Lab 1

```
#1. Install Numpy
import numpy as np

#2. Check the Numpy version installed
print(np.__version__)

1.23.4

#3. Create 1-D Array in numpy:
np_arr1=np.array([1,2,3,4])

#4. Use list to create 1D array (you may also specify data type i.e. dtype=’int16’)
lis=[5,4,3,2,1]
np_arr2=np.array(lis,dtype="int16")
type(np_arr2.dtype)

numpy.dtype[int16]

#5. User tuple to create 1D array
tup=(1,2,3,4,5,6)

#6. Use arange function to create 1D array of int
np_arr2=np.array(tup,dtype="int16")
type(np_arr2.dtype)

numpy.dtype[int16]

#7. Use arange function to create 1D array of float
#dtype = symbols(int->'i', uint->'u',float->'f',double->'d',complex->'D',bool->'b'
np_arr3=np.arange(10,dtype='f')
np_arr3

array([0., 1., 2., 3., 4., 5., 6., 7., 8., 9.], dtype=float32)

#8. Create 1D array of mixed elements int and float, and print the array and see the output
np_arr1=np.array([1,2.3,6,3.1,5,4.33])
print(np_arr1)

[1.   2.3  6.   3.1  5.   4.33]

#9. Create 1D array of mixed elements int, float, and str, then print the array and see the
output
np_arr1=np.array(['asf',2.3,6,'l',5,4.33])
print(np_arr1)

['asf' '2.3' '6' 'l' '5' '4.33']

#10. Create a 2D array of dimensions 2x2
np_arr1=np.array([[1,2],[3,4]])
print(np_arr1)

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

```
#11. Print the shape, size, and memory used by this array in bytes (use itemsize, or nbytes)
print("Shape : ",np_arr1.shape)
print("Size : ",np_arr1.size)
print("ItemSize : ",np_arr1.itemsize)
print("Total Memory : ",np_arr1.size*np_arr1.itemsize)
```

```
Shape :  (2, 2)
Size :  4
ItemSize :  8
Total Memory :  32
```

```
#12. Check the type of any array variable
print(type(np_arr1))
print(type(np_arr1[1][1]))
```

```
<class 'numpy.ndarray'>
<class 'numpy.int64'>
```

```
#13. Check indexing on array with help of examples
np_arr1[1][1]
```

```
4
```

```
#14. Using arange function create an 3D array of dimensions = (2,3,4) , first element of
this array is 0 and last element is 23 in increasing order, store this array in a variable
b.
b=np.arange(24).reshape(2,3,4)
print(b)
```

```
[[[ 0  1  2  3]
  [ 4  5  6  7]
  [ 8  9 10 11]]

 [[12 13 14 15]
  [16 17 18 19]
  [20 21 22 23]]]
```

```
#15. What index can produce output:
#    array([[ 0, 1, 2, 3],
#           [ 4, 5, 6, 7],
#           [ 8, 9, 10, 11]])
print(b[0])
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
```

```
#16. What index can produce output: 0
print(b[0][0][0])
```

```
0
```

```
#17. What index can produce output: array([4, 5, 6, 7])
b[0][1]
```

```
array([4, 5, 6, 7])
```

```
#18. What index can produce output: array([0,12])
b[:,0,0]

array([ 0, 12])

#19. What index can produce output: array([4,6])
b[:1,1,::2]

array([[4, 6]])

#20. Check the output of b[… , 1]
b[..., 1]

array([[ 1,  5,  9],
       [13, 17, 21]])

#21. What index can produce output: array( [1, 5, 9] )
b[0,:,1]

array([1, 5, 9])

#22. What index can produce output: array([3,7,11])
b[0,:,3]

array([ 3,  7, 11])

#23. What index can produce output: array([11, 7,3])
b[0,-3:,3]

array([ 3,  7, 11])

#24. What index can produce output: array([3,11])
b[0,::2,3]

array([ 3, 11])
```

Lab 2

```
import pandas as pd
import numpy as np

#1. Create a simple Pandas Series from a list.
lis=[10,20,30,40,50]
sr=pd.Series(lis)
print(sr)

0    10
1    20
2    30
3    40
4    50
dtype: int64

#2. Return the first and last values of the Series created above
sr[0]
sr[4]

50

#3. Create a simple Pandas Series with your own labels i.e. index
label=['a','b','c','d']
df=pd.DataFrame([1,2,3,4],label)
print(df)

   0
a  1
b  2
c  3
d  4

#4. Access the values using your own index and print the value, also try –ve
index.
print(df[0][1])
print(df[0][-1])

2
4

#5. Create a simple Pandas Series from a dictionary
dic1={'a':23,'b':12,'c':18,'d':25,'e':18,'f':20,'g':12}
sr=pd.Series(dic1)
print(sr)

a    23
b    12
c    18
```

```
d    25
e    18
f    20
g    12
dtype: int64
```

```
#6. Create a Series using only calories intake data from user defined indexes
"day1","day2", and "day3".
a,b,c=input().split()
dic2={'day1':int(a),'day2':int(b),'day3':int(c)}
sr1=pd.Series(dic2)
print(sr1)
```

```
455 656 459
day1    455
day2    656
day3    459
dtype: int64
```

```
#7. Create a Series of heterogeneous data types and check the data type of
the Series as well as individual items.
dic2={'d':456,2:'hello',6.9:34}
sr2=pd.Series(dic2)
print(sr)
print(type(sr2[2]))
print(type(sr2['d']))
print(type(sr2))
```

```
a    23
b    12
c    18
d    25
e    18
f    20
g    12
dtype: int64
<class 'str'>
<class 'int'>
<class 'pandas.core.series.Series'>
```

```
#8. Compute min, max, mean values of a Series
print(sr.min())
print(sr.max())
print(sr.mean())
```

```
12
25
18.285714285714285
```

```
#14.Sort the values of a Series in ascending and descending order and print
print(sr.sort_values(),'\n')
print(sr.sort_values(ascending=False))
```

```
b    12
g    12
c    18
e    18
f    20
a    23
d    25
dtype: int64

d    25
a    23
f    20
c    18
e    18
b    12
g    12
dtype: int64

#15.Print the number of occurrences of unique values in a series. (use
value_counts)
sr.value_counts()

12    2
18    2
23    1
25    1
20    1
dtype: int64

#16.Create a Series of 10 integers, and later change its dtype to be float
(use astype).
sr3=pd.Series(range(10))
sr3=sr3.astype(float)
print(sr3)

0    0.0
1    1.0
2    2.0
3    3.0
4    4.0
5    5.0
6    6.0
7    7.0
8    8.0
9    9.0
dtype: float64

#17.Convert the Series you created above to numpy array (use to_numpy(), or
array )
nparr=np.array(sr3)
print(nparr)
print(type(nparr))
```

```
[0. 1. 2. 3. 4. 5. 6. 7. 8. 9.]
<class 'numpy.ndarray'>
```

```
#18.Delete an item from Series using single index.
sr3.drop(9)
```

```
0    0.0
1    1.0
2    2.0
3    3.0
4    4.0
5    5.0
6    6.0
7    7.0
8    8.0
dtype: float64
```

```
#19.Find the number of items in a series. (use len or count)
print(len(sr3))
print(sr3.count())
```

```
10
10
```

```
#20. Append Series by assigning a value to a new index. (S[n]=v)
sr3[9]=33
sr3
```

```
0     0.0
1     1.0
2     2.0
3     3.0
4     4.0
5     5.0
6     6.0
7     7.0
8     8.0
9    33.0
dtype: float64
```

```
#21.Check if a value is present in a Series. (use type cast to a set or check
in values)
print(3 in sr3)
print(53 in sr3)
```

```
True
False
```

```
#22. Print the index of a Series and also if all indexes appear only once.
(use is_unique)
pd.Index(sr2).is_unique
```

```
True
```

```
#23. Create two Series one with default index, other with index like
‘a’,’b’,’c’,’d’, etc. then access both the Series based on label and position
s1=pd.Series([0,1,2,3,4])
s2=pd.Series(range(0,5), index=['a','b','c','d','e'])

print(s1)
print(s2)

0    0
1    1
2    2
3    3
4    4
dtype: int64
a    0
b    1
c    2
d    3
e    4
dtype: int64

#24. Try function at and iat on above problem and observe the difference in
output with respect to loc and iloc.
print(sr2.iat[2])
print(sr2.at[2])

34
hello
```

Lab 3

```
#Ex:1
import pandas as pd
import numpy as np
df=pd.DataFrame([1,3,5,12,6,8],[10,11,12,20,50,8])
print(df)

     0
10   1
11   3
12   5
20  12
50   6
8    8

#Ex:2
df=pd.DataFrame({'A':[1,3,5,12,6,8],'B':[10,11,12,20,50,8]},index=[0,1,2,3,4,5])
print(df)

    A   B
0   1  10
1   3  11
2   5  12
3  12  20
4   6  50
5   8   8

#1.b. Create a dataframe which looks like the output shown below.
df=pd.DataFrame({'a':[1,2,8,4],'b':[5,6,9,8],'c':[11,12,30,14]},index=[0,1,2,3])
print(df)

   a  b   c
0  1  5  11
1  2  6  12
2  8  9  30
3  4  8  14

#1.b. Create a dataframe which looks like the output shown below.
df=pd.DataFrame({'X':[78,85,96,80,86],'Y':[84,94,89,83,86],'Z':
[86,97,96,72,83]},index=[0,1,2,3,4])
print(df)

    X   Y   Z
0  78  84  86
1  85  94  97
2  96  89  96
3  80  83  72
4  86  86  83

#2. Create and display a DataFrame from a specified dictionary data which has the index
labels.:
exam_data = {'name': ['Anastasia', 'Dima', 'Katherine', 'James', 'Emily', 'Michael',
'Matthew', 'Laura', 'Kevin', 'Jonas'],
             'score': [12.5, 9, 16.5, np.nan, 9, 20, 14.5, np.nan, 8, 19],
             'attempts': [1, 3, 2, 3, 2, 3, 1, 1, 2, 1],
             'qualify': ['yes', 'no', 'yes', 'no', 'no', 'yes', 'yes', 'no', 'no', 'yes']}
labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
df=pd.DataFrame(exam_data,labels)
```

```
print(df)

        name  score  attempts qualify
a  Anastasia   12.5         1     yes
b       Dima    9.0         3      no
c  Katherine   16.5         2     yes
d      James    NaN         3      no
e      Emily    9.0         2      no
f    Michael   20.0         3     yes
g    Matthew   14.5         1     yes
h      Laura    NaN         1      no
i      Kevin    8.0         2      no
j      Jonas   19.0         1     yes
```

```
#3. Write a python script to display a summary of the basic information about a specified
DataFrame and its data. Sample Python dictionary data and list labels:
df.info()

<class 'pandas.core.frame.DataFrame'>
Index: 10 entries, a to j
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   name      10 non-null     object
 1   score     8 non-null      float64
 2   attempts  10 non-null     int64
 3   qualify   10 non-null     object
dtypes: float64(1), int64(1), object(2)
memory usage: 400.0+ bytes
```

```
#4. Write a python script to get the first 3 rows of a given DataFrame.
print(df.head(3))

        name  score  attempts qualify
a  Anastasia   12.5         1     yes
b       Dima    9.0         3      no
c  Katherine   16.5         2     yes
```

```
#5. Write a python script to select the 'name' and 'score' columns from the following
DataFrame. Sample Python dictionary data and list labels:
print(df.iloc[:,[1,3]])

  score qualify
a  12.5     yes
b   9.0      no
c  16.5     yes
d  11.5      no
e   9.0      no
f  20.0     yes
g  14.5     yes
h   NaN      no
i   8.0      no
j  19.0     yes
```

```
#8. Write a python script to count the number of rows and columns of a DataFrame. Sample
Python dictionary data and list labels:
print(len(df))
print(len(df.columns))

10
```

```
4
```

```
#9. Write a python script to select the rows where the score is missing, i.e. is NaN.
print(df[df['score'].isna()])
```

	name	score	attempts	qualify
d	James	NaN	3	no
h	Laura	NaN	1	no

```
#10. Write a python script to select the rows the score is between 15 and 20 (inclusive).
print(df[df['score'].between(15,20)])
```

```
        name score attempts qualify
c  Katherine  16.5        2     yes
f    Michael  20.0        3     yes
j      Jonas  19.0        1     yes
```

```
#11. Write a python script to select the rows where number of attempts in the examination is
less than 2 and score greater than 15.
print(df[(df['score']>15) * (df['attempts']<2)])
```

```
    name score attempts qualify
j  Jonas  19.0        1     yes
```

```
#12. Write a python script to change the score in row 'd' to 11.5.
df.loc[['d'],['score']]=11.5
```

```
#13. Write a python script to calculate the sum of the examination attempts by the students.
df['attempts'].sum()
```

```
19
```

```
#14. Write a python script to calculate the mean score for each different student in
DataFrame.
df['score'].mean()
```

```
13.333333333333334
```

```
#15. Write a python script to append a new row 'k' to data frame with given values for each
column. Now delete the new row and return the original DataFrame.
df.loc['k'] = [1, 'Suresh', 'yes', 15.5]
df = df.drop('k')
print(df)
```

```
        name score attempts qualify
a  Anastasia  12.5        1     yes
b       Dima   9.0        3      no
c  Katherine  16.5        2     yes
d      James  11.5        3      no
e      Emily   9.0        2      no
f    Michael  20.0        3     yes
g    Matthew  14.5        1     yes
h      Laura   NaN        1      no
i      Kevin   8.0        2      no
j      Jonas  19.0        1     yes
```

#16.a. Write a python script to sort the DataFrame first by 'name' in descending order.

```
print(df.sort_values(by='name', ascending=False))
```

```
        name  score  attempts  qualify
f    Michael   20.0         3      yes
g    Matthew   14.5         1      yes
h      Laura    NaN         1       no
i      Kevin    8.0         2       no
c  Katherine   16.5         2      yes
j      Jonas   19.0         1      yes
d      James   11.5         3       no
e      Emily    9.0         2       no
b       Dima    9.0         3       no
a  Anastasia   12.5         1      yes
```

#16.b. Write a python script to sort the DataFrame first by 'qualify' in descending order

```
print(df.sort_values(by= 'qualify', ascending=False))
```

```
        name  score  attempts  qualify
a  Anastasia   12.5         1      yes
c  Katherine   16.5         2      yes
f    Michael   20.0         3      yes
g    Matthew   14.5         1      yes
j      Jonas   19.0         1      yes
b       Dima    9.0         3       no
d      James   11.5         3       no
e      Emily    9.0         2       no
h      Laura    NaN         1       no
i      Kevin    8.0         2       no
```

Lab 4

```
import pandas as pd
import numpy as np
from sklearn.datasets import fetch_openml

#1
df=pd.read_csv("housing.csv")

#2
print(df.info())
print(df.describe())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  float64
 9   ocean_proximity     20640 non-null  object 
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
None
          longitude      latitude  housing_median_age   total_rooms  \
count  20640.000000  20640.000000        20640.000000  20640.000000   
mean    -119.569704     35.631861           28.639486   2635.763081   
std        2.003532      2.135952           12.585558   2181.615252   
min     -124.350000     32.540000            1.000000      2.000000   
25%     -121.800000     33.930000           18.000000   1447.750000   
50%     -118.490000     34.260000           29.000000   2127.000000   
75%     -118.010000     37.710000           37.000000   3148.000000   
max     -114.310000     41.950000           52.000000  39320.000000   

       total_bedrooms    population    households  median_income  \
count    20433.000000  20640.000000  20640.000000   20640.000000   
mean       537.870553   1425.476744    499.539680       3.870671   
std        421.385070   1132.462122    382.329753       1.899822   
min          1.000000      3.000000      1.000000       0.499900   
25%        296.000000    787.000000    280.000000       2.563400   
50%        435.000000   1166.000000    409.000000       3.534800   
75%        647.000000   1725.000000    605.000000       4.743250   
max       6445.000000  35682.000000   6082.000000      15.000100   

       median_house_value  
count        20640.000000  
mean        206855.816909  
std         115395.615874  
min          14999.000000  
25%         119600.000000  
50%         179700.000000  
75%         264725.000000  
max         500001.000000  
```

```
#3
print("Number of Rows : ",len(df))
print("Number of Columns  : ",len(df.columns))
df.shape
```

```
Number of Rows :  20640
Number of Columns  :  10

(20640, 10)
```

```
#4
y=df["median_house_value"]
```

```
#5
print(df.head(5))
print(df.iloc[:5,:])
```

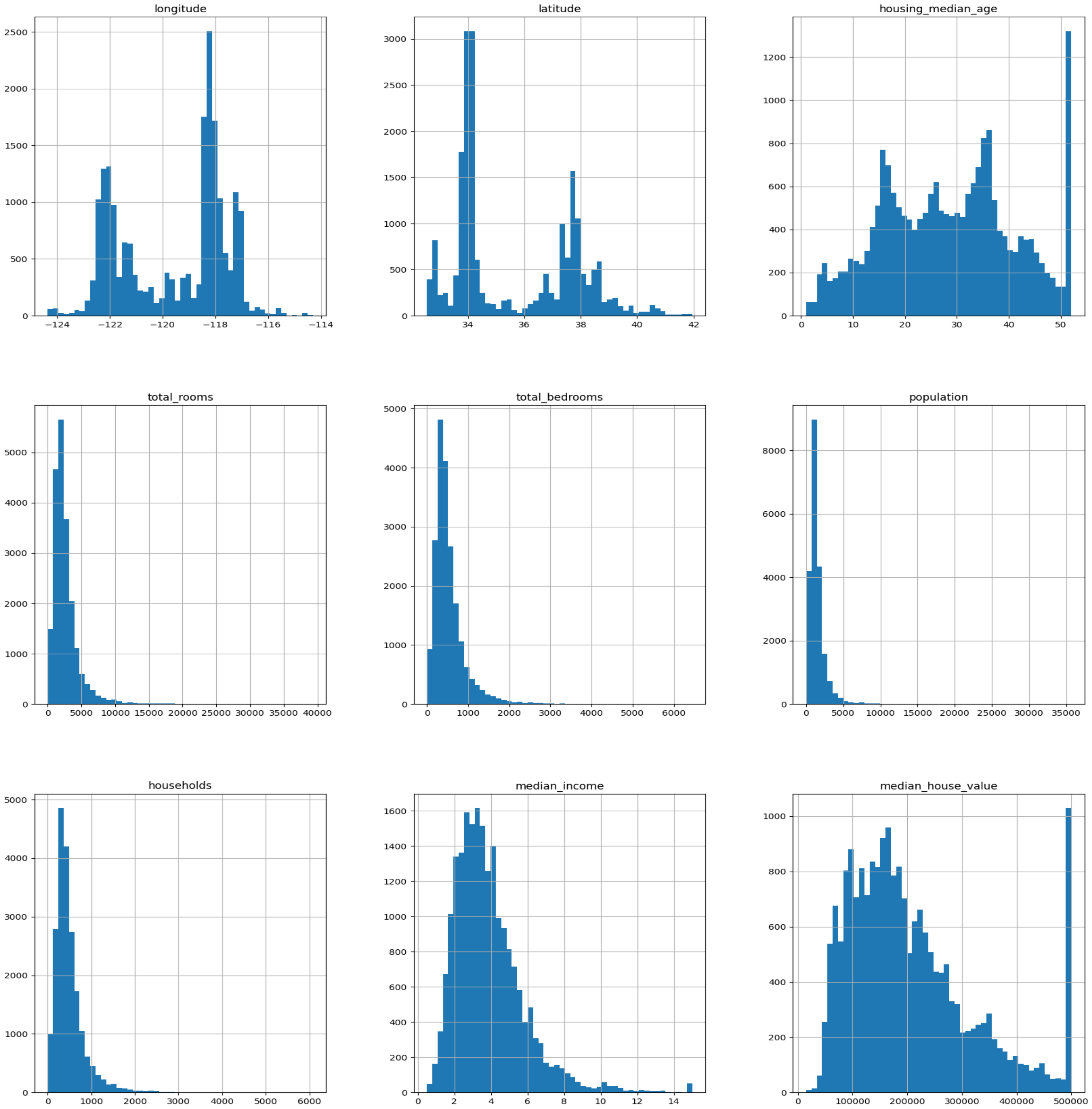
```
   longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
0    -122.23     37.88                41.0        880.0           129.0
1    -122.22     37.86                21.0       7099.0          1106.0
2    -122.24     37.85                52.0       1467.0           190.0
3    -122.25     37.85                52.0       1274.0           235.0
4    -122.25     37.85                52.0       1627.0           280.0

   population  households  median_income  median_house_value ocean_proximity
0       322.0       126.0         8.3252            452600.0        NEAR BAY
1      2401.0      1138.0         8.3014            358500.0        NEAR BAY
2       496.0       177.0         7.2574            352100.0        NEAR BAY
3       558.0       219.0         5.6431            341300.0        NEAR BAY
4       565.0       259.0         3.8462            342200.0        NEAR BAY
   longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
0    -122.23     37.88                41.0        880.0           129.0
1    -122.22     37.86                21.0       7099.0          1106.0
2    -122.24     37.85                52.0       1467.0           190.0
3    -122.25     37.85                52.0       1274.0           235.0
4    -122.25     37.85                52.0       1627.0           280.0

   population  households  median_income  median_house_value ocean_proximity
0       322.0       126.0         8.3252            452600.0        NEAR BAY
1      2401.0      1138.0         8.3014            358500.0        NEAR BAY
2       496.0       177.0         7.2574            352100.0        NEAR BAY
3       558.0       219.0         5.6431            341300.0        NEAR BAY
4       565.0       259.0         3.8462            342200.0        NEAR BAY
```

```
#6
df.describe()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	m
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000	20640.000000	20
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744	499.539680	3.8
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462122	382.329753	1.8
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1.000000	0.4
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.000000	280.000000	2.5
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000	409.000000	3.5
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000000	605.000000	4.7

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	m
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	6082.000000	15

```
#6
import matplotlib.pyplot as plt
df.hist(bins=50,figsize=(20,25))
plt.show()
```



```
#8
n_val=""
df.isna().sum()
```

```
longitude               0
latitude                0
housing_median_age      0
total_rooms             0
total_bedrooms        207
population              0
households              0
median_income           0
median_house_value      0
ocean_proximity         0
dtype: int64
```

```
df.value_counts("ocean_proximity")
```

```
ocean_proximity
<1H OCEAN     9136
INLAND        6551
NEAR OCEAN    2658
NEAR BAY      2290
ISLAND           5
dtype: int64
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  float64
 9   ocean_proximity     20640 non-null  object 
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

```
#10
df["total_bedrooms"]=df["total_bedrooms"].fillna(df["total_bedrooms"].mode())
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  float64
 9   ocean_proximity     20640 non-null  object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

```
#11
df["total_bedrooms"].sum()
```

```
10990309.0
```

```
#12
pd.cut(df["median_income"],bins=[0., 1.5, 3.0, 4.5, 6., np.inf],labels=[1, 2,
3, 4, 5])
```

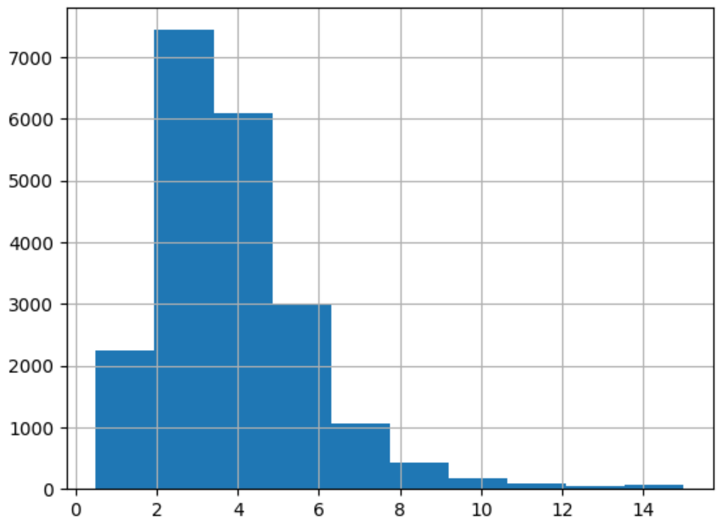
```
0        5
1        5
2        5
3        4
4        3
        ..
20635    2
20636    2
20637    2
20638    2
20639    2
Name: median_income, Length: 20640, dtype: category
Categories (5, int64): [1 < 2 < 3 < 4 < 5]
```

```
#13
df.value_counts("median_income")
```

```
median_income
15.0001    49
3.1250     49
2.8750     46
4.1250     44
2.6250     44
           ..
3.2010      1
3.2015      1
3.2016      1
3.2021      1
3.7569      1
Length: 12928, dtype: int64
```

```
#14
df["median_income"].hist()
```

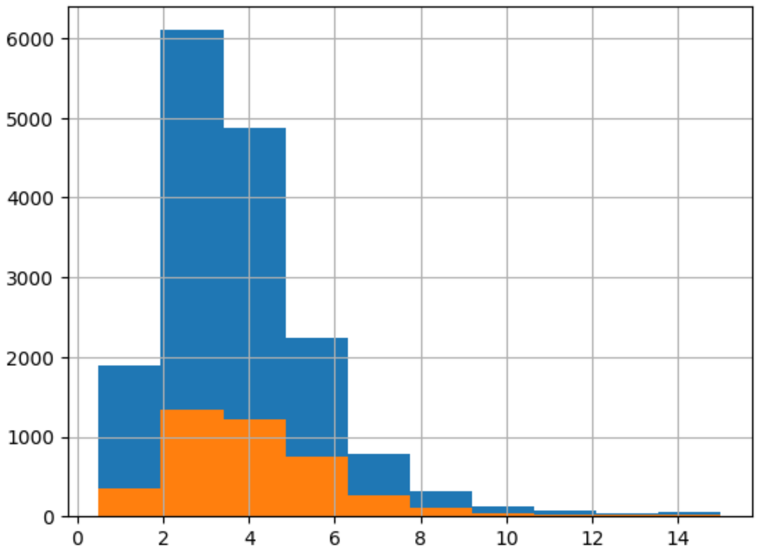
```
<Axes: >
```



```
#15
temp_train,temp_test=df[:][:int((80/100)*len(df))],df[:]
[int((80/100)*len(df)):]
```

```
#16
temp_train["median_income"].hist()
temp_test["median_income"].hist()
```

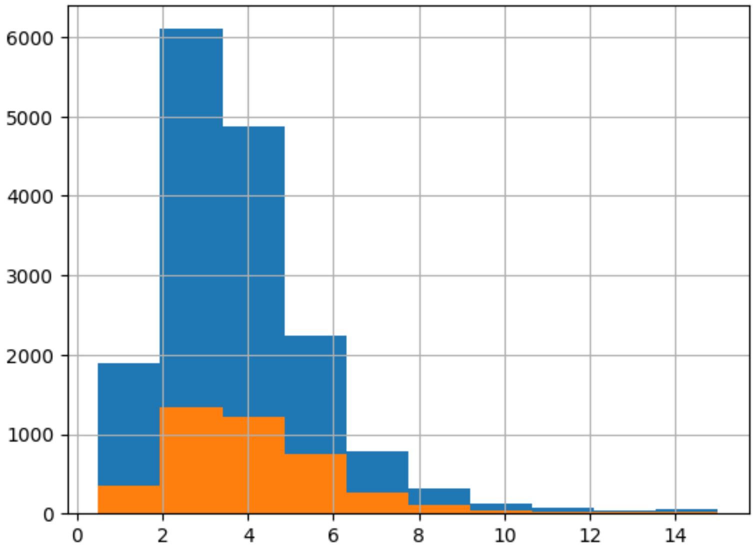
```
<Axes: >
```



```
#17
from sklearn.model_selection import StratifiedShuffleSplit
split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
#for train_index, test_index in split.split(df, df["median_income"]):
#    train = df.loc[train_index]
#    test = df.loc[test_index]
```

```
#18
temp_train["median_income"].hist()
temp_test["median_income"].hist()
```

```
<Axes: >
```



```
#19
correlation=df.corr()
correlation["median_income"].sort_values()
```

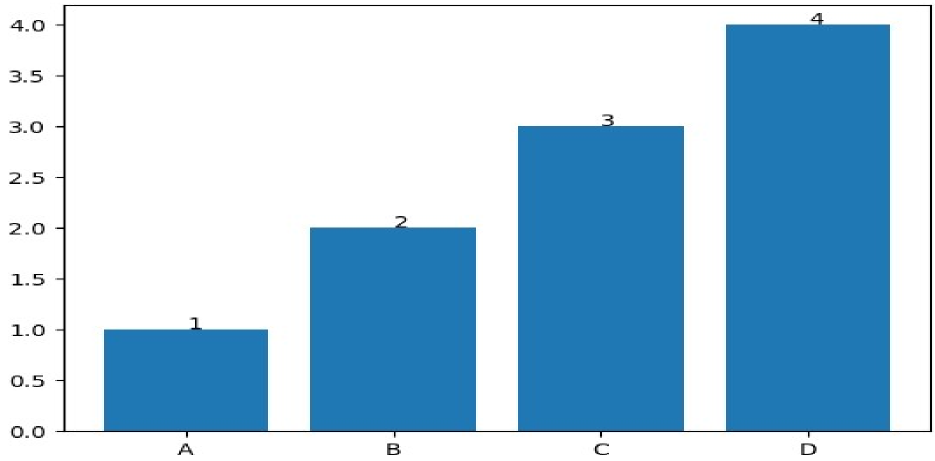
```
/tmp/ipykernel_11888/2923073434.py:2: FutureWarning: The default value of numeric_only in
DataFrame.corr is deprecated. In a future version, it will default to False. Select only
valid columns or specify the value of numeric_only to silence this warning.
  correlation=df.corr()
```

```
housing_median_age   -0.119034
latitude             -0.079809
longitude            -0.015176
total_bedrooms       -0.007723
population            0.004834
households            0.013033
total_rooms           0.198050
median_house_value    0.688075
median_income         1.000000
Name: median_income, dtype: float64
```

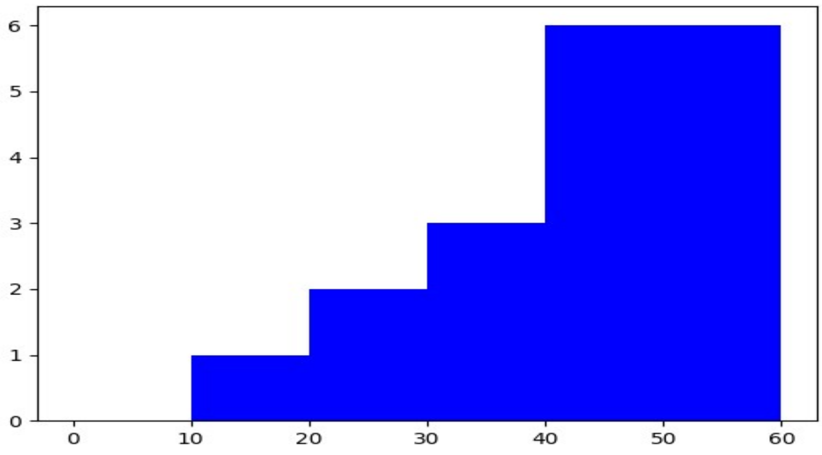
```
#20
from sklearn.preprocessing import OrdinalEncoder
from numpy import asarray
oe=OrdinalEncoder()
#df_cat_oe =oe.fit_transform(df["median_income"])
df_cat_oe=oe.fit_transform(asarray(df['median_income']).reshape(-1,1))
print(df_cat_oe)
[[12416.]
 [12411.]
 [11941.]
 ...
 [ 1037.]
 [ 1405.]
 [ 2752.]]
```

Lab 5

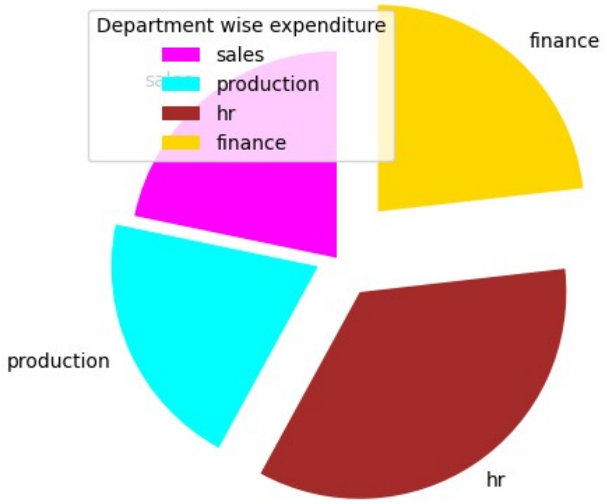
```
#1.
import matplotlib.pyplot as plt
import pandas as pd
def test_value_on_bar_top():
    x=["A","B","C","D"]
    y=[1,2,3,4]
    plt.bar(x,y)
    for index,value in enumerate(y):
        plt.text(index,value,str(value))
    plt.show()
test_value_on_bar_top()
```



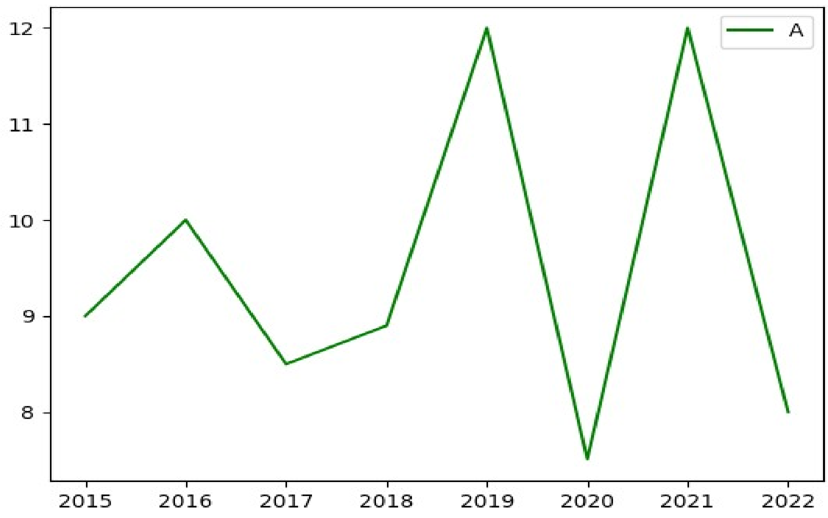
```
#2. Histplot
import matplotlib.pyplot as plt
import pandas as pd
emp_ages=[22,45,30,59,58,56,57,58,41,45,43,43,50,40,34,33,25,19]
bins=[0,10,20,30,40,50,60]
plt.hist(emp_ages,bins,rwidth=0.8,color="blue")
plt.show()
```



```
#3 Piechart
import matplotlib.pyplot as plt
import pandas as pd
slices=[15,14,24,16]
dept_name=['sales','production','hr','finance']
colors=['magenta','cyan','brown','gold']
plt.pie(slices,labels=dept_name,colors=colors,startangle=90,explode=(0,0.1,0.
2,0.3))
plt.legend(title="Department wise expenditure")
plt.show()
```



```
#4
import matplotlib.pyplot as plt
import pandas as pd
x=[2015,2016,2017,2018,2019,2020,2021,2022]
y=[9,10,8.5,8.9,12,7.51,12,8]
plt.plot(x,y,color="green",label="Profit")
plt.title="Company sales"
plt.xlabel="years"
plt.ylabel="Profit"
plt.legend("A")
                                                          plt.show()
```


A
12
11
10
9
8
2015
2016
2017
2018
2019
2020
2021
2022