```python
In [1]: import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

Problem 1

```python
In [2]: titanic = sns.load_dataset('titanic')
titanic
```

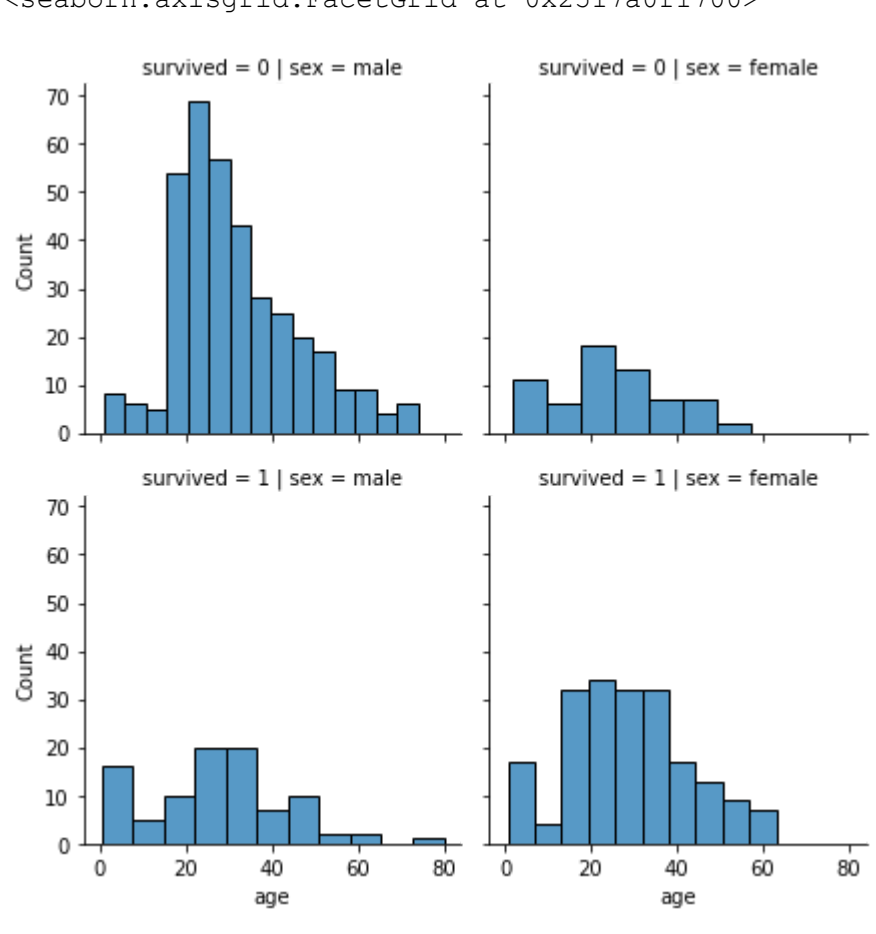
Out[2]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
886	0	2	male	27.0	0	0	13.0000	S	Second	man	True	NaN	Southampton	no	True
887	1	1	female	19.0	0	0	30.0000	S	First	woman	False	B	Southampton	yes	True
888	0	3	female	NaN	1	2	23.4500	S	Third	woman	False	NaN	Southampton	no	False
889	1	1	male	26.0	0	0	30.0000	C	First	man	True	C	Cherbourg	yes	True
890	0	3	male	32.0	0	0	7.7500	Q	Third	man	True	NaN	Queenstown	no	True

891 rows × 15 columns

```python
In [3]: g = sns.FacetGrid(titanic, col="sex",  row="survived")
g.map_dataframe(sns.histplot, x="age")
g.set_axis_labels("age", "Count")
```

Out[3]: <seaborn.axisgrid.FacetGrid at 0x2517a0f1700>



Observations:

1. Number of male passengers are more than female passengers. (Male: 577, Female: 314)
2. Death count is higher in male passengers compare to female passengers.
3. Survival ratio is higher in female passengers compare to male passengers.
4. From total male passenger's death, most of are having age between 20 to 40. Same thing we can observe for the female passengers too.
5. Only 2 males and 3 females are survived whose age are above 60.
6. Same number of male and female children survived whose age are below 10 years.
7. Regards to aged people whose age is higher than 60, Survival ratio is higher in female passengers compare to male passengers.
8. All the female who are above 60, survived.

Problem 2

```python
In [4]: from sklearn.impute import SimpleImputer
import statistics
```

```python
In [5]: mpg_file = "auto-mpg.data"
column_names = ['mpg', 'cylinders', 'displacement', 'horsepower', 'weight', 'acceleration', 'year', 'origin', 'name']
mpg_df = pd.read_csv(mpg_file, delim_whitespace=True, names=column_names)
mpg_df
```

Out[5]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	name
0	18.0	8	307.0	130.0	3504.0	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165.0	3693.0	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150.0	3436.0	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150.0	3433.0	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140.0	3449.0	10.5	70	1	ford torino
...	...	...	...	...	...	...	...	...	...
393	27.0	4	140.0	86.00	2790.0	15.6	82	1	ford mustang gl
394	44.0	4	97.0	52.00	2130.0	24.6	82	2	vw pickup
395	32.0	4	135.0	84.00	2295.0	11.6	82	1	dodge rampage
396	28.0	4	120.0	79.00	2625.0	18.6	82	1	ford ranger
397	31.0	4	119.0	82.00	2720.0	19.4	82	1	chevy s-10

398 rows × 9 columns

```python
In [6]: horsepower_missing_index = mpg_df[mpg_df.horsepower=='?'].index
mpg_df.loc[horsepower_missing_index]
```

Out[6]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	name
32	25.0	4	98.0	?	2046.0	19.0	71	1	ford pinto
126	21.0	6	200.0	?	2875.0	17.0	74	1	ford maverick
330	40.9	4	85.0	?	1835.0	17.3	80	2	renault lecar deluxe
336	23.6	4	140.0	?	2905.0	14.3	80	1	ford mustang cobra
354	34.5	4	100.0	?	2320.0	15.8	81	2	renault 18i
374	23.0	4	151.0	?	3035.0	20.5	82	1	amc concord dl

```python
In [7]: mpg_df.loc[horsepower_missing_index, 'horsepower'] = np.nan
mpg_df.loc[mpg_df.horsepower.isnull()]
```

Out[7]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	name
32	25.0	4	98.0	NaN	2046.0	19.0	71	1	ford pinto
126	21.0	6	200.0	NaN	2875.0	17.0	74	1	ford maverick
330	40.9	4	85.0	NaN	1835.0	17.3	80	2	renault lecar deluxe
336	23.6	4	140.0	NaN	2905.0	14.3	80	1	ford mustang cobra
354	34.5	4	100.0	NaN	2320.0	15.8	81	2	renault 18i
374	23.0	4	151.0	NaN	3035.0	20.5	82	1	amc concord dl

```python
In [8]: # Change horsepower datatype from object to float
mpg_df.horsepower = mpg_df.horsepower.apply(pd.to_numeric)
mpg_df.dtypes
```

```
Out[8]: mpg              float64
cylinders          int64
displacement     float64
horsepower       float64
weight           float64
acceleration     float64
year               int64
origin             int64
name              object
dtype: object
```

```python
In [9]: # summary statistic for the each numerical column
print(mpg_df[['mpg', 'cylinders', 'displacement', 'horsepower', 'weight', 'acceleration', 'year', 'origin', 'name']].mean())
print('\n')
print(mpg_df[['mpg', 'cylinders', 'displacement', 'horsepower', 'weight', 'acceleration', 'year', 'origin', 'name']].median())
print('\n')
print(mpg_df[['mpg', 'cylinders', 'displacement', 'horsepower', 'weight', 'acceleration', 'year', 'origin', 'name']].mode())
```

```
mpg               23.514573
cylinders          5.454774
displacement     193.425879
horsepower       104.469388
weight          2970.424623
acceleration      15.568090
year              76.010050
origin             1.572864
dtype: float64


mpg               23.0
cylinders          4.0
displacement     148.5
horsepower        93.5
weight          2803.5
acceleration      15.5
year              76.0
origin             1.0
dtype: float64


    mpg  cylinders  displacement  horsepower  weight  acceleration  year  \
0  13.0        4.0          97.0       150.0  1985.0          14.5  73.0
1   NaN        NaN           NaN         NaN  2130.0           NaN   NaN

   origin        name
0     1.0  ford pinto
1     NaN         NaN
```

```python
In [10]: # replace NaN with mean value
a = pd.DataFrame(mpg_df.horsepower, dtype="category")
imp_mean = SimpleImputer(strategy="mean")
f_mean = imp_mean.fit_transform(a)
print(f_mean)
```

```
[[130.        ]
 [165.        ]
 [150.        ]
 [150.        ]
 [140.        ]
 [198.        ]
 [220.        ]
 [215.        ]
 [225.        ]
 [190.        ]
 [170.        ]
 [160.        ]
 [150.        ]
 [225.        ]
 [ 95.        ]
 [ 95.        ]
 [ 97.        ]
 [ 85.        ]
 [ 88.        ]
 [ 46.        ]
 [ 87.        ]
 [ 90.        ]
 [ 95.        ]
 [113.        ]
 [ 90.        ]
 [215.        ]
 [200.        ]
 [210.        ]
 [193.        ]
 [ 88.        ]
 [ 90.        ]
 [ 95.        ]
 [104.46938776]
 [100.        ]
 [105.        ]
 [100.        ]
 [ 88.        ]
 [100.        ]
 [165.        ]
 [175.        ]
 [153.        ]
 [150.        ]
 [180.        ]
 [170.        ]
 [175.        ]
 [110.        ]
 [ 72.        ]
 [100.        ]
 [ 88.        ]
 [ 86.        ]
 [ 90.        ]
 [ 70.        ]
 [ 76.        ]
 [ 65.        ]
 [ 69.        ]
 [ 60.        ]
 [ 70.        ]
 [ 95.        ]
 [ 80.        ]
 [ 54.        ]
 [ 90.        ]
 [ 86.        ]
 [165.        ]
 [175.        ]
 [150.        ]
 [153.        ]
 [150.        ]
 [208.        ]
 [155.        ]
 [160.        ]
 [190.        ]
 [ 97.        ]
 [150.        ]
 [130.        ]
 [140.        ]
 [150.        ]
 [112.        ]
 [ 76.        ]
 [ 87.        ]
 [ 69.        ]
 [ 86.        ]
 [ 92.        ]
 [ 97.        ]
 [ 80.        ]
 [ 88.        ]
 [175.        ]
 [150.        ]
 [145.        ]
 [137.        ]
 [150.        ]
 [198.        ]
 [150.        ]
 [158.        ]
 [150.        ]
 [215.        ]
 [225.        ]
 [175.        ]
 [105.        ]
 [100.        ]
 [100.        ]
 [ 88.        ]
 [ 95.        ]
 [ 46.        ]
 [150.        ]
 [167.        ]
 [170.        ]
 [180.        ]
 [100.        ]
 [ 88.        ]
 [ 72.        ]
 [ 94.        ]
 [ 90.        ]
 [ 85.        ]
 [107.        ]
 [ 90.        ]
 [145.        ]
 [230.        ]
 [ 49.        ]
 [ 75.        ]
 [ 91.        ]
 [112.        ]
 [150.        ]
 [110.        ]
 [122.        ]
 [180.        ]
 [ 95.        ]
 [104.46938776]
 [100.        ]
 [100.        ]
 [ 67.        ]
 [ 80.        ]
 [ 65.        ]
 [ 75.        ]
 [100.        ]
 [110.        ]
 [105.        ]
 [140.        ]
 [150.        ]
 [150.        ]
 [140.        ]
 [150.        ]
 [ 83.        ]
 [ 67.        ]
 [ 78.        ]
 [ 52.        ]
 [ 61.        ]
 [ 75.        ]
 [ 75.        ]
 [ 75.        ]
 [ 97.        ]
 [ 93.        ]
 [ 67.        ]
 [ 95.        ]
 [105.        ]
 [ 72.        ]
 [ 72.        ]
 [170.        ]
 [145.        ]
 [150.        ]
 [148.        ]
 [110.        ]
 [105.        ]
 [110.        ]
 [ 95.        ]
 [110.        ]
 [110.        ]
 [129.        ]
 [ 75.        ]
 [ 83.        ]
 [100.        ]
 [ 78.        ]
 [ 96.        ]
 [ 71.        ]
 [ 97.        ]
 [ 97.        ]
 [ 70.        ]
 [ 90.        ]
 [ 95.        ]
 [ 88.        ]
 [ 98.        ]
 [115.        ]
 [ 53.        ]
 [ 86.        ]
 [ 81.        ]
 [ 92.        ]
 [ 79.        ]
 [ 83.        ]
 [140.        ]
 [150.        ]
 [120.        ]
 [152.        ]
 [100.        ]
 [105.        ]
 [ 81.        ]
 [ 90.        ]
 [ 52.        ]
 [ 60.        ]
 [ 70.        ]
 [ 53.        ]
 [100.        ]
 [ 78.        ]
 [110.        ]
 [ 95.        ]
 [ 71.        ]
 [ 70.        ]
 [ 75.        ]
 [ 72.        ]
 [102.        ]
 [150.        ]
 [ 88.        ]
 [108.        ]
 [120.        ]
 [180.        ]
 [145.        ]
 [130.        ]
 [150.        ]
 [ 68.        ]
 [ 80.        ]
 [ 58.        ]
 [ 96.        ]
 [ 70.        ]
 [145.        ]
 [110.        ]
 [145.        ]
 [130.        ]
 [110.        ]
 [105.        ]
 [100.        ]
 [ 98.        ]
 [180.        ]
 [170.        ]
 [190.        ]
 [149.        ]
 [ 78.        ]
 [ 88.        ]
 [ 75.        ]
 [ 89.        ]
 [ 63.        ]
 [ 83.        ]
 [ 67.        ]
 [ 78.        ]
 [ 97.        ]
 [110.        ]
 [110.        ]
 [ 48.        ]
 [ 66.        ]
 [ 52.        ]
 [ 70.        ]
 [ 60.        ]
 [110.        ]
 [140.        ]
 [139.        ]
 [105.        ]
 [ 95.        ]
 [ 85.        ]
 [ 88.        ]
 [100.        ]
 [ 90.        ]
 [105.        ]
 [ 85.        ]
 [110.        ]
 [120.        ]
 [145.        ]
 [165.        ]
 [139.        ]
 [140.        ]
 [ 68.        ]
 [ 95.        ]
 [ 97.        ]
 [ 75.        ]
 [ 95.        ]
 [105.        ]
 [ 85.        ]
 [ 97.        ]
 [103.        ]
 [125.        ]
 [115.        ]
 [133.        ]
 [ 71.        ]
 [ 68.        ]
 [115.        ]
 [ 85.        ]
 [ 88.        ]
 [ 90.        ]
 [110.        ]
 [130.        ]
 [129.        ]
 [138.        ]
 [135.        ]
 [155.        ]
 [142.        ]
 [125.        ]
 [150.        ]
 [ 71.        ]
 [ 65.        ]
 [ 80.        ]
 [ 80.        ]
 [ 77.        ]
 [125.        ]
 [ 71.        ]
 [ 90.        ]
 [ 70.        ]
 [ 70.        ]
 [ 65.        ]
 [ 69.        ]
 [ 90.        ]
 [115.        ]
 [115.        ]
 [ 90.        ]
 [ 76.        ]
 [ 60.        ]
 [ 70.        ]
 [ 65.        ]
 [ 90.        ]
 [ 88.        ]
 [ 90.        ]
 [ 90.        ]
 [ 78.        ]
 [ 90.        ]
 [ 75.        ]
 [ 92.        ]
 [ 75.        ]
 [ 65.        ]
 [105.        ]
 [ 65.        ]
 [ 48.        ]
 [ 48.        ]
 [ 67.        ]
 [ 67.        ]
 [ 67.        ]
 [104.46938776]
 [ 67.        ]
 [ 62.        ]
 [132.        ]
 [100.        ]
 [ 88.        ]
 [104.46938776]
 [ 72.        ]
 [ 84.        ]
 [ 84.        ]
 [ 92.        ]
 [110.        ]
 [ 84.        ]
 [ 58.        ]
 [ 64.        ]
 [ 60.        ]
 [ 67.        ]
 [ 65.        ]
 [ 62.        ]
 [ 68.        ]
 [ 63.        ]
 [ 65.        ]
 [ 65.        ]
 [ 74.        ]
 [104.46938776]
 [ 75.        ]
 [ 75.        ]
 [100.        ]
 [ 74.        ]
 [ 80.        ]
 [ 76.        ]
 [116.        ]
 [120.        ]
 [110.        ]
 [105.        ]
 [ 88.        ]
 [ 85.        ]
 [ 88.        ]
 [ 88.        ]
 [ 88.        ]
 [ 85.        ]
 [ 84.        ]
 [ 90.        ]
 [ 92.        ]
 [104.46938776]
 [ 74.        ]
 [ 68.        ]
 [ 68.        ]
 [ 63.        ]
 [ 70.        ]
 [ 88.        ]
 [ 75.        ]
 [ 70.        ]
 [ 67.        ]
 [ 67.        ]
 [ 67.        ]
 [110.        ]
 [ 85.        ]
 [ 92.        ]
 [112.        ]
 [ 96.        ]
 [ 84.        ]
 [ 90.        ]
 [ 86.        ]
 [ 52.        ]
 [ 84.        ]
 [ 79.        ]
 [ 82.        ]]
```

```python
In [11]: # variance after replacing with the mean value
var_mean = f_mean.var()
print(var_mean)
```

```
1455.511639831812
```

In [12]:
```python
# replace NaN with median value
b = pd.DataFrame(mpg_df.horsepower, dtype="category")
imp_median = SimpleImputer(strategy="median")
f_median = imp_median.fit_transform(b)
print(f_median)
```

```
[[130. ]
 [165. ]
 [150. ]
 [150. ]
 [140. ]
 [198. ]
 [220. ]
 [215. ]
 [225. ]
 [190. ]
 [170. ]
 [160. ]
 [150. ]
 [225. ]
 [ 95. ]
 [ 95. ]
 [ 97. ]
 [ 85. ]
 [ 88. ]
 [ 46. ]
 [ 87. ]
 [ 90. ]
 [ 95. ]
 [113. ]
 [ 90. ]
 [215. ]
 [200. ]
 [210. ]
 [193. ]
 [ 88. ]
 [ 90. ]
 [ 95. ]
 [ 93.5]
 [100. ]
 [105. ]
 [100. ]
 [ 88. ]
 [100. ]
 [165. ]
 [175. ]
 [153. ]
 [150. ]
 [180. ]
 [170. ]
 [175. ]
 [110. ]
 [ 72. ]
 [100. ]
 [ 88. ]
 [ 86. ]
 [ 90. ]
 [ 70. ]
 [ 76. ]
 [ 65. ]
 [ 69. ]
 [ 60. ]
 [ 70. ]
 [ 95. ]
 [ 80. ]
 [ 54. ]
 [ 90. ]
 [ 86. ]
 [165. ]
 [175. ]
 [150. ]
 [153. ]
 [150. ]
 [208. ]
 [155. ]
 [160. ]
 [190. ]
 [ 97. ]
 [150. ]
 [130. ]
 [140. ]
 [150. ]
 [112. ]
 [ 76. ]
 [ 87. ]
 [ 69. ]
 [ 86. ]
 [ 92. ]
 [ 97. ]
 [ 80. ]
 [ 88. ]
 [175. ]
 [150. ]
 [145. ]
 [137. ]
 [150. ]
 [198. ]
 [150. ]
 [158. ]
 [150. ]
 [215. ]
 [225. ]
 [175. ]
 [105. ]
 [100. ]
 [100. ]
 [ 88. ]
 [ 95. ]
 [ 46. ]
 [150. ]
 [167. ]
 [170. ]
 [180. ]
 [100. ]
 [ 88. ]
 [ 72. ]
 [ 94. ]
 [ 90. ]
 [ 85. ]
 [107. ]
 [ 90. ]
 [145. ]
 [230. ]
 [ 49. ]
 [ 75. ]
 [ 91. ]
 [112. ]
 [150. ]
 [110. ]
 [122. ]
 [180. ]
 [ 95. ]
 [ 93.5]
 [100. ]
 [100. ]
 [ 67. ]
 [ 80. ]
 [ 65. ]
 [ 75. ]
 [100. ]
 [110. ]
 [105. ]
 [140. ]
 [150. ]
 [150. ]
 [140. ]
 [150. ]
 [ 83. ]
 [ 67. ]
 [ 78. ]
 [ 52. ]
 [ 61. ]
 [ 75. ]
 [ 75. ]
 [ 75. ]
 [ 97. ]
 [ 93. ]
 [ 67. ]
 [ 95. ]
 [105. ]
 [ 72. ]
 [ 72. ]
 [170. ]
 [145. ]
 [150. ]
 [148. ]
 [110. ]
 [105. ]
 [ 95. ]
 [110. ]
 [110. ]
 [129. ]
 [ 75. ]
 [ 83. ]
 [100. ]
 [ 78. ]
 [ 96. ]
 [ 71. ]
 [ 97. ]
 [ 97. ]
 [ 70. ]
 [ 90. ]
 [ 95. ]
 [ 88. ]
 [ 98. ]
 [115. ]
 [ 53. ]
 [ 86. ]
 [ 81. ]
 [ 92. ]
 [ 79. ]
 [ 83. ]
 [140. ]
 [150. ]
 [120. ]
 [152. ]
 [100. ]
 [105. ]
 [ 81. ]
 [ 90. ]
 [ 52. ]
 [ 60. ]
 [ 70. ]
 [ 53. ]
 [100. ]
 [ 78. ]
 [110. ]
 [ 95. ]
 [ 71. ]
 [ 70. ]
 [ 75. ]
 [ 72. ]
 [102. ]
 [150. ]
 [ 88. ]
 [108. ]
 [120. ]
 [180. ]
 [145. ]
 [130. ]
 [150. ]
 [ 68. ]
 [ 80. ]
 [ 58. ]
 [ 96. ]
 [ 70. ]
 [145. ]
 [110. ]
 [145. ]
 [130. ]
 [110. ]
 [105. ]
 [100. ]
 [ 98. ]
 [180. ]
 [170. ]
 [190. ]
 [149. ]
 [ 78. ]
 [ 88. ]
 [ 75. ]
 [ 89. ]
 [ 63. ]
 [ 83. ]
 [ 67. ]
 [ 78. ]
 [ 97. ]
 [110. ]
 [110. ]
 [ 48. ]
 [ 66. ]
 [ 52. ]
 [ 70. ]
 [ 60. ]
 [110. ]
 [140. ]
 [139. ]
 [105. ]
 [ 95. ]
 [ 85. ]
 [ 88. ]
 [100. ]
 [ 90. ]
 [105. ]
 [ 85. ]
 [110. ]
 [120. ]
 [145. ]
 [165. ]
 [139. ]
 [140. ]
 [ 68. ]
 [ 95. ]
 [ 97. ]
 [ 75. ]
 [ 95. ]
 [105. ]
 [ 85. ]
 [ 97. ]
 [103. ]
 [125. ]
 [115. ]
 [133. ]
 [ 71. ]
 [ 68. ]
 [115. ]
 [ 85. ]
 [ 88. ]
 [ 90. ]
 [110. ]
 [130. ]
 [129. ]
 [138. ]
 [135. ]
 [155. ]
 [142. ]
 [125. ]
 [150. ]
 [ 71. ]
 [ 65. ]
 [ 80. ]
 [ 80. ]
 [ 77. ]
 [125. ]
 [ 71. ]
 [ 90. ]
 [ 70. ]
 [ 70. ]
 [ 65. ]
 [ 69. ]
 [ 90. ]
 [115. ]
 [115. ]
 [ 90. ]
 [ 76. ]
 [ 60. ]
 [ 70. ]
 [ 65. ]
 [ 90. ]
 [ 88. ]
 [ 90. ]
 [ 90. ]
 [ 78. ]
 [ 90. ]
 [ 75. ]
 [ 92. ]
 [ 75. ]
 [ 65. ]
 [105. ]
 [ 65. ]
 [ 48. ]
 [ 48. ]
 [ 67. ]
 [ 67. ]
 [ 67. ]
 [ 93.5]
 [ 67. ]
 [ 62. ]
 [132. ]
 [100. ]
 [ 88. ]
 [ 93.5]
 [ 72. ]
 [ 84. ]
 [ 84. ]
 [ 92. ]
 [110. ]
 [ 84. ]
 [ 58. ]
 [ 64. ]
 [ 60. ]
 [ 67. ]
 [ 65. ]
 [ 62. ]
 [ 68. ]
 [ 63. ]
 [ 65. ]
 [ 65. ]
 [ 74. ]
 [ 93.5]
 [ 75. ]
 [ 75. ]
 [100. ]
 [ 74. ]
 [ 80. ]
 [ 76. ]
 [116. ]
 [120. ]
 [110. ]
 [105. ]
 [ 88. ]
 [ 85. ]
 [ 88. ]
 [ 88. ]
 [ 88. ]
 [ 85. ]
 [ 84. ]
 [ 90. ]
 [ 92. ]
 [ 93.5]
 [ 74. ]
 [ 68. ]
 [ 68. ]
 [ 63. ]
 [ 70. ]
 [ 88. ]
 [ 75. ]
 [ 70. ]
 [ 67. ]
 [ 67. ]
 [ 67. ]
 [110. ]
 [ 85. ]
 [ 92. ]
 [112. ]
 [ 96. ]
 [ 84. ]
 [ 90. ]
 [ 86. ]
 [ 52. ]
 [ 84. ]
 [ 79. ]
 [ 82. ]]
```

In [13]:
```python
# variance after replacing with the median value
var_median = f_median.var()
print(var_median)
```

```
1457.2982752960781
```

In [14]:
```python
# replace NaN with mode value
c = pd.DataFrame(mpg_df.horsepower, dtype="category")
imp_mode = SimpleImputer(strategy="most_frequent")
f_mode = imp_mode.fit_transform(c)
print(f_mode)
```

```
[[130.]
 [165.]
 [150.]
 [150.]
 [140.]
 [198.]
 [220.]
 [215.]
 [225.]
 [190.]
 [170.]
 [160.]
 [150.]
 [225.]
 [ 95.]
 [ 95.]
 [ 97.]
 [ 85.]
 [ 88.]
 [ 46.]
 [ 87.]
 [ 90.]
 [ 95.]
 [113.]
 [ 90.]
 [215.]
 [200.]
 [210.]
 [193.]
 [ 88.]
 [ 90.]
 [ 95.]
 [150.]
 [100.]
 [105.]
 [100.]
 [ 88.]
 [100.]
 [165.]
 [175.]
 [153.]
 [150.]
 [180.]
 [170.]
 [175.]
 [110.]
 [ 72.]
 [100.]
 [ 88.]
 [ 86.]
 [ 90.]
 [ 70.]
 [ 76.]
 [ 65.]
 [ 69.]
 [ 60.]
 [ 70.]
 [ 95.]
 [ 80.]
 [ 54.]
 [ 90.]
 [ 86.]
 [165.]
 [175.]
 [150.]
 [153.]
 [150.]
 [208.]
 [155.]
 [160.]
 [190.]
 [ 97.]
 [150.]
 [130.]
 [140.]
 [150.]
 [112.]
 [ 76.]
 [ 87.]
 [ 69.]
 [ 86.]
 [ 92.]
 [ 97.]
 [ 80.]
 [ 88.]
 [175.]
 [150.]
 [145.]
 [137.]
 [150.]
 [198.]
 [150.]
 [158.]
 [150.]
 [215.]
 [225.]
 [175.]
 [105.]
 [100.]
 [100.]
 [ 88.]
 [ 95.]
 [ 46.]
 [150.]
 [167.]
 [170.]
 [180.]
 [100.]
 [ 88.]
 [ 72.]
 [ 94.]
 [ 90.]
 [ 85.]
 [107.]
 [ 90.]
 [145.]
 [230.]
 [ 49.]
 [ 75.]
 [ 91.]
 [112.]
 [150.]
 [110.]
 [122.]
 [180.]
 [ 95.]
 [150.]
 [100.]
 [100.]
 [ 67.]
 [ 80.]
 [ 65.]
 [ 75.]
 [100.]
 [110.]
 [105.]
 [140.]
 [150.]
 [150.]
 [140.]
 [150.]
 [ 83.]
 [ 67.]
 [ 78.]
 [ 52.]
 [ 61.]
 [ 75.]
 [ 75.]
 [ 75.]
 [ 97.]
 [ 93.]
 [ 67.]
 [ 95.]
 [105.]
 [ 72.]
 [ 72.]
 [170.]
 [145.]
 [150.]
 [148.]
 [110.]
 [105.]
 [ 95.]
 [110.]
 [110.]
 [129.]
 [ 75.]
 [ 83.]
 [100.]
 [ 78.]
 [ 96.]
 [ 71.]
 [ 97.]
 [ 97.]
 [ 70.]
 [ 90.]
 [ 95.]
 [ 88.]
 [ 98.]
 [115.]
 [ 53.]
 [ 86.]
 [ 81.]
 [ 92.]
 [ 79.]
 [ 83.]
 [140.]
 [150.]
 [120.]
 [152.]
 [100.]
 [105.]
 [ 81.]
 [ 90.]
 [ 52.]
 [ 60.]
 [ 70.]
 [ 53.]
 [100.]
 [ 78.]
 [110.]
 [ 95.]
 [ 71.]
 [ 70.]
 [ 75.]
 [ 72.]
 [102.]
 [150.]
 [ 88.]
 [108.]
 [120.]
 [180.]
 [145.]
 [130.]
 [150.]
 [ 68.]
 [ 80.]
 [ 58.]
 [ 96.]
 [ 70.]
 [145.]
 [110.]
 [145.]
 [130.]
 [110.]
 [105.]
 [100.]
 [ 98.]
 [180.]
 [170.]
 [190.]
 [149.]
 [ 78.]
 [ 88.]
 [ 75.]
 [ 89.]
 [ 63.]
 [ 83.]
 [ 67.]
 [ 78.]
 [ 97.]
 [110.]
 [110.]
 [ 48.]
 [ 66.]
 [ 52.]
 [ 70.]
 [ 60.]
 [110.]
 [140.]
 [139.]
 [105.]
 [ 95.]
 [ 85.]
 [ 88.]
 [100.]
 [ 90.]
 [105.]
 [ 85.]
 [110.]
 [120.]
 [145.]
 [165.]
 [139.]
 [140.]
 [ 68.]
 [ 95.]
 [ 97.]
 [ 75.]
 [ 95.]
 [105.]
 [ 85.]
 [ 97.]
 [103.]
 [125.]
 [115.]
 [133.]
 [ 71.]
 [ 68.]
 [115.]
 [ 85.]
 [ 88.]
 [ 90.]
 [110.]
 [130.]
 [129.]
 [138.]
 [135.]
 [155.]
 [142.]
 [125.]
 [150.]
 [ 71.]
 [ 65.]
 [ 80.]
 [ 80.]
 [ 77.]
 [125.]
 [ 71.]
 [ 90.]
 [ 70.]
 [ 70.]
 [ 65.]
 [ 69.]
 [ 90.]
 [115.]
 [115.]
 [ 90.]
 [ 76.]
 [ 60.]
 [ 70.]
 [ 65.]
 [ 90.]
 [ 88.]
 [ 90.]
 [ 90.]
 [ 78.]
 [ 90.]
 [ 75.]
 [ 92.]
 [ 75.]
 [ 65.]
 [105.]
 [ 65.]
 [ 48.]
 [ 48.]
 [ 67.]
 [ 67.]
 [ 67.]
 [150.]
 [ 67.]
 [ 62.]
 [132.]
 [100.]
 [ 88.]
 [150.]
 [ 72.]
 [ 84.]
 [ 84.]
 [ 92.]
 [110.]
 [ 84.]
 [ 58.]
 [ 64.]
 [ 60.]
 [ 67.]
 [ 65.]
 [ 62.]
 [ 68.]
 [ 63.]
 [ 65.]
 [ 65.]
 [ 74.]
 [150.]
 [ 75.]
 [ 75.]
 [100.]
 [ 74.]
 [ 80.]
 [ 76.]
 [116.]
 [120.]
 [110.]
 [105.]
 [ 88.]
 [ 85.]
 [ 88.]
 [ 88.]
 [ 88.]
 [ 85.]
 [ 84.]
 [ 90.]
 [ 92.]
 [150.]
 [ 74.]
 [ 68.]
 [ 68.]
 [ 63.]
 [ 70.]
 [ 88.]
 [ 75.]
 [ 70.]
 [ 67.]
 [ 67.]
 [ 67.]
 [110.]
 [ 85.]
 [ 92.]
 [112.]
 [ 96.]
 [ 84.]
 [ 90.]
 [ 86.]
 [ 52.]
 [ 84.]
 [ 79.]
 [ 82.]]
```

In [15]:
```python
# variance after replacing with the mode value
var_mode = f_mode.var()
print(var_mode)
```

```
1486.29231585061
```

In [16]:
```python
# Find lowest variance from mean, median and mode imputing strategies
min(var_mean,var_median,var_mode)
```

Out[16]:
```
1455.511639831812
```

Conclusion:

Among mean, median and mode imputation methods mean have a lowest variance which is 1455.511639831812. Whereas in with median and mode method, we get 1457.2982752960781 and 1486.29231585061 variance, respectively.

Variance means how far the given data is spread out. Zero variance means data values in sample are identical. When data is very close to the mean, variance become very low. So, when we replace the NaN with mean value, we get lowest variance.

Problem 3

In [17]:
```python
from sklearn import datasets
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
```

```
In [18]: iris = datasets.load_iris()
         iris
```

```
Out[18]: {'data': array([[5.1, 3.5, 1.4, 0.2],
        [4.9, 3. , 1.4, 0.2],
        [4.7, 3.2, 1.3, 0.2],
        [4.6, 3.1, 1.5, 0.2],
        [5. , 3.6, 1.4, 0.2],
        [5.4, 3.9, 1.7, 0.4],
        [4.6, 3.4, 1.4, 0.3],
        [5. , 3.4, 1.5, 0.2],
        [4.4, 2.9, 1.4, 0.2],
        [4.9, 3.1, 1.5, 0.1],
        [5.4, 3.7, 1.5, 0.2],
        [4.8, 3.4, 1.6, 0.2],
        [4.8, 3. , 1.4, 0.1],
        [4.3, 3. , 1.1, 0.1],
        [5.8, 4. , 1.2, 0.2],
        [5.7, 4.4, 1.5, 0.4],
        [5.4, 3.9, 1.3, 0.4],
        [5.1, 3.5, 1.4, 0.3],
        [5.7, 3.8, 1.7, 0.3],
        [5.1, 3.8, 1.5, 0.3],
        [5.4, 3.4, 1.7, 0.2],
        [5.1, 3.7, 1.5, 0.4],
        [4.6, 3.6, 1. , 0.2],
        [5.1, 3.3, 1.7, 0.5],
        [4.8, 3.4, 1.9, 0.2],
        [5. , 3. , 1.6, 0.2],
        [5. , 3.4, 1.6, 0.4],
        [5.2, 3.5, 1.5, 0.2],
        [5.2, 3.4, 1.4, 0.2],
        [4.7, 3.2, 1.6, 0.2],
        [4.8, 3.1, 1.6, 0.2],
        [5.4, 3.4, 1.5, 0.4],
        [5.2, 4.1, 1.5, 0.1],
        [5.5, 4.2, 1.4, 0.2],
        [4.9, 3.1, 1.5, 0.2],
        [5. , 3.2, 1.2, 0.2],
        [5.5, 3.5, 1.3, 0.2],
        [4.9, 3.6, 1.4, 0.1],
        [4.4, 3. , 1.3, 0.2],
        [5.1, 3.4, 1.5, 0.2],
        [5. , 3.5, 1.3, 0.3],
        [4.5, 2.3, 1.3, 0.3],
        [4.4, 3.2, 1.3, 0.2],
        [5. , 3.5, 1.6, 0.6],
        [5.1, 3.8, 1.9, 0.4],
        [4.8, 3. , 1.4, 0.3],
        [5.1, 3.8, 1.6, 0.2],
        [4.6, 3.2, 1.4, 0.2],
        [5.3, 3.7, 1.5, 0.2],
        [5. , 3.3, 1.4, 0.2],
        [7. , 3.2, 4.7, 1.4],
        [6.4, 3.2, 4.5, 1.5],
        [6.9, 3.1, 4.9, 1.5],
        [5.5, 2.3, 4. , 1.3],
        [6.5, 2.8, 4.6, 1.5],
        [5.7, 2.8, 4.5, 1.3],
        [6.3, 3.3, 4.7, 1.6],
        [4.9, 2.4, 3.3, 1. ],
        [6.6, 2.9, 4.6, 1.3],
        [5.2, 2.7, 3.9, 1.4],
        [5. , 2. , 3.5, 1. ],
        [5.9, 3. , 4.2, 1.5],
        [6. , 2.2, 4. , 1. ],
        [6.1, 2.9, 4.7, 1.4],
        [5.6, 2.9, 3.6, 1.3],
        [6.7, 3.1, 4.4, 1.4],
        [5.6, 3. , 4.5, 1.5],
        [5.8, 2.7, 4.1, 1. ],
        [6.2, 2.2, 4.5, 1.5],
        [5.6, 2.5, 3.9, 1.1],
        [5.9, 3.2, 4.8, 1.8],
        [6.1, 2.8, 4. , 1.3],
        [6.3, 2.5, 4.9, 1.5],
        [6.1, 2.8, 4.7, 1.2],
        [6.4, 2.9, 4.3, 1.3],
        [6.6, 3. , 4.4, 1.4],
        [6.8, 2.8, 4.8, 1.4],
        [6.7, 3. , 5. , 1.7],
        [6. , 2.9, 4.5, 1.5],
        [5.7, 2.6, 3.5, 1. ],
        [5.5, 2.4, 3.8, 1.1],
        [5.5, 2.4, 3.7, 1. ],
        [5.8, 2.7, 3.9, 1.2],
        [6. , 2.7, 5.1, 1.6],
        [5.4, 3. , 4.5, 1.5],
        [6. , 3.4, 4.5, 1.6],
        [6.7, 3.1, 4.7, 1.5],
        [6.3, 2.3, 4.4, 1.3],
        [5.6, 3. , 4.1, 1.3],
        [5.5, 2.5, 4. , 1.3],
        [5.5, 2.6, 4.4, 1.2],
        [6.1, 3. , 4.6, 1.4],
        [5.8, 2.6, 4. , 1.2],
        [5. , 2.3, 3.3, 1. ],
        [5.6, 2.7, 4.2, 1.3],
        [5.7, 3. , 4.2, 1.2],
        [5.7, 2.9, 4.2, 1.3],
        [6.2, 2.9, 4.3, 1.3],
        [5.1, 2.5, 3. , 1.1],
        [5.7, 2.8, 4.1, 1.3],
        [6.3, 3.3, 6. , 2.5],
        [5.8, 2.7, 5.1, 1.9],
        [7.1, 3. , 5.9, 2.1],
        [6.3, 2.9, 5.6, 1.8],
        [6.5, 3. , 5.8, 2.2],
        [7.6, 3. , 6.6, 2.1],
        [4.9, 2.5, 4.5, 1.7],
        [7.3, 2.9, 6.3, 1.8],
        [6.7, 2.5, 5.8, 1.8],
        [7.2, 3.6, 6.1, 2.5],
        [6.5, 3.2, 5.1, 2. ],
        [6.4, 2.7, 5.3, 1.9],
        [6.8, 3. , 5.5, 2.1],
        [5.7, 2.5, 5. , 2. ],
        [5.8, 2.8, 5.1, 2.4],
        [6.4, 3.2, 5.3, 2.3],
        [6.5, 3. , 5.5, 1.8],
        [7.7, 3.8, 6.7, 2.2],
        [7.7, 2.6, 6.9, 2.3],
        [6. , 2.2, 5. , 1.5],
        [6.9, 3.2, 5.7, 2.3],
        [5.6, 2.8, 4.9, 2. ],
        [7.7, 2.8, 6.7, 2. ],
        [6.3, 2.7, 4.9, 1.8],
        [6.7, 3.3, 5.7, 2.1],
        [7.2, 3.2, 6. , 1.8],
        [6.2, 2.8, 4.8, 1.8],
        [6.1, 3. , 4.9, 1.8],
        [6.4, 2.8, 5.6, 2.1],
        [7.2, 3. , 5.8, 1.6],
        [7.4, 2.8, 6.1, 1.9],
        [7.9, 3.8, 6.4, 2. ],
        [6.4, 2.8, 5.6, 2.2],
        [6.3, 2.8, 5.1, 1.5],
        [6.1, 2.6, 5.6, 1.4],
        [7.7, 3. , 6.1, 2.3],
        [6.3, 3.4, 5.6, 2.4],
        [6.4, 3.1, 5.5, 1.8],
        [6. , 3. , 4.8, 1.8],
        [6.9, 3.1, 5.4, 2.1],
        [6.7, 3.1, 5.6, 2.4],
        [6.9, 3.1, 5.1, 2.3],
        [5.8, 2.7, 5.1, 1.9],
        [6.8, 3.2, 5.9, 2.3],
        [6.7, 3.3, 5.7, 2.5],
        [6.7, 3. , 5.2, 2.3],
        [6.3, 2.5, 5. , 1.9],
        [6.5, 3. , 5.2, 2. ],
        [6.2, 3.4, 5.4, 2.3],
        [5.9, 3. , 5.1, 1.8]]),
 'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]),
 'frame': None,
 'target_names': array(['setosa', 'versicolor', 'virginica'], dtype='<U10'),
 'DESCR': '.. _iris_dataset:\n\nIris plants dataset\n--------------------\n\n**Data Set Characteristi
cs:**\n\n    :Number of Instances: 150 (50 in each of three classes)\n    :Number of Attributes: 4 nu
meric, predictive attributes and the class\n    :Attribute Information:\n        - sepal length in cm
\n        - sepal width in cm\n        - petal length in cm\n        - petal width in cm\n        - c
lass:\n                - Iris-Setosa\n                - Iris-Versicolour\n                - Iris-Virg
inica\n                \n    :Summary Statistics:\n\n    ============== ==== ==== ======= ===== =====
===============\n                    Min  Max   Mean    SD   Class Correlation\n    ============== ==
== ==== ======= ===== ====================\n    sepal length:   4.3  7.9   5.84   0.83    0.7826\n
sepal width:    2.0  4.4   3.05   0.43   -0.4194\n    petal length:   1.0  6.9   3.76   1.76    0.949
0  (high!)\n    petal width:    0.1  2.5   1.20   0.76     0.9565  (high!)\n    ============== ==== ==
== ======= ===== ====================\n\n    :Missing Attribute Values: None\n    :Class Distributio
n: 33.3% for each of 3 classes.\n    :Creator: R.A. Fisher\n    :Donor: Michael Marshall (MARSHALL%PL
U@io.arc.nasa.gov)\n    :Date: July, 1988\n\nThe famous Iris database, first used by Sir R.A. Fisher.
The dataset is taken\nfrom Fisher\'s paper. Note that it\'s the same as in R, but not as in the UCI\n
Machine Learning Repository, which has two wrong data points.\n\nThis is perhaps the best known datab
ase to be found in the\npattern recognition literature.  Fisher\'s paper is a classic in the field an
d\nis referenced frequently to this day.  (See Duda & Hart, for example.)  The\ndata set contains 3 c
lasses of 50 instances each, where each class refers to a\ntype of iris plant.  One class is linearly
separable from the other 2; the\nlatter are NOT linearly separable from each other.\n\n.. topic:: Ref
erences\n\n   - Fisher, R.A. "The use of multiple measurements in taxonomic problems"\n     Annual Eu
genics, 7, Part II, 179-188 (1936); also in "Contributions to\n     Mathematical Statistics" (John Wi
ley, NY, 1950).\n   - Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysis.\n
  (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.\n   - Dasarathy, B.V. (1980) "Nos
ing Around the Neighborhood: A New System\n     Structure and Classification Rule for Recognition in
Partially Exposed\n     Environments".  IEEE Transactions on Pattern Analysis and Machine\n     Intel
ligence, Vol. PAMI-2, No. 1, 67-71.\n   - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".  IE
EE Transactions\n     on Information Theory, May 1972, 431-433.\n   - See also: 1988 MLC Proceedings,
54-64.  Cheeseman et al"s AUTOCLASS II\n     conceptual clustering system finds 3 classes in the dat
a.\n   - Many, many more ...',
 'feature_names': ['sepal length (cm)',
  'sepal width (cm)',
  'petal length (cm)',
  'petal width (cm)'],
 'filename': 'C:\\anaconda3\\lib\\site-packages\\sklearn\\datasets\\data\\iris.csv'}
```

```
In [19]: iris_df = pd.DataFrame(data=iris.data, columns=["SepalLength", "SepalWidth", "PetalLength", "PetalWidt
         h"])
         iris_df
```

Out[19]:

	SepalLength	SepalWidth	PetalLength	PetalWidth
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

```
In [20]: # orginal feature variance
         print(iris_df.SepalLength.var())
         print(iris_df.SepalWidth.var())
         print(iris_df.PetalLength.var())
         print(iris_df.PetalWidth.var())

0.6856935123042505
0.1899794183445188
3.1162778523489942
0.5810062639821029
```

```
In [21]: scaler = StandardScaler()
         scaler.fit(iris_df.values)
         scaled_data = scaler.transform(iris_df.values)
```

```
In [22]: pca_4 = PCA(n_components=4)
         pca_4.fit(scaled_data)
         pca_4_transformed = pca_4.transform(scaled_data)
```

```
In [23]: pca_4.explained_variance_ratio_
Out[23]: array([0.72962445, 0.22850762, 0.03668922, 0.00517871])
```

```
In [24]: pca_4.explained_variance_ratio_*100
Out[24]: array([72.96244541, 22.85076179,  3.66892189,  0.51787091])
```

```
In [25]: sum(pca_4.explained_variance_ratio_*100)
Out[25]: 99.99999999999999
```

```
In [26]: np.cumsum(pca_4.explained_variance_ratio_*100)
Out[26]: array([ 72.96244541,  95.8132072 ,  99.48212909, 100.        ])
```

```
In [27]: print("Variance explained by the First principal component = ",
         np.cumsum(pca_4.explained_variance_ratio_*100)[0])
         print("Variance explained by the First 2 principal component = ",
         np.cumsum(pca_4.explained_variance_ratio_*100)[1])
         print("Variance explained by the First 3 principal component = ",
         np.cumsum(pca_4.explained_variance_ratio_*100)[2])
         print("Variance explained by the First 4 principal component = ",
         np.cumsum(pca_4.explained_variance_ratio_*100)[3])

Variance explained by the First principal component =  72.96244541329986
Variance explained by the First 2 principal component =  95.81320720000164
Variance explained by the First 3 principal component =  99.48212908928451
Variance explained by the First 4 principal component =  99.99999999999999
```

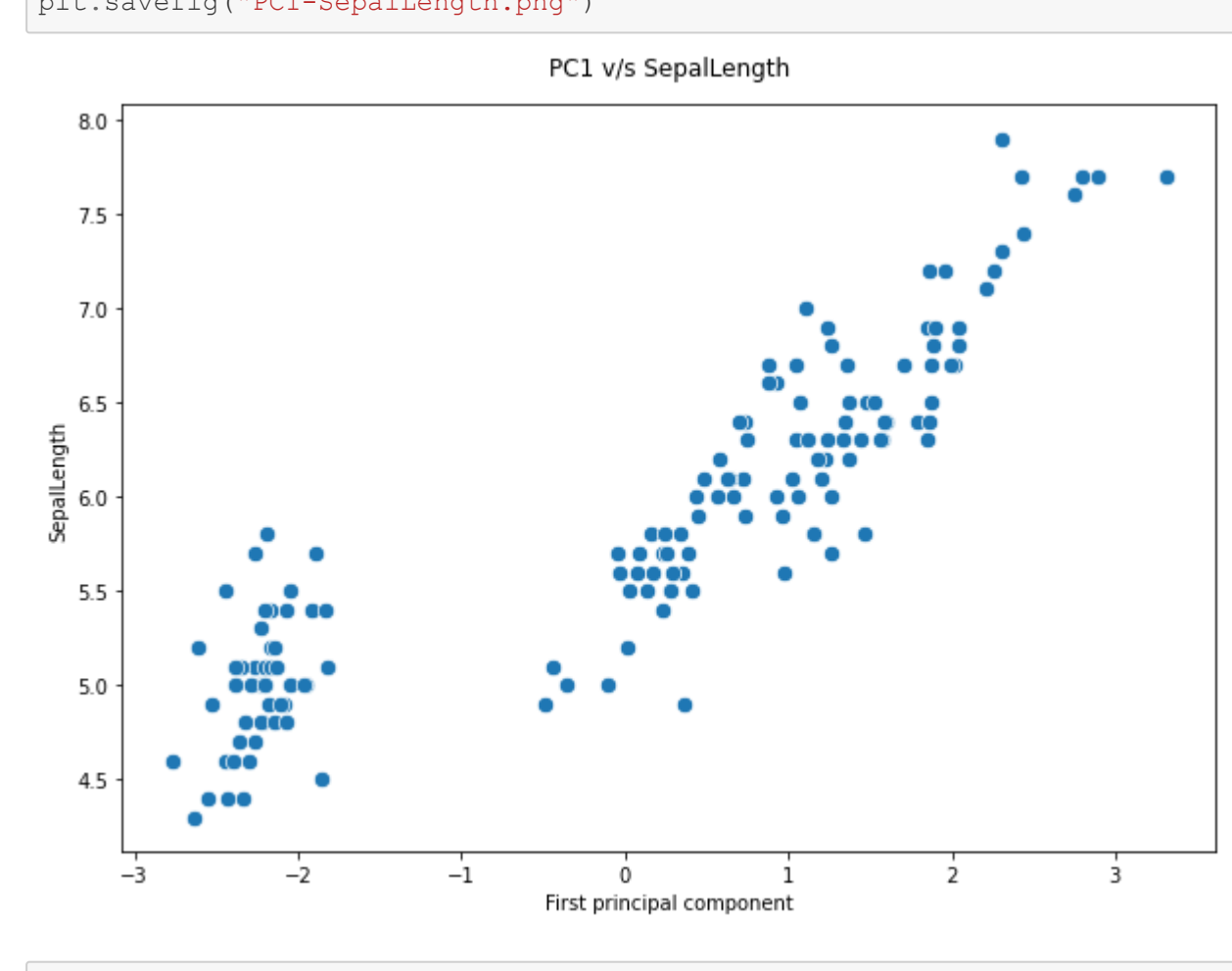
Conclusion:

Percentage Variance explained by the First principal component = 72.96244541 % Percentage Variance explained by the Second principal component = 22.85076179 % Percentage Variance explained by the Third principal component = 3.66892189 % Percentage Variance explained by the Fourth principal component = 0.51787091 %

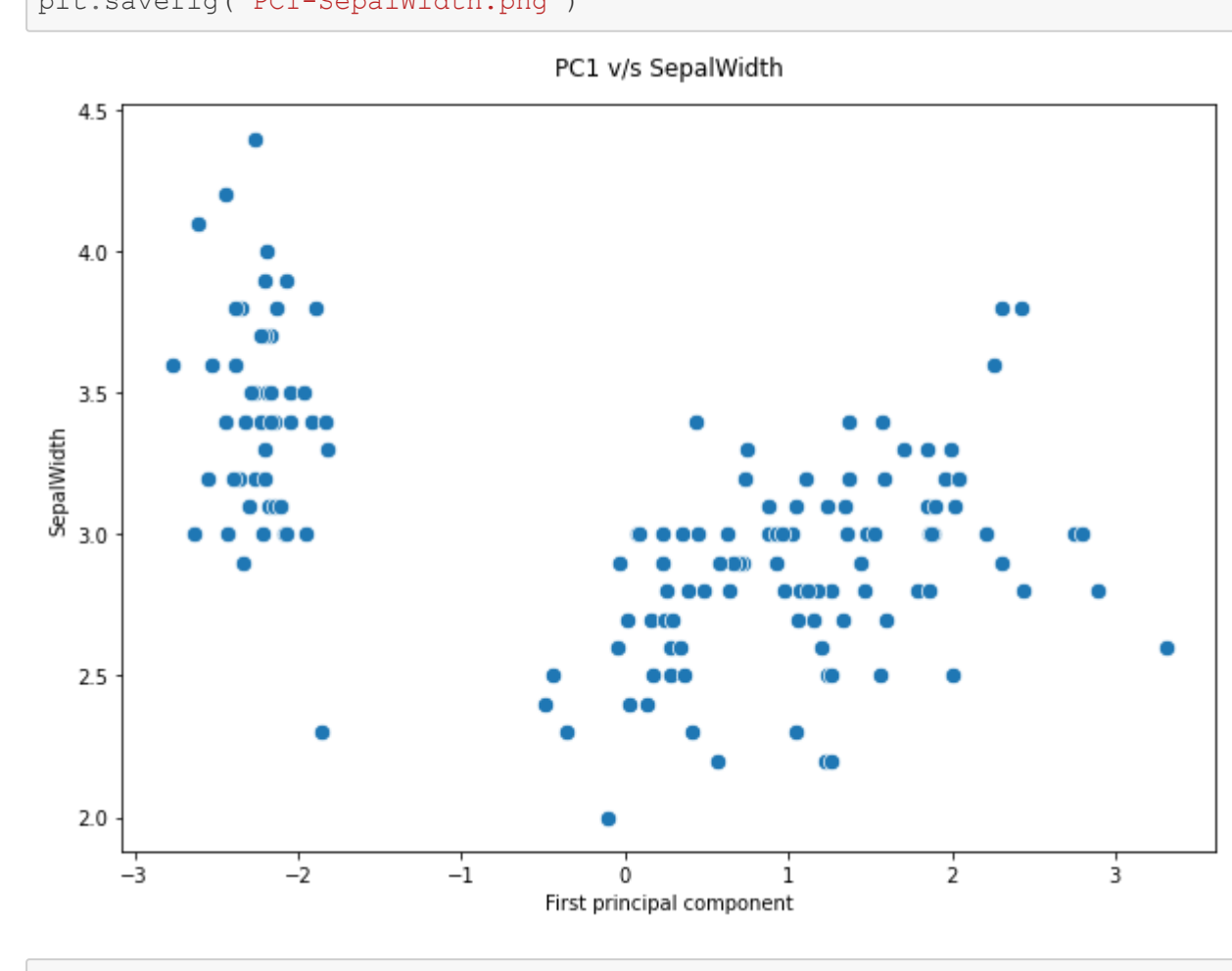
The first and second component acquire about 95.8132072% of the variability in the dataset. So, we can reduce the dimensionality with 2 components.

Problem 4

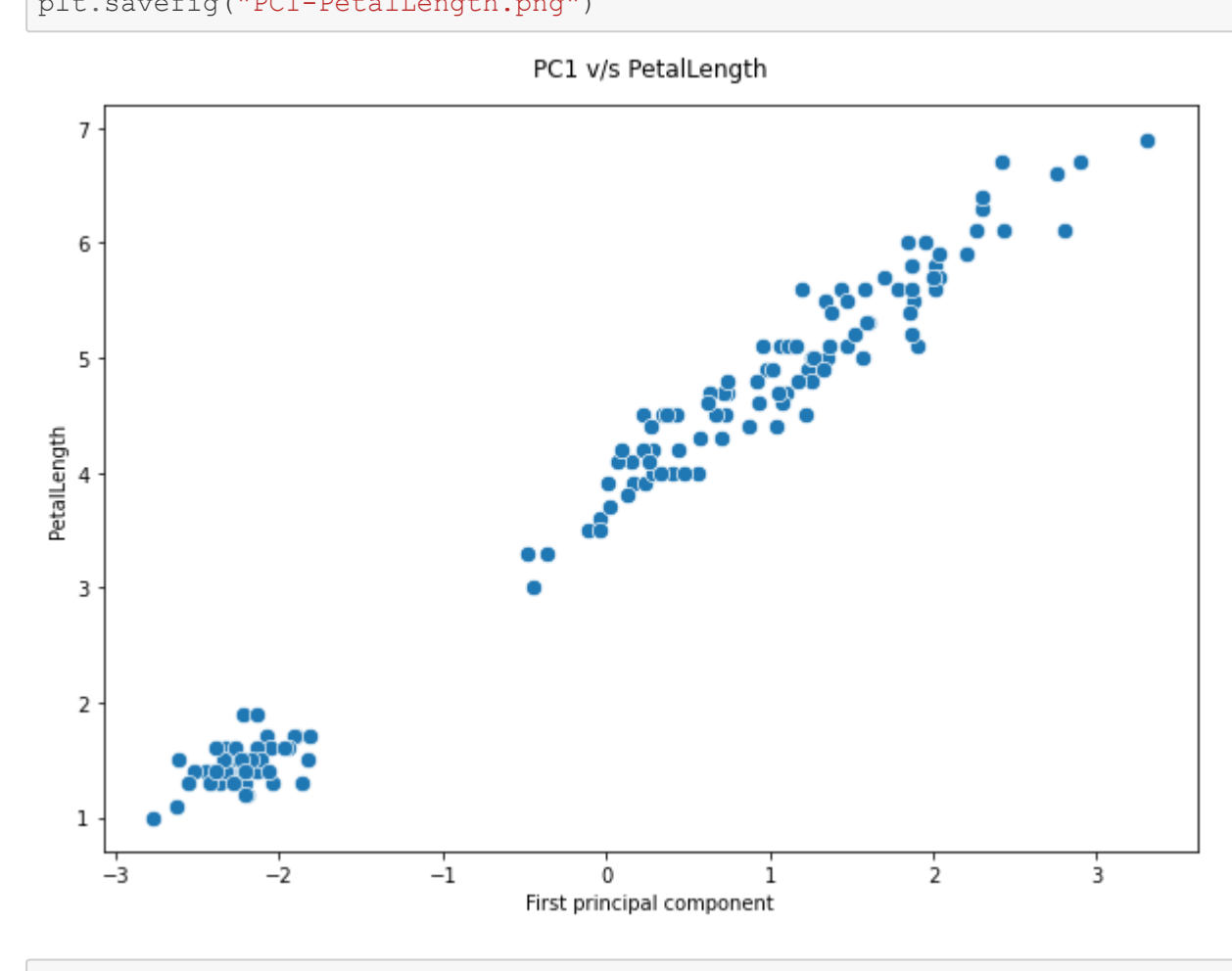
```
In [28]: plt.figure(figsize=(10,7))
         sns.scatterplot(x=pca_4_transformed[:,0], y=iris_df.SepalLength, s=70)
         plt.title("PC1 v/s SepalLength",pad=15)
         plt.xlabel("First principal component")
         plt.ylabel("SepalLength")
         plt.savefig("PC1-SepalLength.png")
```



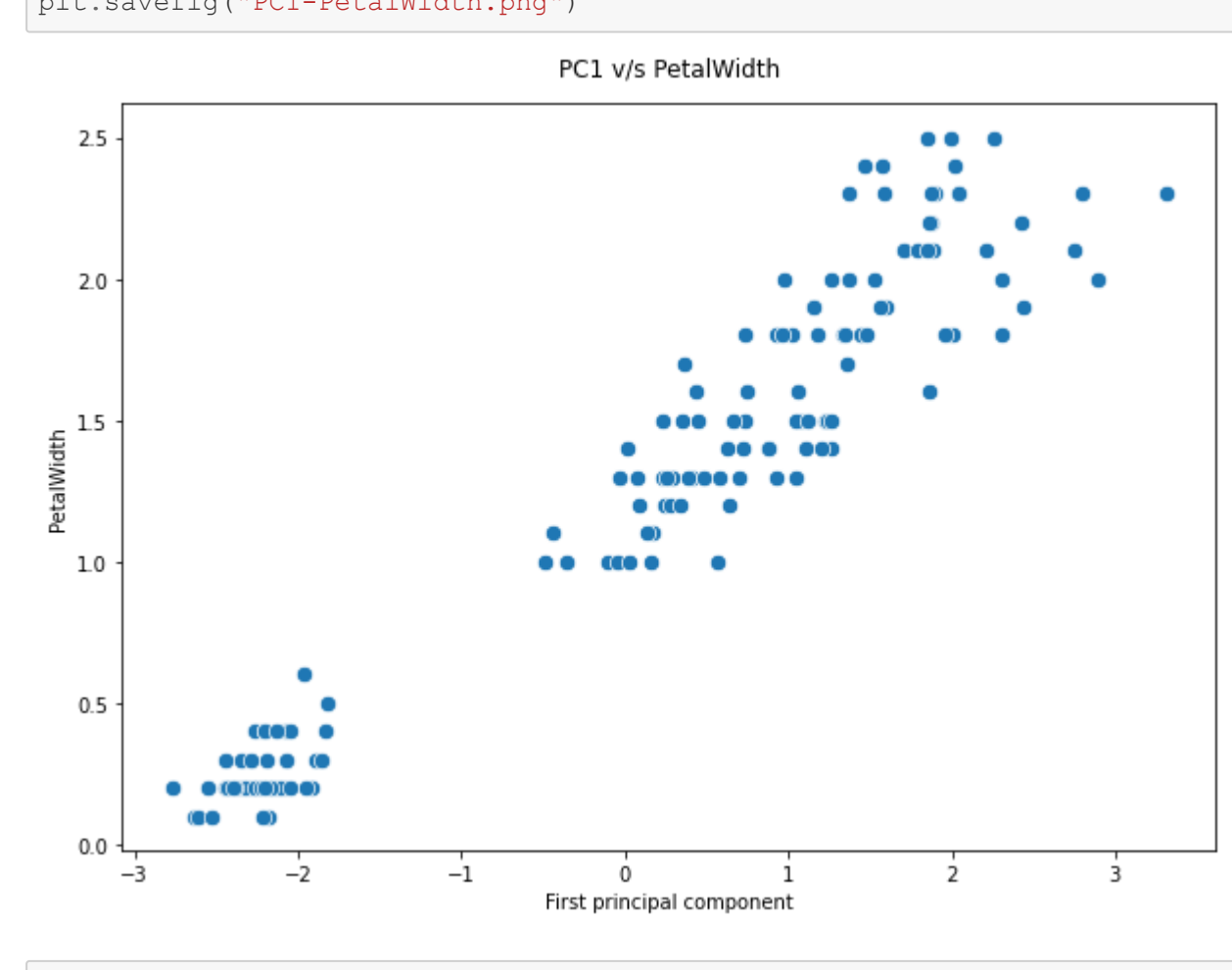

```
In [29]: plt.figure(figsize=(10,7))
         sns.scatterplot(x=pca_4_transformed[:,0], y=iris_df.SepalWidth, s=70)
         plt.title("PC1 v/s SepalWidth",pad=15)
         plt.xlabel("First principal component")
         plt.ylabel("SepalWidth")
         plt.savefig("PC1-SepalWidth.png")
```




```
In [30]: plt.figure(figsize=(10,7))
         sns.scatterplot(x=pca_4_transformed[:,0], y=iris_df.PetalLength, s=70)
         plt.title("PC1 v/s PetalLength",pad=15)
         plt.xlabel("First principal component")
         plt.ylabel("PetalLength")
         plt.savefig("PC1-PetalLength.png")
```




```
In [31]: plt.figure(figsize=(10,7))
         sns.scatterplot(x=pca_4_transformed[:,0], y=iris_df.PetalWidth, s=70)
         plt.title("PC1 v/s PetalWidth",pad=15)
         plt.xlabel("First principal component")
         plt.ylabel("PetalWidth")
         plt.savefig("PC1-PetalWidth.png")
```




```
In [32]: from scipy import spatial
         cosine_similarity = 1 - spatial.distance.cosine(iris_df["PetalLength"], iris_df["PetalWidth"])
         cosine_similarity
Out[32]: 0.9835496832996021
```

```
In [33]: print(iris_df["PetalLength"].corr(iris_df["PetalWidth"]))

0.9628654314027961
```

```
In [34]: pcadf = pd.DataFrame(pca_4_transformed, columns = ['PC1','PC2','PC3','PC4'])
         pcadf
```

Out[34]:

	PC1	PC2	PC3	PC4
0	-2.264703	0.480027	-0.127706	-0.024168
1	-2.080961	-0.674134	-0.234609	-0.103007
2	-2.364229	-0.341908	0.044201	-0.028377
3	-2.299384	-0.597395	0.091290	0.065956
4	-2.389842	0.646835	0.015738	0.035923
...	...	...	...	...
145	1.870503	0.386966	0.256274	-0.389257
146	1.564580	-0.896687	-0.026371	-0.220192
147	1.521170	0.269069	0.180178	-0.119171
148	1.372788	1.011254	0.933395	-0.026129
149	0.960656	-0.024332	0.528249	0.163078

150 rows × 4 columns

```
In [35]: # Find correlation coefficient
         print(pcadf["PC1"].corr(iris_df["SepalLength"]))
         print(pcadf["PC1"].corr(iris_df["SepalWidth"]))
         print(pcadf["PC1"].corr(iris_df["PetalLength"]))
         print(pcadf["PC1"].corr(iris_df["PetalWidth"]))

0.8901687648612951
-0.46014270644790795
0.9915551834193606
0.9649789606692489
```

Conclusion:

With the visual inspection of all the graph, I can say that the graph between principal component 1 (PC1) v/s PetalLength and principal component 1 v/s PetalWidth have more similarities or have a closer relationship.

Petal Length and Petal Width have a closure relation with PC1. Moreover, cosine similarity between both the feature is 0.9835496832996021 (near to 1).

Correlation between Petal Length and PC1: 0.9915551834193606 Correlation between Petal Width and PC1: 0.9649789606692489

Above mention correlation coefficient of both the features represent the closer relationship with PC1 which also portray in the visual inspection of graphs.

Problem 5

```
In [36]: # orginal feature variance
         print(iris_df.SepalLength.var())
         print(iris_df.SepalWidth.var())
         print(iris_df.PetalLength.var())
         print(iris_df.PetalWidth.var())

0.6856935123042505
0.1899794183445188
3.1162778523489942
0.5810062639821029
```

```
In [37]: # total variance of the original features
         num=[iris_df.SepalLength.var(),iris_df.SepalWidth.var(),iris_df.PetalLength.var(),iris_df.PetalWidth.va
         r()]
         sum(num)
Out[37]: 4.572957046979867
```

```
In [38]: # Variance of the four eigenvectors
         print(pcadf.PC1.var())
         print(pcadf.PC2.var())
         print(pcadf.PC3.var())
         print(pcadf.PC4.var())

2.9380850501999936
0.9201649041624869
0.14774182104494807
0.02085386217646228
```

```
In [39]: # total variance of the original features
         num1=[pcadf.PC1.var(),pcadf.PC2.var(),pcadf.PC3.var(),pcadf.PC4.var()]
         sum(num1)
Out[39]: 4.026845637583891
```

```
In [40]: series = pd.Series(num1)
         cumsum = series.cumsum()
         cumsum
Out[40]: 0    2.938085
         1    3.858250
         2    4.005992
         3    4.026846
         dtype: float64
```

```
In [41]: # We should select first two components
         (cumsum[1]/sum(num1))*100
Out[41]: 95.81320720000163
```

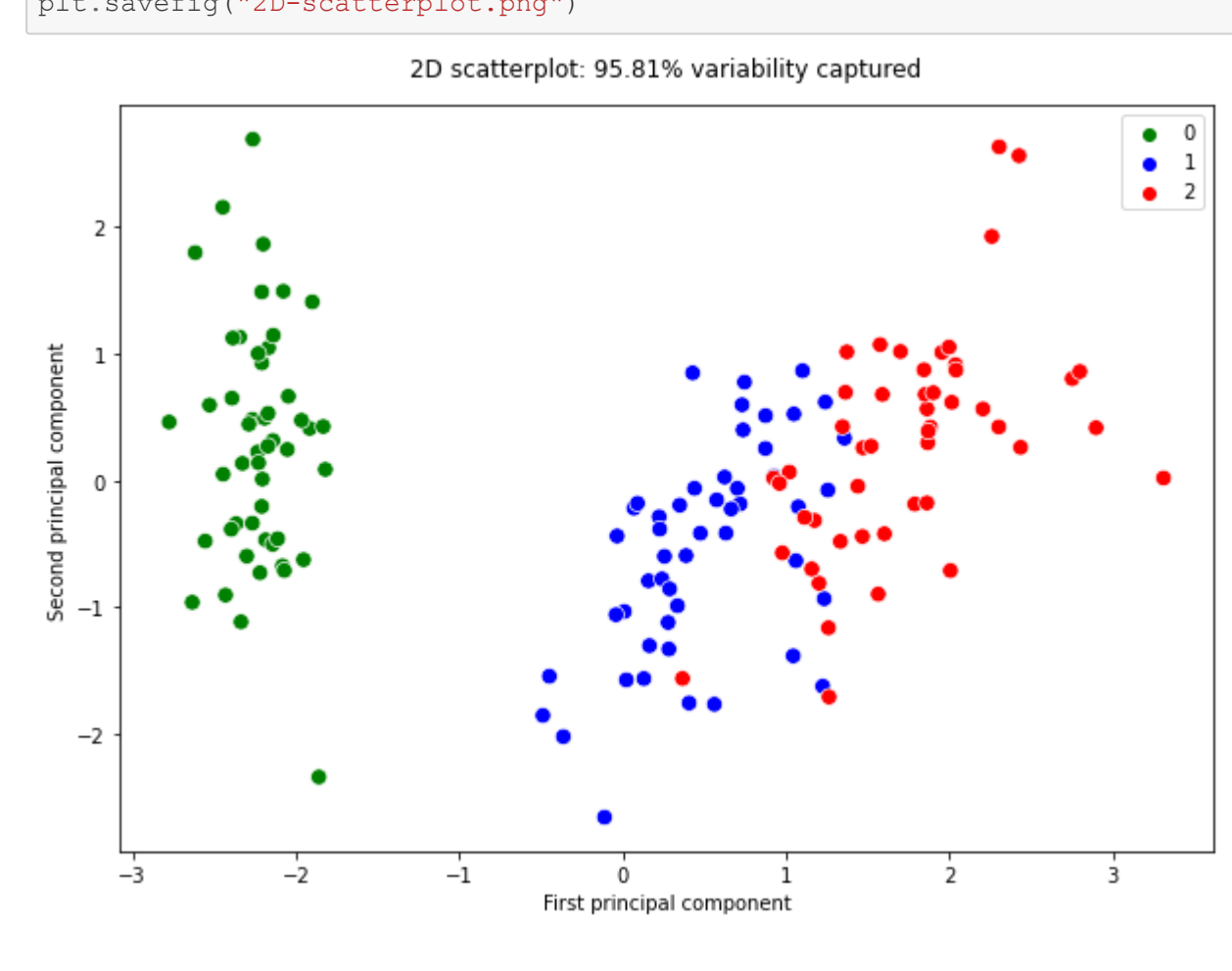
Method - 2

```
In [42]: # Finding principal component with n_component = 0.95
         pca_95 = PCA(n_components=0.95)
         pca_95.fit(scaled_data)
         pca_95_transformed = pca_95.transform(scaled_data)
```

```
In [43]: pca_95.explained_variance_ratio_*100
Out[43]: array([72.96244541, 22.85076179])
```

```
In [44]: np.cumsum(pca_95.explained_variance_ratio_*100)
Out[44]: array([72.96244541, 95.8132072 ])
```

```
In [45]: plt.figure(figsize=(10,7))
         sns.scatterplot(x=pca_95_transformed[:,0], y=pca_95_transformed[:,1], s=70, hue=iris.target, palette=[
         'green','blue','red'])
         plt.title("2D scatterplot: 95.81% variability captured",pad=15)
         plt.xlabel("First principal component")
         plt.ylabel("Second principal component")
         plt.savefig("2D-scatterplot.png")
```



Conclusion:

Total variance of the original feature is 4.572957046979867 but after applying the PCA the variance become 4.026845637583891.

Variance of each principal components (Eigenvectors):

Variance of principal component 1 = 2.9380850501999936 Variance of principal component 2 = 0.9201649041624869 Variance of principal component 3 = 0.14774182104494807 Variance of principal component 4 = 0.02085386217646228 Total = 4.026845637583891

The Sum of first and second principal component variance is 3.858250 which is 95.81% of the total variance. In other words, first two component retain the 95.81% variance. So, we can reduce the dimensionality with 2 components.

PCA will select the number of components such that the amount of variance that needs to be explained. For example, if n_components=0.95, the algorithm will select the number of components while preserving 95% of the variability in the data.

```
In [ ]:
```