 10000 650
2 TV LG 15000 800
3 AC SONY 14000 750
5 AC SONY 32000 2200
Company LG
Rupees 15000
USD 800
dtype: object
Company Rupees USD
Item
AC SONYSONY 46000 2950
TV LGVIDEOCONLG 37000 2150
1475.0
```

Item Company Rupees USD
1 TV VIDEOCON 10000 650
0 TV LG 12000 700
3 AC SONY 14000 750
2 TV LG 15000 800
5 AC SONY 32000 2200

```
import pandas as pd
import numpy as np
import pymysql as py
import sqlalchemy

#14
#(a)
Data = pd.DataFrame({â€™Nameâ€™:[â€™Aparnaâ€™,â€™Pankajâ€™,â€™Ramâ€™,â€™Rameshâ€™, â
€™Naveenâ€™, â€™Krrishanavâ€™, â€™Bhawnaâ€™],
â€™Degreeâ€™:[â€™MBAâ€™,â€™BCAâ€™,â€™M.Techâ€™,â€™MBAâ€™,None,â€™BCAâ€™,â€™MBAâ
€™],
â€™Scoreâ€™:[90, None, 80, 98, 97, 78, 89]})
print(Data)

#(b)
print(Data.groupby(â€™Degreeâ€™).max())

#(c)
Data[â€™Scoreâ€™].fillna(76, inplace = True)
print(Data)

#(d)
Data.set_index = â€™Nameâ€™
print(Data)

#(e)
print(Data.loc[:,[â€™Degreeâ€™,â€™Scoreâ€™]].groupby(by = â€™Degreeâ€™).mean())
print(Data.groupby([â€™Nameâ€™,â€™Degreeâ€™])[â€™Scoreâ€™].mean())

#(f)
print("Total MBA Students :",Data[Data[â€™Degreeâ€™] == â€™MBAâ€™].shape[0])

#(g)
d = (Data[Data[â€™Degreeâ€™] == â€™BCAâ€™][â€™Scoreâ€™])
print("Mode : ",d.mode()[0])
======
output
======
```

Name Degree Score
0 Aparna MBA 90.0
1 Pankaj BCA NaN
2 Ram M.Tech 80.0
3 Ramesh MBA 98.0
4 Naveen None 97.0
5 Krrishanav BCA 78.0
6 Bhawna MBA 89.0

Name Score
Degree
BCA Pankaj 78.0
M.Tech Ram 80.0
MBA Ramesh 98.0
Name Degree Score
0 Aparna MBA 90.0
1 Pankaj BCA 76.0
2 Ram M.Tech 80.0
3 Ramesh MBA 98.0
4 Naveen None 97.0
5 Krrishanav BCA 78.0
6 Bhawna MBA 89.0
Name Degree Score
0 Aparna MBA 90.0
1 Pankaj BCA 76.0
2 Ram M.Tech 80.0
3 Ramesh MBA 98.0
4 Naveen None 97.0
5 Krrishanav BCA 78.0
6 Bhawna MBA 89.0
Score
Degree
BCA 77.000000
M.Tech 80.000000
MBA 92.333333
Name Degree
Aparna MBA 90.0
Bhawna MBA 89.0
Krrishanav BCA 78.0
Pankaj BCA 76.0
Ram M.Tech 80.0
Ramesh MBA 98.0
Name: Score, dtype: float64
Total MBA Students : 3
Mode : 76.0

==========

Matplotlib

```
import matplotlib.pyplot as plt
import numpy as np

categories = [â€™Aâ€™, â€™Bâ€™, â€™Câ€™, â€™Dâ€™]
values = [3, 7, 8, 5]
x = np.arange(1, 11)
```

```
y = np.random.randint(1, 10, 10)

#Bar Chart
plt.figure(figsize = (10, 8))

plt.subplot(3, 3, 1)
plt.bar(categories, values)
plt.title(â€™Bar Chartâ€™)

#Scatter Plot
plt.subplot(3, 3, 2)
plt.scatter(x, y)
plt.title("Scatter Chart")

#Line Chart
plt.subplot(3, 3, 3)
plt.plot(x, y)
plt.title("Line Chart")

#Area Chart
plt.subplot(3, 3, 4)
plt.fill_between(x, y, color = "skyblue", alpha=0.5)
plt.title("Area Chart")

#Pie Chart
plt.subplot(3, 3, 5)
plt.pie(values, labels=categories, autopct=â€™%1.1f%%â€™)
plt.title("Pie Chart")

#Histogram
plt.subplot(3, 3, 6)
plt.hist(y, bins=5)
plt.title(â€™Histogramâ€™)

#Cumulative Distribution (Ogive)
plt.subplot(3, 3, 7)
plt.hist(y, bins=5, cumulative=True, histtype=â€™stepâ€™, color=â€™redâ€™)
plt.title(â€™Ogiveâ€™)

#Dot Plot
plt.subplot(3, 3, 8)
plt.plot(x, y, â€™gpâ€™, markersize=10)
plt.title(â€™Dot Plotâ€™)

# Stem-and-Leaf Display
plt.subplot(3, 3, 9)
plt.stem(x, y,)
plt.title(â€™Stem-and-Leafâ€™)

plt.tight_layout()
plt.show()

================================================================================
============================================================
```

Linear Regression

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
import time

def linear(X, b0, b1):
return [b0+b1*x for x in X]

#b0 - Intercept

def intercept(X, Y, b1):
x_ = np.mean(X)
y_ = np.mean(Y)

return y_ - b1 * x_

#b1 - slope

def slope(X, Y):
x_ = np.mean(X)
y_ = np.mean(Y)

rise = sum([(x-x_) * (y-y_) for x,y in zip(X,Y)])
run = sum([(x-x_)**2 for x in X])

return rise / run

data = pd.read_csv(â€™Automobile_data.csvâ€™)
data.head()

data = data.loc[data[â€™priceâ€™] != â€™?â€™]
data[[â€™priceâ€™]] = data[[â€™priceâ€™]].astype(int)

for i in data.columns:
print(i)
data = data.loc[data[i] != â€™?â€™]

final_data = data[[â€™engine-sizeâ€™, â€™priceâ€™]]

predictor = data[â€™engine-sizeâ€™]
target = data[â€™priceâ€™]

plt.figure(figsize=(8,5))
plt.title("Price vs engine-size")
plt.scatter(predictor, target, color = "#247ba0")
plt.xlabel(â€™engine-sizeâ€™)
plt.ylabel(â€™priceâ€™)

b1 = slope(predictor, target)
```

```
b0 = intercept(predictor, target, b1)
predicted = linear(predictor, b0, b1)
print(predicted)

plt.figure(figsize = (8, 5))
plt.plot(predictor, predicted, color = â€™#f25f5câ€™)
plt.scatter(predictor, predicted, color = â€™#f25f5câ€™)
plt.title(â€™Predicted values bt Linear Regressionâ€™, fontsize = 15)
plt.xlabel(â€™engine-sizeâ€™)
plt.ylabel(â€™priceâ€™)
plt.scatter(predictor, target, color = â€™#247ba0â€™)
plt.show()

================================================================================
============================================================

Pre-processing and Laptop-price prediction

import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler, StandardScaler, OneHotEncoder

from sklearn.preprocessing import LabelEncoder

from scipy import stats

df = pd.read_csv(â€™laptop_price---dataset.csvâ€™)

print(df.head())
print(df.info())

print(df.isnull().sum())

print(df.drop_duplicates())

df[â€™Company_Labelâ€™] = LabelEncoder().fit_transform(df[[â€™Companyâ€™]])
df.head()

df[â€™Companyâ€™] = df[â€™Companyâ€™].str.lower()
df

z = np.abs(stats.zscore(df[â€™Price (Euro)â€™]))
print(z)

threshold_z = 2

outlier_indices = np.where(z > threshold_z)[0]
no_outliers = df.drop(outlier_indices)
print("Original DataFrame Shape:", df.shape)
print("DataFrame Shape after Removing Outliers:", no_outliers.shape)

import seaborn as sns
import matplotlib.pyplot as plt
```

```
corr = no_outliers.corr(numeric_only = True)

sns.heatmap(corr, annot = True)

df = no_outliers

scaler = MinMaxScaler()

df[â€™Price (Euro)_MinMaxScaledâ€™] = scaler.fit_transform(np.array(df[â€™Price (Euro)â€™]).reshape(-1,1))
print(df[â€™Price (Euro)_MinMaxScaledâ€™].head())

scaler = StandardScaler()#Data is in range of

df[â€™Price (Euro)_StandardScalerâ€™] = scaler.fit_transform(np.array(df[â€™Price (Euro)â€™]).reshape(-1,1))
print(df[â€™Price(Euro)_StandardScalerâ€™].head())

df[â€™Price (Euro)_StandardScalerâ€™].min()

==========
Prediction
==========

from sklearn.model_selection import train_test_split

X = df[â€™RAM (GB)â€™]
Y = df[â€™Price (Euro)â€™]X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.33,
random_state=42)
print(X_train, X_test, Y_train, Y_test)

from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(np.array(X_train).reshape(-1,1), Y_train)
lr.intercept_
lr.coef_

from sklearn.metrics import mean_squared_error, r2_score
y_pred = lr.predict(np.array(X_test).reshape(-1,1))
print(â€™MeanSquaredErrorâ€™, np.sqrt(mean_squared_error(Y_test,y_pred)))
print(â€™R2 Scoreâ€™, r2_score(Y_test, y_pred))

================================================================================
==================================================

USA Housing


import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sns

dfHouse = pd.read_csv("./USA_Housing.csv")
dfHouse.head()
```

```
dfHouse.info()

dfHouse.drop_duplicates(inplace=True)
dfHouse.isnull().sum()


import pandas as pd
import matplotlib.pyplot as plt

def PrintChart(dfHouse):
float_cols = dfHouse.select_dtypes(include=[â€™float64â€™]).columns
fig, axes = plt.subplots(len(float_cols), 1, figsize=(10, 5 * (len(float_cols))))

# Box plots for each float64 column
for idx, col in enumerate(float_cols):
axes[idx].boxplot(dfHouse[col])
axes[idx].set_title(f"Box Plot of {col}")
axes[idx].set_ylabel(f"Range of {col}")
axes[idx].set_xlabel(col)

# Adjust layout
plt.tight_layout()
plt.show()
PrintChart(dfHouse)

# Convert it into ranges
chartDfHouse = dfHouse
for i in chartDfHouse.columns:
if chartDfHouse[i].dtype == â€™float64â€™:
q1 = np.percentile(chartDfHouse[i], 25)
q3 = np.percentile(chartDfHouse[i], 75)
iqr = q3 - q1
upperBound = q3 + 1.5 * iqr
lowerBound = q1 - 1.5 * iqr
chartDfHouse = chartDfHouse[(chartDfHouse[i] > lowerBound) & (chartDfHouse[i] < upperBound)]
PrintChart(chartDfHouse)


numericChartDfHouse = chartDfHouse.select_dtypes(include=â€™numberâ€™).corr()
# Create a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(numericChartDfHouse, cmap=â€™coolwarmâ€™, linewidths=.5)
plt.title(â€™Sample Heatmapâ€™)
plt.show()
```