```
import numpy as np
x=np.array([[1,2,3,4],
 [5,6,7,8],
 [9,10,11,12]])
print (x)
```

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
```

```
y=np.array([1,2,3,4,5,6])
print(y)
```

```
[1 2 3 4 5 6]
```

```
print(x[0][2],x[1][1],x[2][1])
```

```
3 6 10
```

```
print("first three array")
print(y[0:3])
print ("last three element")
print(y[3:])
```

```
first three array
[1 2 3]
last three element
[4 5 6]
```

```
print("\nArray x:")
print("Size:", x.size)
print("Number of elements:",x.size)
print("Number of dimensions:", x.ndim)
print("\nArray y:")
print("Size:", y.size)
print("Number of elements:",y.size)
print("Number of dimensions:", y.ndim)
```

```

Array x:
Size: 12
Number of elements: 12
Number of dimensions: 2

Array y:
Size: 6
Number of elements: 6
Number of dimensions: 1
```

```
x10=x+10
print(x10)
```

```
[[11 12 13 14]
 [15 16 17 18]
 [19 20 21 22]]
```

```
print("\n Array x:")
print("Average",np.mean(x))
print("Variance",np.var(x))
print("Standard Deviation",np.std(x))

print("\n Array y:")
print("Average",np.mean(y))
print("Variance",np.var(y))
print("Standard Deviation",np.std(y))
```

```
 Array x:
Average 6.5
Variance 11.916666666666666
Standard Deviation 3.452052529534663

 Array y:
Average 3.5
Variance 2.9166666666666665
Standard Deviation 1.707825127659933
```

```
x_reshaped = x.reshape(2, 6)
print("\nReshaped x (2x6):", x_reshaped)
```

```
Reshaped x (2x6): [[ 1  2  3  4  5  6]
 [ 7  8  9 10 11 12]]
```

```
x_transpose=x.T
print("\nTransposed x:")
print(x_transpose)
y_transpose =y.T
print("\nTransposed y:")
print(y_transpose)
```

```
Transposed x:
[[ 1  5  9]
 [ 2  6 10]
 [ 3  7 11]
 [ 4  8 12]]

Transposed y:
[1 2 3 4 5 6]
```

```
flatten_x = x.flatten()
print("\nFlattened x:")
print(flatten_x)
```

```
Flattened x:
[ 1  2  3  4  5  6  7  8  9 10 11 12]
```

```
P = np.array([[1, 3, 6],
              [1, 4, 5],
              [2, 2, 7]])
print(P)
```

```
[[1 3 6]
 [1 4 5]
 [2 2 7]]
```

```
rank_P = np.linalg.matrix_rank(P)
print("Rank of P:", rank_P)
determinant_P = np.linalg.det(P)
print("Determinant of P:", determinant_P)
diagonal_P = np.diag(P)
print("Diagonal of P:", diagonal_P)
trace_P = np.trace(P)
print("Trace of P:", trace_P)
```

```
Rank of P: 3
Determinant of P: -8.999999999999998
Diagonal of P: [1 4 7]
Trace of P: 12
```

```
eigen_values, eigen_vectors = np.linalg.eig(P)
print("\nEigenvalues:")
print(eigen_values)
print("\nEigenvectors:")
print(eigen_vectors)
```

```
Eigenvalues:
[10.59900683 -0.45701438  1.85800755]

Eigenvectors:
[[-0.55902488 -0.95323847 -0.18562955]
 [-0.55289125 -0.10432388 -0.88924337]
 [-0.61790165  0.2836423   0.41807643]]
```

```
z=np.array([2,3,6,2,5,2])
print(z)
print(np.dot(z,y))
```

```
[2 3 6 2 5 2]
71
```

```
q=np.array([[4,6,7],
            [5,5,2],
            [3,4,6]])
print('P =',P)
```

```
print('Q =',q)
print("P+Q",P+q)
print("P-Q",P-q)
```

```
P = [[1 3 6]
 [1 4 5]
 [2 2 7]]
Q = [[4 6 7]
 [5 5 2]
 [3 4 6]]
P+Q [[ 5  9 13]
 [ 6  9  7]
 [ 5  6 13]]
P-Q [[-3 -3 -1]
 [-4 -1  3]
 [-1 -2  1]]
```

```
print("Element-wise product of P and Q:")
P * q
```

```
Element-wise product of P and Q:
array([[ 4, 18, 42],
       [ 5, 20, 10],
       [ 6,  8, 42]])
```

```
print("p inverse")
print(np.linalg.inv(P))
print("Q inverse")
print(np.linalg.inv(q))
```

```
p inverse
[[-2.          1.          1.        ]
 [-0.33333333  0.55555556 -0.11111111]
 [ 0.66666667 -0.44444444 -0.11111111]]
Q inverse
[[-1.04761905  0.38095238  1.0952381 ]
 [ 1.14285714 -0.14285714 -1.28571429]
 [-0.23809524 -0.0952381   0.47619048]]
```

```
uniform_numbers = np.random.uniform(low=0, high=1, size=10)
print("Uniform Distribution:", uniform_numbers)

gaussian_numbers = np.random.normal(loc=0, scale=1, size=10)
print("Gaussian Distribution:", gaussian_numbers)

logistic_numbers = np.random.logistic(loc=0, scale=1, size=10)
print("Logistic Distribution:", logistic_numbers)
```

```
Uniform Distribution: [0.05237529 0.004243   0.30550437 0.82746903 0.32999429 0.47724595
 0.64488553 0.92460305 0.61634492 0.72443575]
Gaussian Distribution: [-0.91403299  0.22790954 -1.6152679   0.85186585 -1.64601106  0.02081975
 -0.06232084  0.08896939 -0.54895666 -1.40465068]
Logistic Distribution: [ 0.42710182  0.14579586 -0.9961165  -0.37717359  0.81575514  0.75803186
  1.23037215 -0.54305466 -0.38377879  0.6593939 ]
```

```
import pandas as pd
df1 = pd.read_csv('Iris.csv')
df2 = pd.read_csv('BostonHousingprice.csv')
df3 = pd.read_csv('Handwrittendigit.csv')
```

```
print(df1.head())
print(df2.head())
print(df3.head())
```

```
   sepal.length  sepal.width  petal.length  petal.width variety
0           5.1          3.5           1.4          0.2  Setosa
1           4.9          3.0           1.4          0.2  Setosa
2           4.7          3.2           1.3          0.2  Setosa
3           4.6          3.1           1.5          0.2  Setosa
4           5.0          3.6           1.4          0.2  Setosa
      crim    zn  indus  chas    nox     rm   age     dis  rad  tax  ptratio  \
0  0.00632  18.0   2.31     0  0.538  6.575  65.2  4.0900    1  296     15.3
1  0.02731   0.0   7.07     0  0.469  6.421  78.9  4.9671    2  242     17.8
2  0.02729   0.0   7.07     0  0.469  7.185  61.1  4.9671    2  242     17.8
3  0.03237   0.0   2.18     0  0.458  6.998  45.8  6.0622    3  222     18.7
4  0.06905   0.0   2.18     0  0.458  7.147  54.2  6.0622    3  222     18.7

        b  lstat  medv
0  396.90   4.98  24.0
1  396.90   9.14  21.6
2  392.83   4.03  34.7
3  394.63   2.94  33.4
4  396.90   5.33  36.2
```

```
   label  1x1  1x2  1x3  1x4  1x5  1x6  1x7  1x8  1x9  ...  28x19  28x20  \
0      5    0    0    0    0    0    0    0    0    0  ...    0.0    0.0
1      0    0    0    0    0    0    0    0    0    0  ...    0.0    0.0
2      4    0    0    0    0    0    0    0    0    0  ...    0.0    0.0
3      1    0    0    0    0    0    0    0    0    0  ...    0.0    0.0
4      9    0    0    0    0    0    0    0    0    0  ...    0.0    0.0

   28x21  28x22  28x23  28x24  28x25  28x26  28x27  28x28
0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
1    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
2    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
3    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
4    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0

[5 rows x 785 columns]
```

```
print(df1.shape)
print(df2.shape)
print(df3.shape)
```

```
(150, 5)
(506, 14)
(39004, 785)
```

```
print(df1.describe())
print(df2.describe())
print(df3.describe())
```

```
       sepal.length  sepal.width  petal.length  petal.width
count    150.000000   150.000000    150.000000   150.000000
mean       5.843333     3.057333      3.758000     1.199333
std        0.828066     0.435866      1.765298     0.762238
min        4.300000     2.000000      1.000000     0.100000
25%        5.100000     2.800000      1.600000     0.300000
50%        5.800000     3.000000      4.350000     1.300000
75%        6.400000     3.300000      5.100000     1.800000
max        7.900000     4.400000      6.900000     2.500000
             crim          zn       indus        chas         nox          rm  \
count  506.000000  506.000000  506.000000  506.000000  506.000000  506.000000
mean     3.613524   11.363636   11.136779    0.069170    0.554695    6.284634
std      8.601545   23.322453    6.860353    0.253994    0.115878    0.702617
min      0.006320    0.000000    0.460000    0.000000    0.385000    3.561000
25%      0.082045    0.000000    5.190000    0.000000    0.449000    5.885500
50%      0.256510    0.000000    9.690000    0.000000    0.538000    6.208500
75%      3.677083   12.500000   18.100000    0.000000    0.624000    6.623500
max     88.976200  100.000000   27.740000    1.000000    0.871000    8.780000

              age         dis         rad         tax     ptratio           b  \
count  506.000000  506.000000  506.000000  506.000000  506.000000  506.000000
mean    68.574901    3.795043    9.549407  408.237154   18.455534  356.674032
std     28.148861    2.105710    8.707259  168.537116    2.164946   91.294864
min      2.900000    1.129600    1.000000  187.000000   12.600000    0.320000
25%     45.025000    2.100175    4.000000  279.000000   17.400000  375.377500
50%     77.500000    3.207450    5.000000  330.000000   19.050000  391.440000
75%     94.075000    5.188425   24.000000  666.000000   20.200000  396.225000
max    100.000000   12.126500   24.000000  711.000000   22.000000  396.900000

            lstat        medv
count  506.000000  506.000000
mean    12.653063   22.532806
std      7.141062    9.197104
min      1.730000    5.000000
25%      6.950000   17.025000
50%     11.360000   21.200000
75%     16.955000   25.000000
max     37.970000   50.000000
              label      1x1      1x2      1x3      1x4      1x5      1x6  \
count  39004.000000  39004.0  39004.0  39004.0  39004.0  39004.0  39004.0
mean       4.453287      0.0      0.0      0.0      0.0      0.0      0.0
std        2.890306      0.0      0.0      0.0      0.0      0.0      0.0
min        0.000000      0.0      0.0      0.0      0.0      0.0      0.0
25%        2.000000      0.0      0.0      0.0      0.0      0.0      0.0
50%        4.000000      0.0      0.0      0.0      0.0      0.0      0.0
75%        7.000000      0.0      0.0      0.0      0.0      0.0      0.0
max        9.000000      0.0      0.0      0.0      0.0      0.0      0.0

           1x7      1x8      1x9  ...         28x19         28x20  \
count  39004.0  39004.0  39004.0  ...  39003.000000  39003.000000
mean       0.0      0.0      0.0  ...      0.214291      0.114401
std        0.0      0.0      0.0  ...      6.342657      4.461866
min        0.0      0.0      0.0  ...      0.000000      0.000000
25%        0.0      0.0      0.0  ...      0.000000      0.000000
50%        0.0      0.0      0.0  ...      0.000000      0.000000
75%        0.0      0.0      0.0  ...      0.000000      0.000000
max        0.0      0.0      0.0  ...    254.000000    254.000000
```

```
# prompt: Generate a simulated dataset for regression application using NumPy, with the following
```

```
# prompt: Generate a simulated dataset for Regression application using numpy with the following
# properties.
# a. Number of samples = 100
# b. Number of features = 4
# c. Number of targets = 1
# d. Zero noise

import numpy as np

# Set random seed for reproducibility
np.random.seed(42)

# Number of samples, features, and targets
n_samples = 100
n_features = 4
n_targets = 1

# Generate random features (X)
X = np.random.rand(n_samples, n_features)
# Generate target variable (y) as a linear combination of features with no noise
true_coefficients = np.random.rand(n_features)
y = np.dot(X, true_coefficients) #performs a dot product between the
                                 # feature matrix X and true_coefficients.

# Print shapes for verification
print("feature of X:", X[:5])
print("target of y:", y[:5])
```

```
feature of X: [[0.37454012 0.95071431 0.73199394 0.59865848]
 [0.15601864 0.15599452 0.05808361 0.86617615]
 [0.60111501 0.70807258 0.02058449 0.96990985]
 [0.83244264 0.21233911 0.18182497 0.18340451]
 [0.30424224 0.52475643 0.43194502 0.29122914]]
target of y: [1.76130144 0.90208718 1.51305187 0.52093551 0.96392482]
```



```
# prompt: Generate a simulated dataset for Classification application with the following properties.  using sklearn
# a. Number of samples = 100
# b. Number of features = 4
# c. Number of classes = 2
# d. Zero noise

import numpy as np
from sklearn.datasets import make_classification

# Generate a simulated dataset for classification
X, y = make_classification(
    n_samples=100,
    n_features=4,
    n_informative=4,
    n_redundant=0,
    n_classes=2,
    n_clusters_per_class=1,
    flip_y=0.0,
    class_sep=1.0,
    random_state=42  # Set random state for reproducibility
)
# Print the first few samples
print("FEATURE X:\n", X[:5])
print("LABEL of y:\n", y[:5])
```

```
FEATURE X:
 [[-0.38504199  0.63366243  1.04962019  1.19606278]
 [-1.94421158 -2.99326909 -4.52479762 -0.95757598]
 [-1.18450511 -1.49369435 -2.18863413 -0.30839581]
 [ 1.2214474   2.67352631  1.08840826  0.24260342]
 [-2.69450216 -3.09813062 -2.7109857  -2.32296203]]
LABEL of y:
 [1 1 0 1 0]
```

```
series_a= pd.Series([7,11,13,17])
print(series_a)

series_b=pd.Series([100]*5)
print(series_b)

random_series = pd.Series(np.random.randint(0,20, size=20))
print(random_series.describe())

Temperatures=pd.Series([98.6,98.9,100.2,97.9],
index=['Julie','Charlie','Sam','Andrea'])
print(Temperatures)
```

```
dict1={'Julie':98.6,'Charlie':98.9,'Sam':100.2,'Andrea':97.9}
Series_e=pd.Series(dict1)
print(Series_e)
```

```
0     7
1    11
2    13
3    17
dtype: int64
0    100
1    100
2    100
3    100
4    100
dtype: int64
count    20.000000
mean     11.450000
std       5.744334
min       2.000000
25%       7.500000
50%      11.500000
75%      16.500000
max      19.000000
dtype: float64
Julie       98.6
Charlie     98.9
Sam        100.2
Andrea      97.9
dtype: float64
Julie       98.6
Charlie     98.9
Sam        100.2
Andrea      97.9
dtype: float64
```

```
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']

df1= pd.DataFrame(exam_data,index=labels)
print(df1)

df1.loc[df1['name'] == 'James', 'name'] = 'Suresh'
print(df1)

color = ['Red', 'Blue', 'Orange', 'Red', 'White', 'Yellow', 'Purple', 'Green', 'Pink', 'Black']
df1['color'] = color
print(df1)

headers=list(df1.columns)
print(headers)
```

```
        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
        name  score  attempts qualify
a  Anastasia   12.5         1     yes
b       Dima    9.0         3      no
c  Katherine   16.5         2     yes
d     Suresh    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
        name  score  attempts qualify   color
a  Anastasia   12.5         1     yes     Red
b       Dima    9.0         3      no    Blue
c  Katherine   16.5         2     yes  Orange
d     Suresh    NaN         3      no     Red
e      Emily    9.0         2      no   White
f    Michael   20.0         3     yes  Yellow
g    Matthew   14.5         1     yes  Purple
h      Laura    NaN         1      no   Green
i      Kevin    8.0         2      no    Pink
```

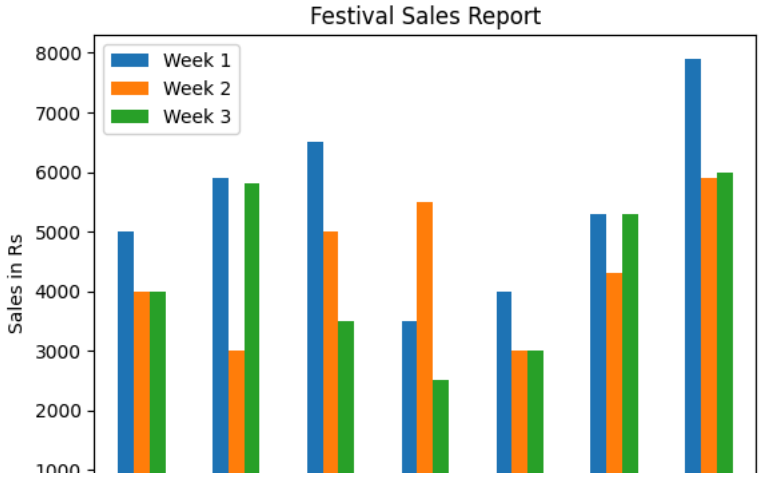
```
    j      Jonas   19.0         1     yes   Black
    ['name', 'score', 'attempts', 'qualify', 'color']
```

```
# prompt: An NGO has participated in a three-week cultural festival. Using Pandas, store the sales
# (in Rs) made day wise for every week in a CSV file named “FestSales.csv”
import pandas as pd
data = {
    'Week 1': [5000, 5900, 6500, 3500, 4000, 5300, 7900],
    'Week 2': [4000, 3000, 5000, 5500, 3000, 4300, 5900],
    'Week 3': [4000, 5800, 3500, 2500, 3000, 5300, 6000]
}

# Create a Pandas DataFrame from the data
df = pd.DataFrame(data)

import matplotlib.pyplot as plt

# Read the CSV data
df = pd.read_csv("FestSales.csv")

# Create a line plot directly from the DataFrame
df.plot(title="Festival Sales Report")

# Customize the plot
plt.xlabel("Days")
plt.ylabel("Sales in Rs")
plt.legend()
plt.grid(True)
plt.show()
```

Generate 10 random numbers using numpy Close

```
df = pd.read_csv("FestSales.csv")

# Add a new column 'Day' at the beginning
df.insert(0, 'Day', ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'])

# Write the modified DataFrame back to the CSV file
df.to_csv("FestSales_modified.csv", index=False)
print(df)
# Create a bar plot
df.plot(kind='bar')

# Customize the plot
plt.title("Festival Sales Report")
plt.ylabel("Sales in Rs")
plt.xlabel("Day")

plt.show()
```

```
         Day  Week 1  Week 2  Week 3
0     Monday    5000    4000    4000
1    Tuesday    5900    3000    5800
2  Wednesday    6500    5000    3500
3   Thursday    3500    5500    2500
4     Friday    4000    3000    3000
5   Saturday    5300    4300    5300
6     Sunday    7900    5900    6000
```




```
# prompt:  Download the data from the following link and keep in your working directory.
# https://people.sc.fsu.edu/~jburkardt/data/csv/trees.csv) Display the number of rows and columns in the data) Display the number of rows

!wget https://people.sc.fsu.edu/~jburkardt/data/csv/trees.csv
import pandas as pd

# Load the CSV file into a pandas DataFrame
df_trees = pd.read_csv("trees.csv")

# Display the number of rows and columns
num_rows, num_cols = df_trees.shape
print(f"Number of rows: {num_rows}")
print(f"Number of columns: {num_cols}")


print(df_trees.describe())

print(df_trees.describe().round(3))

#Find the correlation between the attributes. Comment on the results
print(df_trees.corr())



#Since the data is spread over a wide range with different scales, it is not suitable
#to train models. Hence, bring the data into the range of [0 - 1]
scaler = MinMaxScaler()
df_trees_normalized = pd.DataFrame(scaler.fit_transform(df_trees), columns=df_trees.columns)
print("\nNormalized data:\n", df_trees_normalized.head())



# prompt: The statsmodels package (installed in the code cell above) includes built-in datasets.
# Execute the code below to download data from the American National Election Studies
# of 1996 and print a detailed description of the schema.

import statsmodels.api as sm
anes96 = sm.datasets.anes96
df= anes96.load_pandas().data
print(df.describe)

#a
print(df.head())
#B
print(df.shape[0])
print(df.shape[1])

print(f"The age of the youngest person is:",df['age'].min())
print(f"The age of the oldest person is:",df['age'].max())
print(f"Average TV news watching per week: {df['TVnews'].mean():.1f}")

print("\nMissing Values:")
print(df.isnull().sum())
#C
df.rename(columns={'educ':'education'},inplace=True)
```

```
def categorize_party(Pid):
 if pid in['strong Democrat','Weak Democrat']:
    return 'Democrat'
 elif pid in['Strong Republic','Weak Republic']:
    return 'Republican'
 else:
    return 'Independent'
 df['party'] = df['Pid'].apply(categorize_party)

def age_category(age):
    if 18 <= age <= 24:
        return '18-24'
    elif 24 <= age <= 34:
        return '25-34'
    elif 34 <= age <= 44:
        return '35-44'
    elif 44 <= age <= 54:
        return '45-54'
    elif 54 <= age <= 64:
        return '55-64'
    else:
        return '65 and over'
df['age_category'] = df['age'].apply(age_category)
print(df.head())
```

```
<bound method NDFrame.describe of      popul  TVnews  selfLR  ClinLR  DoleLR  PID   age  educ  income  vote  \
0      0.0     7.0     7.0     1.0     6.0  6.0  36.0   3.0     1.0   1.0
1    190.0     1.0     3.0     3.0     5.0  1.0  20.0   4.0     1.0   0.0
2     31.0     7.0     2.0     2.0     6.0  1.0  24.0   6.0     1.0   0.0
3     83.0     4.0     3.0     4.0     5.0  1.0  28.0   6.0     1.0   0.0
4    640.0     7.0     5.0     6.0     4.0  0.0  68.0   6.0     1.0   0.0
..     ...     ...     ...     ...     ...  ...   ...   ...     ...   ...
939    0.0     7.0     7.0     1.0     6.0  4.0  73.0   6.0    24.0   1.0
940    0.0     7.0     5.0     2.0     6.0  6.0  50.0   6.0    24.0   1.0
941    0.0     3.0     6.0     2.0     7.0  5.0  43.0   6.0    24.0   1.0
942    0.0     6.0     6.0     2.0     5.0  6.0  46.0   7.0    24.0   1.0
943   18.0     7.0     4.0     2.0     6.0  3.0  61.0   7.0    24.0   1.0

     logpopul
0   -2.302585
1    5.247550
2    3.437208
3    4.420045
4    6.461624
..        ...
939 -2.302585
940 -2.302585
941 -2.302585
942 -2.302585
943  2.895912

[944 rows x 11 columns]>
   popul  TVnews  selfLR  ClinLR  DoleLR  PID   age  educ  income  vote  \
0    0.0     7.0     7.0     1.0     6.0  6.0  36.0   3.0     1.0   1.0
1  190.0     1.0     3.0     3.0     5.0  1.0  20.0   4.0     1.0   0.0
2   31.0     7.0     2.0     2.0     6.0  1.0  24.0   6.0     1.0   0.0
3   83.0     4.0     3.0     4.0     5.0  1.0  28.0   6.0     1.0   0.0
4  640.0     7.0     5.0     6.0     4.0  0.0  68.0   6.0     1.0   0.0

   logpopul
0 -2.302585
1  5.247550
2  3.437208
3  4.420045
4  6.461624
944
11
The age of the youngest person is: 19.0
The age of the oldest person is: 91.0
Average TV news watching per week: 3.7

Missing Values:
popul       0
TVnews      0
selfLR      0
ClinLR      0
DoleLR      0
PID         0
age         0
educ        0
income      0
vote        0
logpopul    0
```

```
Start coding or generate with AI.
```

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.