Assignment 7 Solutions

```
#Answer 1 (a)
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]]
```

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


#answer 1 (c)
print(X[0][2],X[1][1],X[2][1])
```

```
3 6 10
```

```
#answer 1 (d)
print(y[:3])
```

```
array([1, 2, 3])
```

```
print(y[-3:])
```

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

```
z= np.array([[[1,2],[3,4]],[[5,6],[7,8]]])
print(z.shape)
print(z.size)
print(z.ndim)
```

```
(2, 2, 2)
8
3
```

```
#answer 1 e Find the size, number of elements and number of dimensions of both X and y.
print('X size is ',X.shape)
print('X number of elements is ',X.size)
print('X number of dimensions is ',X.ndim)
print('y size is ',y.shape)
print('y number of elements is ',y.size)
print('y number of dimensions is ',y.ndim)
```

```
X size is  (3, 4)
X number of elements is  12
X number of dimensions is  2
y size is  (6,)
y number of elements is  6
y number of dimensions is  1
```

```
#1 f Generate and display X_10 = X + 10.
X_10= X+10
print(X_10)
```

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

```
#1 g Find the maximum and minimum values of both X and y.
print('X max is ',X.max())
print('X min is ',X.min())
print('y max is ',y.max())
print('y min is ',y.min())
```

```
X max is  12
X min is  1
y max is  6
y min is  1
```

```
#1 h Calculate the and display Average, Variance, and Standard Deviation values of both X and y.
#Ref website: https://www.bmj.com/about-bmj/resources-readers/publications/statistics-square-one/2-mean-and-standard-deviation
print('X average is ',X.mean())
```

```
print('X variance is ',X.var())
print('X standard deviation is ',X.std())
print('y average is ',y.mean())
print('y variance is ',y.var())
print('y standard deviation is ',y.std())
```

```
X average is  6.5
X variance is  11.916666666666666
X standard deviation is  3.452052529534663
y average is  3.5
y variance is  2.9166666666666665
y standard deviation is  1.707825127659933
```

```
#1 i Reshape 𝑋 as a 2 × 6 matrix.
print(X.reshape(2,6))
```

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

```
#1 j Find and display the transpose of both 𝑋 and 𝑦.
print(np.transpose(X))
print('--------------')
print(np.transpose(y))
```

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

```
#1 k Flatten 𝑋 and display.
print(X.flatten())
```

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

```
#1 L Generate a 3 × 3 square matrix (𝑃).
P = np.array([[1,2,3],
              [4,5,6],
              [7,8,9]])
print(P)
```

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

```
#1 m Find and display the rank.
print(np.linalg.matrix_rank(P))
```

```
2
```

```
# 1 n Find and display the determinant of 𝑃.
print(np.linalg.det(P))
```

```
0.0
```

```
# 1 o Find and display the diagonal of 𝑃.
print(np.diag(P))
```

```
[1 5 9]
```

```
# 1 p Find and display the trace of 𝑃.
print(np.trace(P))
```

```
15
```

```
#  1 q Find the eigen values and eigen vectors of 𝑃.
eigenvalues,eigenvectors = np.linalg.eig(P)
print(eigenvalues)
print('-------------------------------------------------')
print(eigenvectors)
```

```
[ 1.61168440e+01 -1.11684397e+00 -1.30367773e-15]
-------------------------------------------------
[[-0.23197069 -0.78583024  0.40824829]
 [-0.52532209 -0.08675134 -0.81649658]
 [-0.8186735   0.61232756  0.40824829]]
```

```
# 1 r Generate z = [2, 3, 6, 2, 5 , 2]. Find the dot product between y and z.
z = np.array([2,3,6,2,5,2])
print(np.dot(y,z))
```

```
71
```

```
# 1 s Generate a 3 × 3 square matrix (Q). Q =[[4, 6, 7,],[5, 5, 2],[3, 4, 6]]. Find and display P + Q, P − Q
Q =np.array([[4, 6, 7,],[5, 5, 2],[3, 4, 6]])
print(P+Q)
print('------------')
print(P-Q)
```

```
[[ 5  8 10]
 [ 9 10  8]
 [10 12 15]]
------------
[[-3 -4 -4]
 [-1  0  4]
 [ 4  4  3]]
```

```
# 1 t Find and display element wise product of P and Q.
print(np.multiply(P,Q))
print('------------')
print(P*Q)
```

```
[[ 4 12 21]
 [20 25 12]
 [21 32 54]]
------------
[[ 4 12 21]
 [20 25 12]
 [21 32 54]]
```

```
#1 u Find and display inverse of of P and Q.
try:
  print(np.linalg.inv(P))
  print('------------')
except:
  print('singular matrix error')

print('------------')
print(np.linalg.inv(Q))
```

```
singular matrix error
------------
[[-1.04761905  0.38095238  1.0952381 ]
 [ 1.14285714 -0.14285714 -1.28571429]
 [-0.23809524 -0.0952381   0.47619048]]
```

```
#1 v Generate 10 no.s of random numbers of uniform, gaussian and logistic distributions.
print(np.random.rand(10))
print('------------')
print(np.random.randn(10))
print('------------')
print(np.random.logistic(size=10))
```

```
[0.3139038  0.40461287 0.96873472 0.27605594 0.08655859 0.47016001
 0.99498534 0.56551917 0.64045132 0.91406403]
------------
[-0.49668148  0.45438498 -1.09581796 -0.7968974   0.68325458  0.11286536
 -2.10505027 -1.4880822   1.96420485  0.61053996]
------------
[ 0.08938229 -0.57558892  0.35369643  0.84818917  0.52231426 -5.49644894
 -1.00336766  2.32643254  2.96982765 -1.07080474]
```

```
#Go to this link to download the data file
# https://github.com/shivaagarwal-793/Machine-learning-dataset
```

Question 2 Search the internet for the following datasets, download and load the same.

a. Boston housing prices

b. Iris flowers

c. Handwritten digits

In each case

a. Display the first 5 rows.

b. Display the number of rows and columns of the data.

c. Display the descriptive statistics of all the columns.

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


#answer 2 a
print(df1.head())
print(df2.head())
print(df3.head())
```

```
      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
   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm      Species
0   1            5.1           3.5            1.4           0.2  Iris-setosa
1   2            4.9           3.0            1.4           0.2  Iris-setosa
2   3            4.7           3.2            1.3           0.2  Iris-setosa
3   4            4.6           3.1            1.5           0.2  Iris-setosa
4   5            5.0           3.6            1.4           0.2  Iris-setosa
   pixel 00  pixel 01  pixel 02  pixel 03  pixel 04  pixel 05  pixel 06  \
0        88        92         2        99        16        66        94
1        80       100        18        98        60        66       100
2         0        94         9        57        20        19         7
3        95        82        71       100        27        77        77
4        68       100         6        88        47        75        87

   pixel 07  pixel 08  pixel 09  pixel 10  pixel 11  pixel 12  pixel 13  \
0        37        70         0         0        24        42        65
1        29        42         0         0        23        42        61
2         0        20        36        70        68       100       100
3        73       100        80        93        42        56        13
4        82        85        56       100        29        75         6

   pixel 14  pixel 15  label
0       100       100      8
1        56        98      8
2        18        92      8
3         0         0      9
4         0         0      9
```

```
#answer 2 b
print(df1.shape)
print(df2.shape)
print(df3.shape)
```

```
(506, 14)
(150, 6)
(10992, 17)
```

```
#answer 2 c
print(df1.describe())
print(df2.describe())
print(df3.describe())
```

```
              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
               Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
count  150.000000     150.000000    150.000000     150.000000    150.000000
mean    75.500000       5.843333      3.054000       3.758667      1.198667
std     43.445368       0.828066      0.433594       1.764420      0.763161
min      1.000000       4.300000      2.000000       1.000000      0.100000
25%     38.250000       5.100000      2.800000       1.600000      0.300000
50%     75.500000       5.800000      3.000000       4.350000      1.300000
75%    112.750000       6.400000      3.300000       5.100000      1.800000
max    150.000000       7.900000      4.400000       6.900000      2.500000
           pixel 00      pixel 01      pixel 02      pixel 03      pixel 04  \
count  10992.000000  10992.000000  10992.000000  10992.000000  10992.000000
mean      38.814320     85.120269     40.605622     83.774199     49.770378
std       34.257783     16.218571     26.342984     19.163646     34.100515
min        0.000000      0.000000      0.000000      0.000000      0.000000
25%        6.000000     76.000000     20.000000     72.000000     18.000000
50%       32.000000     89.000000     40.000000     91.000000     53.000000
75%       65.000000    100.000000     58.000000    100.000000     78.000000
max      100.000000    100.000000    100.000000    100.000000    100.000000

           pixel 05      pixel 06      pixel 07      pixel 08      pixel 09  \
count  10992.000000  10992.000000  10992.000000  10992.000000  10992.000000
mean      65.573144     51.220251     44.498999     56.868541     33.695961
std       26.996688     30.576881     29.906104     34.135530     27.251548
min        0.000000      0.000000      0.000000      0.000000      0.000000
25%       49.000000     28.000000     23.000000     29.000000      7.000000
50%       71.000000     53.500000     43.000000     60.000000     33.000000
75%       86.000000     74.000000     64.000000     89.000000     54.000000
max      100.000000    100.000000    100.000000    100.000000    100.000000
```

Question3: 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)

# Generate 100 samples with 4 features
X = np.random.rand(100, 4)  # Features in range [0, 1]

# Define the true coefficients for a linear relationship
true_coefficients = np.array([3.5, -2.0, 1.5, 4.0])  # You can customize these values

# Calculate the target variable (Y) with zero noise
Y = X @ true_coefficients  # Linear combination without noise

# Reshape Y to a 2D array with one target column
Y = Y.reshape(-1, 1)

# Display first few rows of the dataset
print("Features (X):\n", X[:5])
print("Target (Y):\n", Y[:5])
```

```
Features (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 (Y):
 [[2.90208665]
 [3.7859062 ]
 [4.59827354]
 [3.49522651]
 [1.82816908]]
```

Question 4 : Generate a simulated dataset for classification application with the following properties.

a. Number of samples = 100

b. Number of features = 4

c. Number of classes = 2

d. Zero noise

```
from sklearn.datasets import make_classification
import numpy as np

# Generate the classification dataset
X, y = make_classification(
    n_samples=100,           # 100 samples
    n_features=4,            # 4 features
    n_informative=4,         # All 4 features are informative
    n_redundant=0,           # No redundant features
    n_clusters_per_class=1,  # Each class is centered around one cluster
    n_classes=2,             # Binary classification
    class_sep=2.0,           # Large class separation to ensure zero noise
    random_state=42          # Reproducibility
)

# Display first few rows of the dataset
print("Features (X):\n", X[:5])
print("Labels (y):\n", y[:5])
```

```
Features (X):
 [[-1.38504199 -0.36633757  0.04962019  2.19606278]
 [-2.94421158 -3.99326909 -5.52479762  0.04242402]
 [-2.18450511 -2.49369435 -3.18863413 -1.30839581]
 [ 0.2214474   1.67352631  0.08840826  1.24260342]
 [-3.69450216 -4.09813062 -3.7109857  -3.32296203]]
Labels (y):
 [1 1 0 1 0]
```

Question 5: Perform the following tasks with pandas Series:

a) Create a Series from the list [7, 11, 13, 17].

b) Create a Series with five elements that are all 100.0.

c) Create a Series with 20 elements that are all random numbers in the range 0 to

Use method describe to produce the Series' basic descriptive statistics.

d) Create a Series called temperatures of the floating-point values 98.6, 98.9, 100.2 and 97.9. Using the index keyword argument, specify the custom indices 'Julie', 'Charlie', 'Sam' and 'Andrea'.

e) Form a dictionary from the names and values in Part (d), then use it to initialize a Series.

```
import pandas as pd
import numpy as np

# Part (a): Create a Series from the list [7, 11, 13, 17].
series_a = pd.Series([7, 11, 13, 17])
print("Part (a): Series from the list [7, 11, 13, 17]:\n", series_a)

# Part (b): Create a Series with five elements that are all 100.0.
series_b = pd.Series([100.0] * 5)
print("\nPart (b): Series with five elements, all 100.0:\n", series_b)

# Part (c): Create a Series with 20 elements that are random numbers in the range 0 to 1.
series_c = pd.Series(np.random.rand(20))
print("\nPart (c): Series with 20 random numbers:\n", series_c)

# Use describe() to produce the basic descriptive statistics of the Series.
print("\nDescriptive statistics for the Series (Part c):\n", series_c.describe())

# Part (d): Create a Series called 'temperatures' with custom indices.
temperatures = pd.Series([98.6, 98.9, 100.2, 97.9], index=['Julie', 'Charlie', 'Sam', 'Andrea'])
print("\nPart (d): Series with custom indices:\n", temperatures)

# Part (e): Form a dictionary from the names and values in Part (d), then use it to initialize a Series.
data_dict = {'Julie': 98.6, 'Charlie': 98.9, 'Sam': 100.2, 'Andrea': 97.9}
series_e = pd.Series(data_dict)
print("\nPart (e): Series initialized from a dictionary:\n", series_e)
```

```
Part (a): Series from the list [7, 11, 13, 17]:
 0     7
1    11
2    13
3    17
dtype: int64
```

```
Part (b): Series with five elements, all 100.0:
 0    100.0
1    100.0
2    100.0
3    100.0
4    100.0
dtype: float64

Part (c): Series with 20 random numbers:
 0     0.103124
1     0.902553
2     0.505252
3     0.826457
4     0.320050
5     0.895523
6     0.389202
7     0.010838
8     0.905382
9     0.091287
10    0.319314
11    0.950062
12    0.950607
13    0.573438
14    0.631837
15    0.448446
16    0.293211
17    0.328665
18    0.672518
19    0.752375
dtype: float64

Descriptive statistics for the Series (Part c):
 count    20.000000
mean      0.543507
std       0.306879
min       0.010838
25%       0.319866
50%       0.539345
75%       0.843724
max       0.950607
dtype: float64

Part (d): Series with custom indices:
 Julie       98.6
Charlie     98.9
Sam        100.2
Andrea      97.9
dtype: float64

Part (e): Series initialized from a dictionary:
```

```
# Question 06: Consider Sample Python dictionary data and list 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']
a) Write a program to create and display a DataFrame from the above specified dictionary data with the corresponding index labels.
b) Write a program to change the name 'James' to 'Suresh' in the name column of the DataFrame.
c) Write a program to insert a new column named ‘color’ in the existing DataFrame and add corresponding color values.
d) Write a program to get list from DataFrame column headers.
```

```
import pandas as pd
import numpy as np

# Sample dictionary data and 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']

# Part (a): Create and display the DataFrame with the corresponding index labels
df = pd.DataFrame(exam_data, index=labels)
print("Part (a): DataFrame from the dictionary data:\n", df)
```

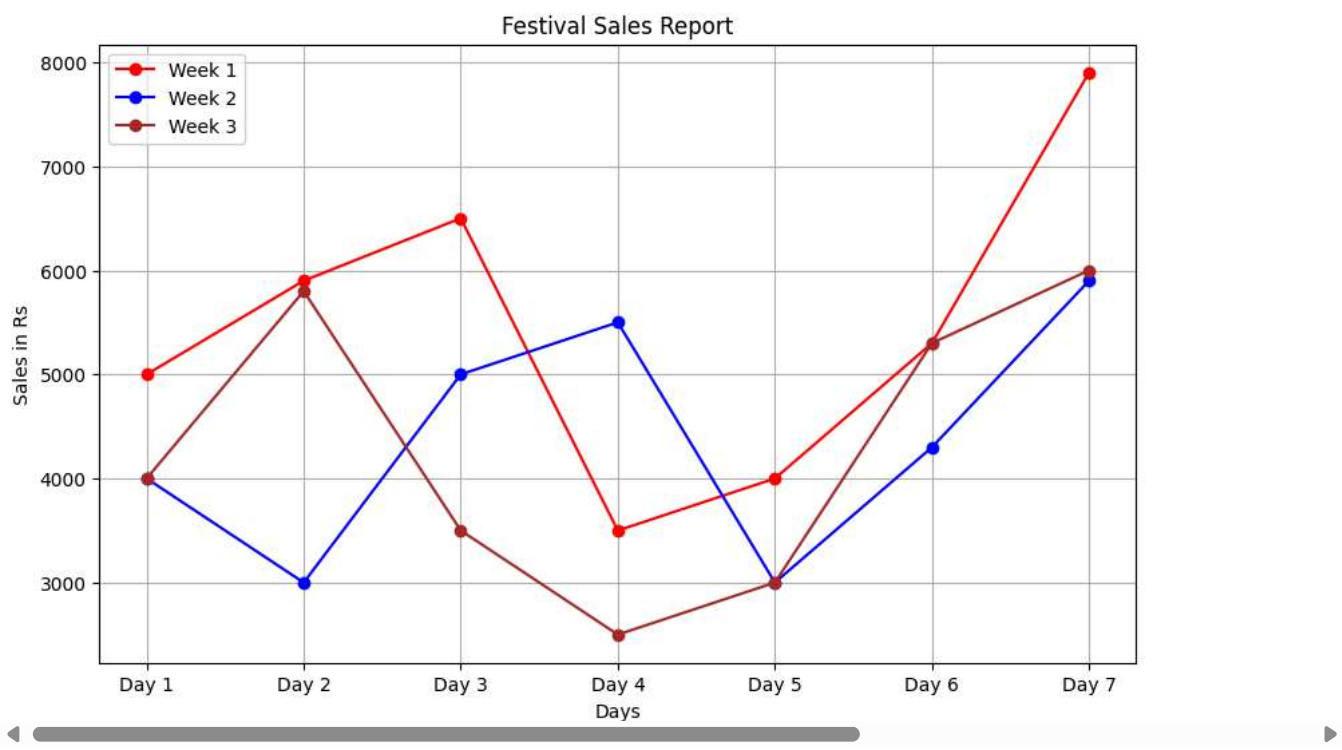
```
# Part (b): Change the name 'James' to 'Suresh' in the 'name' column
df.loc[df['name'] == 'James', 'name'] = 'Suresh'
print("\nPart (b): DataFrame after changing 'James' to 'Suresh':\n", df)

# Part (c): Insert a new column named 'color' with corresponding values
color_values = ['red', 'blue', 'green', 'yellow', 'pink', 'black', 'white', 'orange', 'purple', 'brown']
df['color'] = color_values
print("\nPart (c): DataFrame after adding a 'color' column:\n", df)

# Part (d): Get the list of column headers from the DataFrame
column_headers = df.columns.tolist()
print("\nPart (d): List of column headers:\n", column_headers)
```

```
Part (a): DataFrame from the dictionary data:
         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

Part (b): DataFrame after changing 'James' to 'Suresh':
         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

Part (c): DataFrame after adding a 'color' column:
         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   green
d     Suresh    NaN         3      no  yellow
e      Emily    9.0         2      no    pink
f    Michael   20.0         3     yes   black
g    Matthew   14.5         1     yes   white
h      Laura    NaN         1      no  orange
i      Kevin    8.0         2      no  purple
j      Jonas   19.0         1     yes   brown

Part (d): List of column headers:
 ['name', 'score', 'attempts', 'qualify', 'color']
```

```
# Question 07: An NGO has participated in a three-week cultural festival. Using Pandas, store the sales (in Rs) made day wise for every week in
in the table below.
Week 1 Week 2 Week 3
5000 4000 4000
5900 3000 5800
6500 5000 3500
3500 5500 2500
4000 3000 3000
5300 4300 5300
7900 5900 6000

Write a Python script to display the sales for the three weeks using a Line chart. It should
have the following:
i. Chart title as “Festival Sales Report”.
ii. X - axis label as Days.
iii. Y - axis label as “Sales in Rs”.
iv. Line colours are red for week 1, blue for week 2 and brown for week.
```

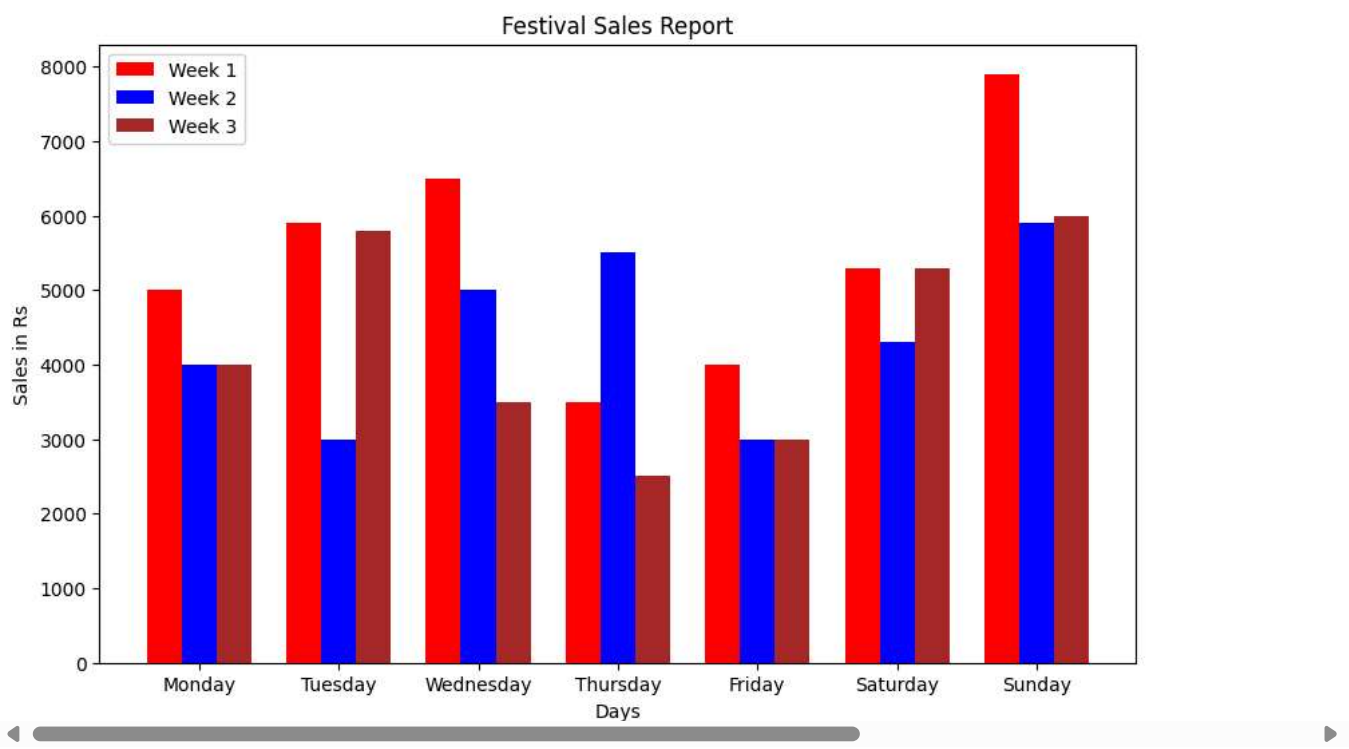
```
import pandas as pd
import matplotlib.pyplot as plt

# Step 1: Create and save the DataFrame to a CSV file
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]
}
df = pd.DataFrame(data, index=["Day 1", "Day 2", "Day 3", "Day 4", "Day 5", "Day 6", "Day 7"])
df.to_csv("FestSales.csv", index=True)

# Step 2: Read the data from the CSV file
df_sales = pd.read_csv("FestSales.csv", index_col=0)

# Step 3: Plot the line chart
plt.figure(figsize=(10, 6))

# Plot each week's sales with specified colors
plt.plot(df_sales.index, df_sales['Week 1'], color='red', marker='o', label='Week 1')
plt.plot(df_sales.index, df_sales['Week 2'], color='blue', marker='o', label='Week 2')
plt.plot(df_sales.index, df_sales['Week 3'], color='brown', marker='o', label='Week 3')

# Add chart title and axis labels
plt.title('Festival Sales Report')
plt.xlabel('Days')
plt.ylabel('Sales in Rs')

# Add a legend
plt.legend()

# Show the chart
plt.grid(True)
plt.show()
```




```
# Question 8: To the above “FestSales.csv” data, add a Day column that contains the different days of week, as shown below.

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

Write a python script to display Bar plot for the “FestSales.csv” file with column Day on x axis.


import pandas as pd
import matplotlib.pyplot as plt

# Step 1: Read the existing data from the CSV file
```

```
df_sales = pd.read_csv("FestSales.csv", index_col=0)

# Step 2: Add the Day column to the DataFrame
days_of_week = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"]
df_sales.insert(0, 'Day', days_of_week)  # Insert the 'Day' column at the first position

# Step 3: Save the updated DataFrame back to the CSV file (optional step)
df_sales.to_csv("FestSales.csv", index=False)

# Step 4: Plot the bar chart
plt.figure(figsize=(10, 6))

# Plot each week's sales as grouped bars
bar_width = 0.25  # Width of each bar
x = range(len(df_sales['Day']))  # X-axis positions

# Create bars for each week
plt.bar(x, df_sales['Week 1'], width=bar_width, label='Week 1', color='red')
plt.bar([i + bar_width for i in x], df_sales['Week 2'], width=bar_width, label='Week 2', color='blue')
plt.bar([i + 2 * bar_width for i in x], df_sales['Week 3'], width=bar_width, label='Week 3', color='brown')

# Customize the chart
plt.xlabel('Days')
plt.ylabel('Sales in Rs')
plt.title('Festival Sales Report')
plt.xticks([i + bar_width for i in x], df_sales['Day'])  # Set x-ticks to the Day column
plt.legend()

# Display the chart
plt.show()
```




```
# Question 9 Perform the following tasks using Python and associated libraries.
a) Download the data from the following link and keep in your working directory.
https://people.sc.fsu.edu/~jburkardt/data/csv/trees.csv
b) Display the number of rows and columns in the data.
c) Display the first and last three data.
d) Generate the statistical overview of the dataset.
e) Generate the statistical overview of the dataset displaying only three digits after decimal point.
f) Find the correlation between the attributes. Comment on the results.
g) 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 o
```

```
import pandas as pd
import numpy as np
import requests
from sklearn.preprocessing import MinMaxScaler
```

```
# Part (a): Download the data from the link and save it in the working directory
url = 'https://people.sc.fsu.edu/~jburkardt/data/csv/trees.csv'
file_name = 'trees.csv'

# Download and save the file locally
response = requests.get(url)
with open(file_name, 'wb') as file:
    file.write(response.content)

# Read the downloaded CSV file into a DataFrame
df = pd.read_csv(file_name)

# Part (b): Display the number of rows and columns in the data
print(f"Part (b): The dataset has {df.shape[0]} rows and {df.shape[1]} columns.\n")

# Part (c): Display the first and last three rows of the dataset
print("Part (c): First three rows of the dataset:\n", df.head(3))
print("\nLast three rows of the dataset:\n", df.tail(3))

# Part (d): Generate the statistical overview of the dataset
print("\nPart (d): Statistical overview of the dataset:\n", df.describe())

# Part (e): Generate statistical overview with three decimal points
pd.set_option('display.float_format', lambda x: '%.3f' % x)  # Set decimal display
print("\nPart (e): Statistical overview with three decimal points:\n", df.describe())

# Part (f): Find the correlation between the attributes and comment on the results
correlation_matrix = df.corr()
print("\nPart (f): Correlation matrix:\n", correlation_matrix)

# Interpretation comment:
# High positive correlations indicate attributes that increase together. For example,
# Girth and Volume show strong positive correlation, suggesting larger trees have more volume.

# Part (g): Normalize the data to bring it into the range [0, 1]
scaler = MinMaxScaler()
normalized_data = scaler.fit_transform(df)  # Scale the entire DataFrame

# Convert the scaled data back into a DataFrame
df_normalized = pd.DataFrame(normalized_data, columns=df.columns)

print("\nPart (g): Normalized Data:\n", df_normalized.head())
```

```
Part (b): The dataset has 31 rows and 4 columns.

Part (c): First three rows of the dataset:
    Index   "Girth (in)"   "Height (ft)"   "Volume(ft^3)"
0      1            8.3              70             10.3
1      2            8.6              65             10.3
2      3            8.8              63             10.2

Last three rows of the dataset:
     Index   "Girth (in)"   "Height (ft)"   "Volume(ft^3)"
28     29           18.0              80             51.5
29     30           18.0              80             51.0
30     31           20.6              87             77.0

Part (d): Statistical overview of the dataset:
            Index   "Girth (in)"   "Height (ft)"   "Volume(ft^3)"
count  31.000000      31.000000       31.000000        31.000000
mean   16.000000      13.248387       76.000000        30.170968
std     9.092121       3.138139        6.371813        16.437846
min     1.000000       8.300000       63.000000        10.200000
25%     8.500000      11.050000       72.000000        19.400000
50%    16.000000      12.900000       76.000000        24.200000
75%    23.500000      15.250000       80.000000        37.300000
max    31.000000      20.600000       87.000000        77.000000

Part (e): Statistical overview with three decimal points:
        Index   "Girth (in)"   "Height (ft)"   "Volume(ft^3)"
count 31.000         31.000          31.000           31.000
mean  16.000         13.248          76.000           30.171
std    9.092          3.138           6.372           16.438
min    1.000          8.300          63.000           10.200
25%    8.500         11.050          72.000           19.400
50%   16.000         12.900          76.000           24.200
75%   23.500         15.250          80.000           37.300
max   31.000         20.600          87.000           77.000

Part (f): Correlation matrix:
                   Index   "Girth (in)"   "Height (ft)"   "Volume(ft^3)"
Index            1.000          0.967           0.467            0.903
 "Girth (in)"    0.967          1.000           0.519            0.967
 "Height (ft)"   0.467          0.519           1.000            0.598
 "Volume(ft^3)"  0.903          0.967           0.598            1.000
```

```
Part (g): Normalized Data:
    Index   "Girth (in)"   "Height (ft)"   "Volume(ft^3)"
0  0.000          0.000           0.292            0.001
1  0.033          0.024           0.083            0.001
2  0.067          0.041           0.000            0.000
3  0.100          0.179           0.375            0.093
4  0.133          0.195           0.750            0.129
```

```
# Question 10: 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 pandas as pd
import statsmodels.api as sm
import numpy as np
anes96 = sm.datasets.anes96
print(anes96.NOTE)
a) The DataFrame (df) contains data on registered voters in the United States,
including demographic information and political preference. Using pandas, print
the first 5 rows of the DataFrame to get a sense of what the data looks like.
b) Answer the following questions.
i. How many observations are in the DataFrame?
ii. How many variables are measured (how many columns)?
iii. What is the age of the youngest person in the data? The oldest?
iv. How many days a week does the average respondent watch TV news (round to
the nearest tenth)?
v. Check for missing values. Are there any?
c) We want to adjust the dataset for our use. Do the following:
i. Rename the educ column as education.
ii. Create a new column called party based on each respondent's answer to PID.
 party should equal Democrat if the respondent selected either Strong
Democrat or Weak Democrat.
 party will equal Republican if the respondent selected Strong or Weak
Republican for PID and
 party will equal Independent if they selected anything else.
iii. Create a new column called age_group that buckets respondents into the following categories based on their age: 18-24, 25-34, 35-44, 45-54
```

```
import pandas as pd
import statsmodels.api as sm

# Step 1: Load the ANES 1996 dataset
anes96 = sm.datasets.anes96.load_pandas()
df = anes96.data  # Load dataset into a DataFrame

# Print dataset schema and notes
print(df.describe())

# Part (a): Print the first 5 rows of the DataFrame
print("\nPart (a): First 5 rows of the DataFrame:\n", df.head())

# Part (b): Answer specific questions
# i. Number of observations
num_observations = df.shape[0]
print(f"\n(b.i): Number of observations: {num_observations}")

# ii. Number of variables (columns)
num_columns = df.shape[1]
print(f"(b.ii): Number of variables (columns): {num_columns}")

# iii. Age of the youngest and oldest person
youngest_age = df['age'].min()
oldest_age = df['age'].max()
print(f"(b.iii): Youngest age: {youngest_age}, Oldest age: {oldest_age}")

# iv. Average days per week respondents watch TV news (round to nearest tenth)
avg_tv_news = df['TVnews'].mean().round(1)
print(f"(b.iv): Average days per week watching TV news: {avg_tv_news}")

# v. Check for missing values
missing_values = df.isnull().sum()
print(f"(b.v): Missing values in each column:\n{missing_values}")

# Part (c): Adjust the dataset
# i. Rename 'educ' column to 'education'
df.rename(columns={'educ': 'education'}, inplace=True)
```

```
print("\n(c.i): Column renamed to 'education'.")

# ii. Create a 'party' column based on 'PID'
def categorize_party(pid):
    if pid in ["Strong Democrat", "Weak Democrat"]:
        return "Democrat"
    elif pid in ["Strong Republican", "Weak Republican"]:
        return "Republican"
    else:
        return "Independent"

df['party'] = df['PID'].apply(categorize_party)
print("\n(c.ii): Created 'party' column based on PID.")

# iii. Create an 'age_group' column for age bucketing
def age_category(age):
    if 18 <= age <= 24:
        return "18-24"
    elif 25 <= age <= 34:
        return "25-34"
    elif 35 <= age <= 44:
        return "35-44"
    elif 45 <= age <= 54:
        return "45-54"
    elif 55 <= age <= 64:
        return "55-64"
    else:
        return "65 and over"

df['age_group'] = df['age'].apply(age_category)
print("\n(c.iii): Created 'age_group' column based on age.")

# Display the modified DataFrame
print("\nModified DataFrame (First 5 rows):\n", df.head())
```

```
          popul  TVnews  selfLR  ClinLR  DoleLR     PID     age    educ  \
count  944.000 944.000 944.000 944.000 944.000 944.000 944.000 944.000
mean   306.381   3.728   4.325   2.940   5.394   2.842  47.043   4.566
std   1082.607   2.677   1.438   1.384   1.269   2.273  16.423   1.599
min      0.000   0.000   1.000   1.000   1.000   0.000  19.000   1.000
25%      1.000   1.000   3.000   2.000   5.000   1.000  34.000   3.000
50%     22.000   3.000   4.000   3.000   6.000   2.000  44.000   4.000
75%    110.000   7.000   6.000   4.000   6.000   5.000  58.000   6.000
max   7300.000   7.000   7.000   7.000   7.000   6.000  91.000   7.000

       income    vote  logpopul
count 944.000 944.000   944.000
mean   16.332   0.416     2.472
std     5.975   0.493     3.187
min     1.000   0.000    -2.303
25%    14.000   0.000     0.095
50%    17.000   0.000     3.096
75%    21.000   1.000     4.701
max    24.000   1.000     8.896

Part (a): First 5 rows of the DataFrame:
     popul  TVnews  selfLR  ClinLR  DoleLR   PID    age  educ  income  vote  \
0   0.000   7.000   7.000   1.000   6.000 6.000 36.000 3.000   1.000 1.000
1 190.000   1.000   3.000   3.000   5.000 1.000 20.000 4.000   1.000 0.000
2  31.000   7.000   2.000   2.000   6.000 1.000 24.000 6.000   1.000 0.000
3  83.000   4.000   3.000   4.000   5.000 1.000 28.000 6.000   1.000 0.000
4 640.000   7.000   5.000   6.000   4.000 0.000 68.000 6.000   1.000 0.000

   logpopul
0    -2.303
1     5.248
2     3.437
3     4.420
4     6.462

(b.i): Number of observations: 944
(b.ii): Number of variables (columns): 11
(b.iii): Youngest age: 19.0, Oldest age: 91.0
(b.iv): Average days per week watching TV news: 3.7
(b.v): Missing values in each column:
popul       0
TVnews      0
selfLR      0
ClinLR      0
DoleLR      0
PID         0
age         0
educ        0
income      0
vote        0
logpopul    0
```

```
dtype: int64

(c.i): Column renamed to 'education'.

(c.ii): Created 'party' column based on PID.

(c.iii): Created 'age_group' column based on age.
```

```
#pca
```

```
from sklearn.decomposition import PCA
import pandas as pd
import numpy as np
df = pd.read_csv("pca.csv")
```

```
pca = PCA(n_components=1)
principalComponents = pca.fit_transform(df)
print(principalComponents)
```

```
[[ 4.30518692]
 [-3.73612869]
 [-5.69282771]
 [ 5.12376947]]
```

Start coding or generate with AI.