Case study on Numpy and Pandas

November 22, 2024

1 NumPy Problems

[114]:
```
import numpy as np
from numpy.linalg import norm
import csv
```

[115]:
```
with open('auto-mpg.csv', 'r') as file:
    reader = csv.reader(file)
    data = list(reader)
```

[116]:
```
# Headers
data[0]
```

[116]:
```
['mpg',
 'cylinders',
 'displacement',
 'horsepower',
 'weight',
 'acceleration',
 'model year',
 'origin',
 'car name']
```

1.1 1. Basic Array Operations

- Convert the mpg column into a NumPy array and calculate:The mean, median, and standard deviation of mpg.
- The number of cars with mpg greater than 25.

[117]:
```
mpg_data = np.array([data[i][0] for i in range(1,len(data))]).astype('float64')
mpg_data
```

[117]:
```
array([18. , 15. , 18. , 16. , 17. , 15. , 14. , 14. , 14. , 15. , 15. ,
       14. , 15. , 14. , 24. , 22. , 18. , 21. , 27. , 26. , 25. , 24. ,
       25. , 26. , 21. , 10. , 10. , 11. ,  9. , 27. , 28. , 25. , 25. ,
       19. , 16. , 17. , 19. , 18. , 14. , 14. , 14. , 14. , 12. , 13. ,
       13. , 18. , 22. , 19. , 18. , 23. , 28. , 30. , 30. , 31. , 35. ,
       27. , 26. , 24. , 25. , 23. , 20. , 21. , 13. , 14. , 15. , 14. ,
```

```
17. , 11. , 13. , 12. , 13. , 19. , 15. , 13. , 13. , 14. , 18. ,
22. , 21. , 26. , 22. , 28. , 23. , 28. , 27. , 13. , 14. , 13. ,
14. , 15. , 12. , 13. , 13. , 14. , 13. , 12. , 13. , 18. , 16. ,
18. , 18. , 23. , 26. , 11. , 12. , 13. , 12. , 18. , 20. , 21. ,
22. , 18. , 19. , 21. , 26. , 15. , 16. , 29. , 24. , 20. , 19. ,
15. , 24. , 20. , 11. , 20. , 21. , 19. , 15. , 31. , 26. , 32. ,
25. , 16. , 16. , 18. , 16. , 13. , 14. , 14. , 14. , 29. , 26. ,
26. , 31. , 32. , 28. , 24. , 26. , 24. , 26. , 31. , 19. , 18. ,
15. , 15. , 16. , 15. , 16. , 14. , 17. , 16. , 15. , 18. , 21. ,
20. , 13. , 29. , 23. , 20. , 23. , 24. , 25. , 24. , 18. , 29. ,
19. , 23. , 23. , 22. , 25. , 33. , 28. , 25. , 25. , 26. , 27. ,
17.5, 16. , 15.5, 14.5, 22. , 22. , 24. , 22.5, 29. , 24.5, 29. ,
33. , 20. , 18. , 18.5, 17.5, 29.5, 32. , 28. , 26.5, 20. , 13. ,
19. , 19. , 16.5, 16.5, 13. , 13. , 13. , 31.5, 30. , 36. , 25.5,
33.5, 17.5, 17. , 15.5, 15. , 17.5, 20.5, 19. , 18.5, 16. , 15.5,
15.5, 16. , 29. , 24.5, 26. , 25.5, 30.5, 33.5, 30. , 30.5, 22. ,
21.5, 21.5, 43.1, 36.1, 32.8, 39.4, 36.1, 19.9, 19.4, 20.2, 19.2,
20.5, 20.2, 25.1, 20.5, 19.4, 20.6, 20.8, 18.6, 18.1, 19.2, 17.7,
18.1, 17.5, 30. , 27.5, 27.2, 30.9, 21.1, 23.2, 23.8, 23.9, 20.3,
17. , 21.6, 16.2, 31.5, 29.5, 21.5, 19.8, 22.3, 20.2, 20.6, 17. ,
17.6, 16.5, 18.2, 16.9, 15.5, 19.2, 18.5, 31.9, 34.1, 35.7, 27.4,
25.4, 23. , 27.2, 23.9, 34.2, 34.5, 31.8, 37.3, 28.4, 28.8, 26.8,
33.5, 41.5, 38.1, 32.1, 37.2, 28. , 26.4, 24.3, 19.1, 34.3, 29.8,
31.3, 37. , 32.2, 46.6, 27.9, 40.8, 44.3, 43.4, 36.4, 30. , 44.6,
40.9, 33.8, 29.8, 32.7, 23.7, 35. , 23.6, 32.4, 27.2, 26.6, 25.8,
23.5, 30. , 39.1, 39. , 35.1, 32.3, 37. , 37.7, 34.1, 34.7, 34.4,
29.9, 33. , 34.5, 33.7, 32.4, 32.9, 31.6, 28.1, 30.7, 25.4, 24.2,
22.4, 26.6, 20.2, 17.6, 28. , 27. , 34. , 31. , 29. , 27. , 24. ,
23. , 36. , 37. , 31. , 38. , 36. , 36. , 36. , 34. , 38. , 32. ,
38. , 25. , 38. , 26. , 22. , 32. , 36. , 27. , 27. , 44. , 32. ,
28. , 31. ])
```

[118]:
```python
print("Mean of mpg = ", np.mean(mpg_data))
print("Median of mpg = ", np.median(mpg_data))
print("Standard deviation of mpg = ", np.std(mpg_data))
print("Number of cars with mpg>25 = ", np.count_nonzero(mpg_data>25))
```

```
Mean of mpg =  23.514572864321607
Median of mpg =  23.0
Standard deviation of mpg =  7.806159061274433
Number of cars with mpg>25 =  158
```

1.2 2. Filtering

Using NumPy, filter all cars with more than 6 cylinders.
Return the corresponding car name as a list.

[119]:
```
cylinder_data = np.array([data[i][1] for i in range(1,len(data))]).astype('int')
car_name_data = np.array([data[i][8] for i in range(1,len(data))])
```

[120]:
```
car_with_more_than_6_cylinders = [car_name_data[i] for i in␣
↪range(len(car_name_data)) if cylinder_data[i]>6 ]
print("Cars with with more than 6 cylinders = ",␣
↪len(car_with_more_than_6_cylinders))
```

```
Cars with with more than 6 cylinders =  103
```

[121]:
```
car_with_more_than_6_cylinders
```

[121]:
```
['chevrolet chevelle malibu',
 'buick skylark 320',
 'plymouth satellite',
 'amc rebel sst',
 'ford torino',
 'ford galaxie 500',
 'chevrolet impala',
 'plymouth fury iii',
 'pontiac catalina',
 'amc ambassador dpl',
 'dodge challenger se',
 "plymouth 'cuda 340",
 'chevrolet monte carlo',
 'buick estate wagon (sw)',
 'ford f250',
 'chevy c20',
 'dodge d200',
 'hi 1200d',
 'chevrolet impala',
 'pontiac catalina brougham',
 'ford galaxie 500',
 'plymouth fury iii',
 'dodge monaco (sw)',
 'ford country squire (sw)',
 'pontiac safari (sw)',
 'chevrolet impala',
 'pontiac catalina',
 'plymouth fury iii',
 'ford galaxie 500',
 'amc ambassador sst',
 'mercury marquis',
 'buick lesabre custom',
 'oldsmobile delta 88 royale',
 'chrysler newport royal',
 'amc matador (sw)',
 'chevrolet chevelle concours (sw)',
```

```
'ford gran torino (sw)',
'plymouth satellite custom (sw)',
'buick century 350',
'amc matador',
'chevrolet malibu',
'ford gran torino',
'dodge coronet custom',
'mercury marquis brougham',
'chevrolet caprice classic',
'ford ltd',
'plymouth fury gran sedan',
'chrysler new yorker brougham',
'buick electra 225 custom',
'amc ambassador brougham',
'chevrolet impala',
'ford country',
'plymouth custom suburb',
'oldsmobile vista cruiser',
'chevrolet monte carlo s',
'pontiac grand prix',
'dodge dart custom',
'oldsmobile omega',
'ford gran torino',
'buick century luxus (sw)',
'dodge coronet custom (sw)',
'ford gran torino (sw)',
'amc matador (sw)',
'pontiac catalina',
'chevrolet bel air',
'plymouth grand fury',
'ford ltd',
'chevrolet monza 2+2',
'ford mustang ii',
'chevrolet chevelle malibu classic',
'dodge coronet brougham',
'amc matador',
'ford gran torino',
'plymouth volare premier v8',
'cadillac seville',
'chevy c10',
'ford f108',
'dodge d100',
'chevrolet caprice classic',
'oldsmobile cutlass supreme',
'dodge monaco brougham',
'mercury cougar brougham',
'pontiac grand prix lj',
```

```
'chevrolet monte carlo landau',
'chrysler cordoba',
'ford thunderbird',
'oldsmobile cutlass salon brougham',
'dodge diplomat',
'mercury monarch ghia',
'chevrolet monte carlo landau',
'ford futura',
'dodge magnum xe',
'chevrolet caprice classic',
'ford ltd landau',
'mercury grand marquis',
'dodge st. regis',
'buick estate wagon (sw)',
'ford country squire (sw)',
'chevrolet malibu classic (sw)',
'chrysler lebaron town @ country (sw)',
'cadillac eldorado',
'oldsmobile cutlass salon brougham',
'oldsmobile cutlass ls']
```

1.3 3. Statistical Analysis

Compute the 25th, 50th, and 75th percentiles of the weight column using NumPy.

[122]:
```
weight_data = np.array([data[i][4] for i in range(1,len(data))]).astype('int')
```

[123]:
```
print("25th percentile of weight column = ", np.percentile(weight_data,25))
print("50th percentile of weight column = ", np.percentile(weight_data,50))
print("75th percentile of weight column = ", np.percentile(weight_data,75))
```

```
25th percentile of weight column =  2223.75
50th percentile of weight column =  2803.5
75th percentile of weight column =  3608.0
```

1.4 4. Array Manipulation

Convert the acceleration column into a NumPy array and normalize its values (scale between 0 and 1).

[124]:
```
acceleration_data = np.array([data[i][5] for i in range(1,len(data))]).
↪astype('float64')
acceleration_data
```

[124]:
```
array([12. , 11.5, 11. , 12. , 10.5, 10. ,  9. ,  8.5, 10. ,  8.5, 10. ,
        8. ,  9.5, 10. , 15. , 15.5, 15.5, 16. , 14.5, 20.5, 17.5, 14.5,
       17.5, 12.5, 15. , 14. , 15. , 13.5, 18.5, 14.5, 15.5, 14. , 19. ,
       13. , 15.5, 15.5, 15.5, 15.5, 12. , 11.5, 13.5, 13. , 11.5, 12. ,
```

```
12. , 13.5, 19. , 15. , 14.5, 14. , 14. , 19.5, 14.5, 19. , 18. ,
19. , 20.5, 15.5, 17. , 23.5, 19.5, 16.5, 12. , 12. , 13.5, 13. ,
11.5, 11. , 13.5, 13.5, 12.5, 13.5, 12.5, 14. , 16. , 14. , 14.5,
18. , 19.5, 18. , 16. , 17. , 14.5, 15. , 16.5, 13. , 11.5, 13. ,
14.5, 12.5, 11.5, 12. , 13. , 14.5, 11. , 11. , 11. , 16.5, 18. ,
16. , 16.5, 16. , 21. , 14. , 12.5, 13. , 12.5, 15. , 19. , 19.5,
16.5, 13.5, 18.5, 14. , 15.5, 13. ,  9.5, 19.5, 15.5, 14. , 15.5,
11. , 14. , 13.5, 11. , 16.5, 17. , 16. , 17. , 19. , 16.5, 21. ,
17. , 17. , 18. , 16.5, 14. , 14.5, 13.5, 16. , 15.5, 16.5, 15.5,
14.5, 16.5, 19. , 14.5, 15.5, 14. , 15. , 15.5, 16. , 16. , 16. ,
21. , 19.5, 11.5, 14. , 14.5, 13.5, 21. , 18.5, 19. , 19. , 15. ,
13.5, 12. , 16. , 17. , 16. , 18.5, 13.5, 16.5, 17. , 14.5, 14. ,
17. , 15. , 17. , 14.5, 13.5, 17.5, 15.5, 16.9, 14.9, 17.7, 15.3,
13. , 13. , 13.9, 12.8, 15.4, 14.5, 17.6, 17.6, 22.2, 22.1, 14.2,
17.4, 17.7, 21. , 16.2, 17.8, 12.2, 17. , 16.4, 13.6, 15.7, 13.2,
21.9, 15.5, 16.7, 12.1, 12. , 15. , 14. , 18.5, 14.8, 18.6, 15.5,
16.8, 12.5, 19. , 13.7, 14.9, 16.4, 16.9, 17.7, 19. , 11.1, 11.4,
12.2, 14.5, 14.5, 16. , 18.2, 15.8, 17. , 15.9, 16.4, 14.1, 14.5,
12.8, 13.5, 21.5, 14.4, 19.4, 18.6, 16.4, 15.5, 13.2, 12.8, 19.2,
18.2, 15.8, 15.4, 17.2, 17.2, 15.8, 16.7, 18.7, 15.1, 13.2, 13.4,
11.2, 13.7, 16.5, 14.2, 14.7, 14.5, 14.8, 16.7, 17.6, 14.9, 15.9,
13.6, 15.7, 15.8, 14.9, 16.6, 15.4, 18.2, 17.3, 18.2, 16.6, 15.4,
13.4, 13.2, 15.2, 14.9, 14.3, 15. , 13. , 14. , 15.2, 14.4, 15. ,
20.1, 17.4, 24.8, 22.2, 13.2, 14.9, 19.2, 14.7, 16. , 11.3, 12.9,
13.2, 14.7, 18.8, 15.5, 16.4, 16.5, 18.1, 20.1, 18.7, 15.8, 15.5,
17.5, 15. , 15.2, 17.9, 14.4, 19.2, 21.7, 23.7, 19.9, 21.8, 13.8,
17.3, 18. , 15.3, 11.4, 12.5, 15.1, 14.3, 17. , 15.7, 16.4, 14.4,
12.6, 12.9, 16.9, 16.4, 16.1, 17.8, 19.4, 17.3, 16. , 14.9, 16.2,
20.7, 14.2, 15.8, 14.4, 16.8, 14.8, 18.3, 20.4, 19.6, 12.6, 13.8,
15.8, 19. , 17.1, 16.6, 19.6, 18.6, 18. , 16.2, 16. , 18. , 16.4,
20.5, 15.3, 18.2, 17.6, 14.7, 17.3, 14.5, 14.5, 16.9, 15. , 15.7,
16.2, 16.4, 17. , 14.5, 14.7, 13.9, 13. , 17.3, 15.6, 24.6, 11.6,
18.6, 19.4])
```

[125]:
```
normalised_acceleration_data = acceleration_data/norm(acceleration_data)
print("Normalised acceleration values = \n", normalised_acceleration_data)
```

```
Normalised acceleration values =
 [0.03804627 0.03646101 0.03487575 0.03804627 0.03329049 0.03170523
 0.0285347  0.02694944 0.03170523 0.02694944 0.03170523 0.02536418
 0.03011997 0.03170523 0.04755784 0.0491431  0.0491431  0.05072836
 0.04597258 0.06499572 0.05548415 0.04597258 0.05548415 0.03963153
 0.04755784 0.04438732 0.04755784 0.04280206 0.05865467 0.04597258
 0.0491431  0.04438732 0.06023993 0.0412168  0.0491431  0.0491431
 0.0491431  0.0491431  0.03804627 0.03646101 0.04280206 0.0412168
 0.03646101 0.03804627 0.03804627 0.04280206 0.06023993 0.04755784
 0.04597258 0.04438732 0.04438732 0.06182519 0.04597258 0.06023993
 0.05706941 0.06023993 0.06499572 0.0491431  0.05389889 0.07450728
```

0.06182519	0.05231362	0.03804627	0.03804627	0.04280206	0.0412168
0.03646101	0.03487575	0.04280206	0.04280206	0.03963153	0.04280206
0.03963153	0.04438732	0.05072836	0.04438732	0.04597258	0.05706941
0.06182519	0.05706941	0.05072836	0.05389889	0.04597258	0.04755784
0.05231362	0.0412168	0.03646101	0.0412168	0.04597258	0.03963153
0.03646101	0.03804627	0.0412168	0.04597258	0.03487575	0.03487575
0.03487575	0.05231362	0.05706941	0.05072836	0.05231362	0.05072836
0.06658098	0.04438732	0.03963153	0.0412168	0.03963153	0.04755784
0.06023993	0.06182519	0.05231362	0.04280206	0.05865467	0.04438732
0.0491431	0.0412168	0.03011997	0.06182519	0.0491431	0.04438732
0.0491431	0.03487575	0.04438732	0.04280206	0.03487575	0.05231362
0.05389889	0.05072836	0.05389889	0.06023993	0.05231362	0.06658098
0.05389889	0.05389889	0.05706941	0.05231362	0.04438732	0.04597258
0.04280206	0.05072836	0.0491431	0.05231362	0.0491431	0.04597258
0.05231362	0.06023993	0.04597258	0.0491431	0.04438732	0.04755784
0.0491431	0.05072836	0.05072836	0.05072836	0.06658098	0.06182519
0.03646101	0.04438732	0.04597258	0.04280206	0.06658098	0.05865467
0.06023993	0.06023993	0.04755784	0.04280206	0.03804627	0.05072836
0.05389889	0.05072836	0.05865467	0.04280206	0.05231362	0.05389889
0.04597258	0.04438732	0.05389889	0.04755784	0.05389889	0.04597258
0.04280206	0.05548415	0.0491431	0.05358183	0.04724079	0.05611825
0.048509	0.0412168	0.0412168	0.04407027	0.04058269	0.04882605
0.04597258	0.0558012	0.0558012	0.0703856	0.07006855	0.04502142
0.0551671	0.05611825	0.06658098	0.05136247	0.0564353	0.03868038
0.05389889	0.05199657	0.04311911	0.04977721	0.0418509	0.06943445
0.0491431	0.05294773	0.03836332	0.03804627	0.04755784	0.04438732
0.05865467	0.04692374	0.05897172	0.0491431	0.05326478	0.03963153
0.06023993	0.04343616	0.04724079	0.05199657	0.05358183	0.05611825
0.06023993	0.0351928	0.03614396	0.03868038	0.04597258	0.04597258
0.05072836	0.05770351	0.05009426	0.05389889	0.05041131	0.05199657
0.04470437	0.04597258	0.04058269	0.04280206	0.06816624	0.04565553
0.06150814	0.05897172	0.05199657	0.0491431	0.0418509	0.04058269
0.06087404	0.05770351	0.05009426	0.04882605	0.05453299	0.05453299
0.05009426	0.05294773	0.05928877	0.04787489	0.0418509	0.042485
0.03550985	0.04343616	0.05231362	0.04502142	0.04660668	0.04597258
0.04692374	0.05294773	0.0558012	0.04724079	0.05041131	0.04311911
0.04977721	0.05009426	0.04724079	0.05263068	0.04882605	0.05770351
0.05485004	0.05770351	0.05263068	0.04882605	0.042485	0.0418509
0.04819195	0.04724079	0.04533847	0.04755784	0.0412168	0.04438732
0.04819195	0.04565553	0.04755784	0.06372751	0.0551671	0.07862896
0.0703856	0.0418509	0.04724079	0.06087404	0.04660668	0.05072836
0.03582691	0.04089974	0.0418509	0.04660668	0.05960583	0.0491431
0.05199657	0.05231362	0.05738646	0.06372751	0.05928877	0.05009426
0.0491431	0.05548415	0.04755784	0.04819195	0.05675236	0.04565553
0.06087404	0.06880034	0.07514139	0.0630934	0.06911739	0.04375321
0.05485004	0.05706941	0.048509	0.03614396	0.03963153	0.04787489
0.04533847	0.05389889	0.04977721	0.05199657	0.04565553	0.03994859
0.04089974	0.05358183	0.05199657	0.05104542	0.0564353	0.06150814

```
 0.05485004 0.05072836 0.04724079 0.05136247 0.06562982 0.04502142
 0.05009426 0.04565553 0.05326478 0.04692374 0.05802057 0.06467866
 0.06214224 0.03994859 0.04375321 0.05009426 0.06023993 0.05421594
 0.05263068 0.06214224 0.05897172 0.05706941 0.05136247 0.05072836
 0.05706941 0.05199657 0.06499572 0.048509   0.05770351 0.0558012
 0.04660668 0.05485004 0.04597258 0.04597258 0.05358183 0.04755784
 0.04977721 0.05136247 0.05199657 0.05389889 0.04597258 0.04660668
 0.04407027 0.0412168  0.05485004 0.04946015 0.07799486 0.03677806
 0.05897172 0.06150814]
```

[126]:

```python
normalised_acceleration_data_min_max = (acceleration_data-np.
 ↪min(acceleration_data))/(np.max(acceleration_data)-np.min(acceleration_data))
print("Normalised acceleration values using min-max scaling = \n",␣
 ↪normalised_acceleration_data_min_max)
```

```
Normalised acceleration values using min-max scaling =
 [0.23809524 0.20833333 0.17857143 0.23809524 0.14880952 0.11904762
 0.05952381 0.0297619  0.11904762 0.0297619  0.11904762 0.
 0.08928571 0.11904762 0.41666667 0.44642857 0.44642857 0.47619048
 0.38690476 0.74404762 0.56547619 0.38690476 0.56547619 0.26785714
 0.41666667 0.35714286 0.41666667 0.32738095 0.625      0.38690476
 0.44642857 0.35714286 0.6547619  0.29761905 0.44642857 0.44642857
 0.44642857 0.44642857 0.23809524 0.20833333 0.32738095 0.29761905
 0.20833333 0.23809524 0.23809524 0.32738095 0.6547619  0.41666667
 0.38690476 0.35714286 0.35714286 0.68452381 0.38690476 0.6547619
 0.5952381  0.6547619  0.74404762 0.44642857 0.53571429 0.92261905
 0.68452381 0.50595238 0.23809524 0.23809524 0.32738095 0.29761905
 0.20833333 0.17857143 0.32738095 0.32738095 0.26785714 0.32738095
 0.26785714 0.35714286 0.47619048 0.35714286 0.38690476 0.5952381
 0.68452381 0.5952381  0.47619048 0.53571429 0.38690476 0.41666667
 0.50595238 0.29761905 0.20833333 0.29761905 0.38690476 0.26785714
 0.20833333 0.23809524 0.29761905 0.38690476 0.17857143 0.17857143
 0.17857143 0.50595238 0.5952381  0.47619048 0.50595238 0.47619048
 0.77380952 0.35714286 0.26785714 0.29761905 0.26785714 0.41666667
 0.6547619  0.68452381 0.50595238 0.32738095 0.625      0.35714286
 0.44642857 0.29761905 0.08928571 0.68452381 0.44642857 0.35714286
 0.44642857 0.17857143 0.35714286 0.32738095 0.17857143 0.50595238
 0.53571429 0.47619048 0.53571429 0.6547619  0.50595238 0.77380952
 0.53571429 0.53571429 0.5952381  0.50595238 0.35714286 0.38690476
 0.32738095 0.47619048 0.44642857 0.50595238 0.44642857 0.38690476
 0.50595238 0.6547619  0.38690476 0.44642857 0.35714286 0.41666667
 0.44642857 0.47619048 0.47619048 0.47619048 0.77380952 0.68452381
 0.20833333 0.35714286 0.38690476 0.32738095 0.77380952 0.625
 0.6547619  0.6547619  0.41666667 0.32738095 0.23809524 0.47619048
 0.53571429 0.47619048 0.625      0.32738095 0.50595238 0.53571429
 0.38690476 0.35714286 0.53571429 0.41666667 0.53571429 0.38690476
 0.32738095 0.56547619 0.44642857 0.5297619  0.41071429 0.57738095
 0.43452381 0.29761905 0.29761905 0.35119048 0.28571429 0.44047619
```

```
0.38690476 0.57142857 0.57142857 0.8452381  0.83928571 0.36904762
0.55952381 0.57738095 0.77380952 0.48809524 0.58333333 0.25
0.53571429 0.5        0.33333333 0.45833333 0.30952381 0.82738095
0.44642857 0.51785714 0.24404762 0.23809524 0.41666667 0.35714286
0.625      0.4047619  0.63095238 0.44642857 0.52380952 0.26785714
0.6547619  0.33928571 0.41071429 0.5        0.5297619  0.57738095
0.6547619  0.18452381 0.20238095 0.25       0.38690476 0.38690476
0.47619048 0.60714286 0.46428571 0.53571429 0.4702381  0.5
0.36309524 0.38690476 0.28571429 0.32738095 0.80357143 0.38095238
0.67857143 0.63095238 0.5        0.44642857 0.30952381 0.28571429
0.66666667 0.60714286 0.46428571 0.44047619 0.54761905 0.54761905
0.46428571 0.51785714 0.63690476 0.42261905 0.30952381 0.32142857
0.19047619 0.33928571 0.50595238 0.36904762 0.39880952 0.38690476
0.4047619  0.51785714 0.57142857 0.41071429 0.4702381  0.33333333
0.45833333 0.46428571 0.41071429 0.51190476 0.44047619 0.60714286
0.55357143 0.60714286 0.51190476 0.44047619 0.32142857 0.30952381
0.42857143 0.41071429 0.375      0.41666667 0.29761905 0.35714286
0.42857143 0.38095238 0.41666667 0.7202381  0.55952381 1.
0.8452381  0.30952381 0.41071429 0.66666667 0.39880952 0.47619048
0.19642857 0.29166667 0.30952381 0.39880952 0.64285714 0.44642857
0.5        0.50595238 0.60119048 0.7202381  0.63690476 0.46428571
0.44642857 0.56547619 0.41666667 0.42857143 0.58928571 0.38095238
0.66666667 0.81547619 0.93452381 0.70833333 0.82142857 0.3452381
0.55357143 0.5952381  0.43452381 0.20238095 0.26785714 0.42261905
0.375      0.53571429 0.45833333 0.5        0.38095238 0.27380952
0.29166667 0.5297619  0.5        0.48214286 0.58333333 0.67857143
0.55357143 0.47619048 0.41071429 0.48809524 0.75595238 0.36904762
0.46428571 0.38095238 0.52380952 0.4047619  0.61309524 0.73809524
0.69047619 0.27380952 0.3452381  0.46428571 0.6547619  0.54166667
0.51190476 0.69047619 0.63095238 0.5952381  0.48809524 0.47619048
0.5952381  0.5        0.74404762 0.43452381 0.60714286 0.57142857
0.39880952 0.55357143 0.38690476 0.38690476 0.5297619  0.41666667
0.45833333 0.48809524 0.5        0.53571429 0.38690476 0.39880952
0.35119048 0.29761905 0.55357143 0.45238095 0.98809524 0.21428571
0.63095238 0.67857143]
```

1.5 5. Broadcasting

Increase all horsepower values by 10% and store the updated values in a new NumPy array. Handle missing data (if any) by replacing it with the mean of the column before applying the increase.

[127]:
```
horse_power_data = np.array([data[i][3] for i in range(1,len(data))])
horse_power_data
```

[127]:
```
array(['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', '?', '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', '?', '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', '?', '67', '62',
       '132', '100', '88', '?', '72', '84', '84', '92', '110', '84', '58',
       '64', '60', '67', '65', '62', '68', '63', '65', '65', '74', '?',
       '75', '75', '100', '74', '80', '76', '116', '120', '110', '105',
       '88', '85', '88', '88', '88', '85', '84', '90', '92', '?', '74',
       '68', '68', '63', '70', '88', '75', '70', '67', '67', '67', '110',
       '85', '92', '112', '96', '84', '90', '86', '52', '84', '79', '82'],
      dtype='<U3')
```

[128]:

```python
mean_sum = 0
count = 0
for i in range(len(horse_power_data)):
    if(horse_power_data[i] != '?'):
        count += 1
        mean_sum += horse_power_data[i].astype('int')
mean = mean_sum/count

for i in range(len(horse_power_data)):
    if(horse_power_data[i] == '?'):
```

```
        horse_power_data[i] = np.round(mean)

horse_power_data = horse_power_data.astype('int')

horse_power_data
```

[128]:
```
array([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, 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, 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,  67,  62, 132, 100,  88, 104,  72,
        84,  84,  92, 110,  84,  58,  64,  60,  67,  65,  62,  68,  63,
        65,  65,  74, 104,  75,  75, 100,  74,  80,  76, 116, 120, 110,
       105,  88,  85,  88,  88,  88,  85,  84,  90,  92, 104,  74,  68,
        68,  63,  70,  88,  75,  70,  67,  67,  67, 110,  85,  92, 112,
        96,  84,  90,  86,  52,  84,  79,  82])
```

[129]:
```
increased_horse_power_data = horse_power_data+horse_power_data*0.1
print("Increased horsepower (+10%) = \n", increased_horse_power_data)
```

```
Increased horsepower (+10%) =
 [143.   181.5 165.   165.   154.   217.8 242.   236.5 247.5 209.   187.   176.
 165.   247.5 104.5 104.5 106.7  93.5  96.8  50.6  95.7  99.  104.5 124.3
  99.  236.5 220.  231.  212.3  96.8  99.  104.5 114.4 110.  115.5 110.
  96.8 110.  181.5 192.5 168.3 165.  198.  187.  192.5 121.   79.2 110.
  96.8  94.6  99.   77.   83.6  71.5  75.9  66.   77.  104.5  88.   59.4
  99.   94.6 181.5 192.5 165.  168.3 165.  228.8 170.5 176.  209.  106.7
```

```
165.  143.  154.  165.  123.2  83.6  95.7  75.9  94.6 101.2 106.7  88.
 96.8 192.5 165.  159.5 150.7 165.  217.8 165.  173.8 165.  236.5 247.5
192.5 115.5 110.  110.   96.8 104.5  50.6 165.  183.7 187.  198.  110.
 96.8  79.2 103.4  99.   93.5 117.7  99.  159.5 253.   53.9  82.5 100.1
123.2 165.  121.  134.2 198.  104.5 114.4 110.  110.   73.7  88.   71.5
 82.5 110.  121.  115.5 154.  165.  165.  154.  165.   91.3  73.7  85.8
 57.2  67.1  82.5  82.5  82.5 106.7 102.3  73.7 104.5 115.5  79.2  79.2
187.  159.5 165.  162.8 121.  115.5 121.  104.5 121.  121.  141.9  82.5
 91.3 110.   85.8 105.6  78.1 106.7 106.7  77.   99.  104.5  96.8 107.8
126.5  58.3  94.6  89.1 101.2  86.9  91.3 154.  165.  132.  167.2 110.
115.5  89.1  99.   57.2  66.   77.   58.3 110.   85.8 121.  104.5  78.1
 77.   82.5  79.2 112.2 165.   96.8 118.8 132.  198.  159.5 143.  165.
 74.8  88.   63.8 105.6  77.  159.5 121.  159.5 143.  121.  115.5 110.
107.8 198.  187.  209.  163.9  85.8  96.8  82.5  97.9  69.3  91.3  73.7
 85.8 106.7 121.  121.   52.8  72.6  57.2  77.   66.  121.  154.  152.9
115.5 104.5  93.5  96.8 110.   99.  115.5  93.5 121.  132.  159.5 181.5
152.9 154.   74.8 104.5 106.7  82.5 104.5 115.5  93.5 106.7 113.3 137.5
126.5 146.3  78.1  74.8 126.5  93.5  96.8  99.  121.  143.  141.9 151.8
148.5 170.5 156.2 137.5 165.   78.1  71.5  88.   88.   84.7 137.5  78.1
 99.   77.   77.   71.5  75.9  99.  126.5 126.5  99.   83.6  66.   77.
 71.5  99.   96.8  99.   99.   85.8  99.   82.5 101.2  82.5  71.5 115.5
 71.5  52.8  52.8  73.7  73.7  73.7 114.4  73.7  68.2 145.2 110.   96.8
114.4  79.2  92.4  92.4 101.2 121.   92.4  63.8  70.4  66.   73.7  71.5
 68.2  74.8  69.3  71.5  71.5  81.4 114.4  82.5  82.5 110.   81.4  88.
 83.6 127.6 132.  121.  115.5  96.8  93.5  96.8  96.8  96.8  93.5  92.4
 99.  101.2 114.4  81.4  74.8  74.8  69.3  77.   96.8  82.5  77.   73.7
 73.7  73.7 121.   93.5 101.2 123.2 105.6  92.4  99.   94.6  57.2  92.4
 86.9  90.2]
```

1.6 6. Boolean Indexing

Find the average displacement of cars with an origin of 2 (Europe) using NumPy indexing.

```
[130]: displacement_data = np.array([data[i][2] for i in range(1,len(data))])
       origin_data = np.array([data[i][7] for i in range(1,len(data))]).astype('int')
```

```
[131]: origin_boolean_mask = origin_data == 2
```

```
[132]: print("Average displacement of cars with origin 2 (Europe) = ", np.round(np.
        ↪mean(displacement_data[origin_boolean_mask].astype('float64')),2))
```

```
Average displacement of cars with origin 2 (Europe) =  109.14
```

1.7 7. Matrix Operations

Create a 2D NumPy array containing the columns mpg, horsepower, and weight.
Compute the dot product of this matrix with a given vector [1, 0.5, -0.2].

[133]:
```
d2_array = np.array([mpg_data, horse_power_data, weight_data])
print(d2_array)
print(d2_array.shape)
```

```
[[  18.    15.    18. …   32.    28.    31.]
 [ 130.   165.   150. …   84.    79.    82.]
 [3504. 3693. 3436. … 2295. 2625. 2720.]]
(3, 398)
```

[134]:
```
vector = np.array([1, 0.5, -0.2])
print(vector)
print(vector.shape)
```

```
[ 1.   0.5 -0.2]
(3,)
```

[135]:
```
dot_product = np.dot(vector,d2_array)
print("Dot product = ", dot_product)
print(dot_product.shape)
```

```
Dot product =  [-617.8 -641.1 -594.2 -595.6 -602.8 -754.2 -746.8 -740.9 -758.5
-660.
 -612.6 -627.8 -662.2 -490.7 -402.9 -497.1 -488.3 -453.9 -355.  -318.
 -465.9 -417.  -402.5 -364.3 -463.6 -805.5 -765.2 -760.4 -840.9 -355.
 -379.8 -373.1 -332.2 -457.8 -619.3 -598.8 -597.4 -589.6 -745.3 -791.3
 -740.3 -730.2 -889.  -851.2 -927.5 -519.4 -423.6 -587.4 -565.8 -378.
 -351.6 -349.8 -345.  -291.1 -253.1 -309.8 -330.  -384.1 -360.2 -400.8
 -416.6 -381.2 -759.3 -775.5 -737.  -735.3 -642.4 -811.6 -809.9 -799.2
 -776.4 -398.5 -688.4 -741.6 -775.8 -726.4 -512.6 -442.2 -531.3 -377.3
 -414.  -383.6 -429.7 -364.8 -349.  -719.5 -645.4 -712.1 -725.9 -665.4
 -879.4 -804.8 -780.6 -758.4 -826.5 -865.7 -663.7 -553.7 -589.6 -521.
 -542.2 -510.3 -341.  -913.4 -885.7 -832.8 -797.8 -489.8 -391.8 -423.2
 -406.8 -361.8 -400.5 -419.9 -382.  -728.9 -724.6 -319.9 -370.1 -450.9
 -498.6 -589.8 -453.  -480.4 -631.8 -552.9 -502.  -511.2 -602.2 -325.5
 -424.2 -302.7 -445.9 -690.2 -655.4 -652.1 -742.2 -851.8 -802.4 -843.6
 -762.4 -373.3 -333.1 -395.  -272.8 -338.1 -359.5 -360.1 -385.7 -425.3
 -405.7 -335.5 -586.3 -621.3 -635.4 -580.6 -832.6 -800.5 -808.6 -843.4
 -709.4 -710.9 -676.  -691.5 -531.8 -569.2 -556.3 -367.7 -463.3 -512.8
 -456.4 -468.4 -384.1 -436.5 -530.3 -323.4 -578.2 -468.3 -524.4 -518.
 -451.7 -299.5 -421.8 -378.5 -443.4 -385.5 -371.9 -755.5 -747.  -716.9
 -752.5 -574.6 -596.1 -537.9 -549.5 -352.  -378.3 -323.4 -299.5 -660.2
 -657.8 -655.5 -573.6 -300.  -331.  -365.5 -450.5 -559.  -700.  -591.
 -513.  -687.5 -769.5 -725.5 -696.  -663.  -343.5 -361.  -300.  -386.5
 -320.5 -686.  -740.  -740.  -779.  -631.5 -612.  -657.  -637.5 -738.
 -732.5 -754.5 -776.5 -320.  -479.5 -389.5 -481.  -348.2 -340.  -333.5
 -368.5 -492.5 -443.5 -467.5 -329.9 -290.9 -338.2 -339.6 -293.9 -598.1
 -657.6 -624.3 -635.3 -563.  -530.3 -474.9 -615.5 -577.6 -602.9 -550.7
 -650.4 -603.9 -593.3 -588.8 -553.4 -728.5 -367.  -437.  -384.3 -377.6
 -434.4 -473.3 -504.7 -408.6 -494.2 -548.5 -479.9 -599.3 -331.  -363.5
```

```
 -570.  -535.7 -511.7 -587.8 -596.4 -686.  -662.9 -705.5 -680.3 -777.6
 -724.3 -639.3 -694.5 -317.6 -328.4 -307.3 -466.6 -642.1 -694.5 -575.3
 -615.1 -370.8 -360.5 -339.7 -354.2 -460.6 -432.7 -455.7 -432.7 -349.3
 -325.5 -356.9 -334.1 -462.6 -503.6 -531.3 -612.1 -364.3 -467.4 -439.6
 -403.8 -383.3 -342.9 -479.6 -348.7 -348.7 -399.6 -520.1 -586.5 -291.9
 -274.1 -361.7 -308.2 -483.3 -410.3 -421.  -505.4 -389.6 -428.8 -458.4
 -452.2 -466.5 -405.  -282.9 -304.  -286.9 -347.2 -325.5 -341.3 -328.9
 -376.8 -342.1 -413.6 -368.  -377.5 -370.8 -400.1 -440.1 -458.4 -577.9
 -563.3 -496.6 -501.8 -605.6 -665.9 -547.8 -632.9 -449.  -457.  -401.
 -441.5 -434.  -475.  -503.  -532.  -323.  -334.  -329.  -355.5 -354.
 -352.  -367.5 -380.  -321.5 -327.5 -327.5 -509.  -522.5 -445.  -489.
 -453.  -396.  -518.  -488.  -356.  -385.  -457.5 -472. ]
(398,)
```

1.8 8. Sorting

Use NumPy to sort the cars by model_year in descending order and display the first five car names.

[136]:
```
model_year_data = np.array([data[i][6] for i in range(1,len(data))])
model_year_car_name_data = np.array([model_year_data, car_name_data])
```

[137]:
```
desc_sorted_model_year_car_name_data = np.sort(np.
 ↪column_stack(([model_year_data, car_name_data])))[::-1]
```

[138]:
```
for i in range(5):
    print(desc_sorted_model_year_car_name_data[i][1])
```

```
chevy s-10
ford ranger
dodge rampage
vw pickup
ford mustang gl
```

1.9 9. Correlation

Compute the Pearson correlation coefficient between mpg and weight using NumPy.

[139]:
```
print("Pearson Correlation Coefficient = \n", np.corrcoef(mpg_data,␣
 ↪weight_data))
```

```
Pearson Correlation Coefficient =
 [[ 1.         -0.83174093]
 [-0.83174093  1.        ]]
```

1.10 10. Conditional Aggregates

Calculate the mean mpg for cars grouped by the number of cylinders using NumPy techniques.

[140]:
```
mpg_cylinder_data = np.column_stack(([mpg_data, cylinder_data]))
```

[141]:
```python
unique_cylinders = np.unique(np.delete(np.column_stack(([mpg_data,␣
↪cylinder_data])), 0, 1))
unique_cylinders
```

[141]:
```
array([3., 4., 5., 6., 8.])
```

[142]:
```python
for i in unique_cylinders:
    sum_cars = 0
    sum_mpg = 0
    mean = 0
    for j in range(len(mpg_cylinder_data)):
        if (mpg_cylinder_data[j][1]==i):
            sum_cars+=1
            sum_mpg+=mpg_cylinder_data[j][0]
    mean  = sum_mpg/sum_cars
    print("Mean mpg of",i,"cylinder cars = ", mean)
```

```
Mean mpg of 3.0 cylinder cars =  20.55
Mean mpg of 4.0 cylinder cars =  29.28676470588236
Mean mpg of 5.0 cylinder cars =  27.366666666666664
Mean mpg of 6.0 cylinder cars =  19.985714285714284
Mean mpg of 8.0 cylinder cars =  14.963106796116508
```

2 Pandas Problems

[143]:
```python
import pandas as pd
```

2.1 1. Basic Exploration

Load the dataset into a Pandas DataFrame. Display: - The first 10 rows - The total number of rows and columns - Summary statistics for numerical columns

[144]:
```python
df = pd.read_csv("auto-mpg.csv")
```

[145]:
```python
df.dtypes
```

[145]:
```
mpg             float64
cylinders         int64
displacement    float64
horsepower       object
weight            int64
acceleration    float64
model year        int64
origin            int64
car name         object
dtype: object
```

[146]:
```python
df['horsepower'] = df['horsepower'].apply(pd.to_numeric, errors='coerce')
df.dtypes
```

[146]:
```
mpg             float64
cylinders         int64
displacement    float64
horsepower      float64
weight            int64
acceleration    float64
model year        int64
origin            int64
car name         object
dtype: object
```

[147]:
```python
# First 10 rows
df.head(10)
```

[147]:
```
    mpg  cylinders  displacement  horsepower  weight  acceleration  \
0  18.0          8         307.0       130.0    3504          12.0
1  15.0          8         350.0       165.0    3693          11.5
2  18.0          8         318.0       150.0    3436          11.0
3  16.0          8         304.0       150.0    3433          12.0
4  17.0          8         302.0       140.0    3449          10.5
5  15.0          8         429.0       198.0    4341          10.0
6  14.0          8         454.0       220.0    4354           9.0
7  14.0          8         440.0       215.0    4312           8.5
8  14.0          8         455.0       225.0    4425          10.0
9  15.0          8         390.0       190.0    3850           8.5

   model year  origin                   car name
0          70       1  chevrolet chevelle malibu
1          70       1          buick skylark 320
2          70       1         plymouth satellite
3          70       1              amc rebel sst
4          70       1                ford torino
5          70       1           ford galaxie 500
6          70       1           chevrolet impala
7          70       1          plymouth fury iii
8          70       1           pontiac catalina
9          70       1         amc ambassador dpl
```

[148]:
```python
# Total number of rows and columns
print("No. of rows = ", df.shape[0])
print("No. of columns = ", df.shape[1])
```

```
No. of rows =  398
No. of columns =  9
```

[149]:
```
# Summary statistics for numerical columns
df.describe()
```

[149]:

	mpg	cylinders	displacement	horsepower	weight
count	398.000000	398.000000	398.000000	392.000000	398.000000
mean	23.514573	5.454774	193.425879	104.469388	2970.424623
std	7.815984	1.701004	104.269838	38.491160	846.841774
min	9.000000	3.000000	68.000000	46.000000	1613.000000
25%	17.500000	4.000000	104.250000	75.000000	2223.750000
50%	23.000000	4.000000	148.500000	93.500000	2803.500000
75%	29.000000	8.000000	262.000000	126.000000	3608.000000
max	46.600000	8.000000	455.000000	230.000000	5140.000000

	acceleration	model year	origin
count	398.000000	398.000000	398.000000
mean	15.568090	76.010050	1.572864
std	2.757689	3.697627	0.802055
min	8.000000	70.000000	1.000000
25%	13.825000	73.000000	1.000000
50%	15.500000	76.000000	1.000000
75%	17.175000	79.000000	2.000000
max	24.800000	82.000000	3.000000

2.2 2. Filtering and Indexing

Find all cars manufactured in 1975 with a weight less than 3000. Return the DataFrame with selected columns: car_name, weight, and mpg.

[150]:
```
filtered_df = df[(df["model year"]==75) & (df["weight"]<3000)]
filtered_df[["car name","weight","mpg"]]
```

[150]:

	car name	weight	mpg
167	toyota corolla	2171	29.0
168	ford pinto	2639	23.0
169	amc gremlin	2914	20.0
170	pontiac astro	2592	23.0
171	toyota corona	2702	24.0
172	volkswagen dasher	2223	25.0
173	datsun 710	2545	24.0
174	ford pinto	2984	18.0
175	volkswagen rabbit	1937	29.0
177	audi 100ls	2694	23.0
178	peugeot 504	2957	23.0
179	volvo 244dl	2945	22.0
180	saab 99le	2671	25.0
181	honda civic cvcc	1795	33.0

2.3 3. Handling Missing Data

Identify if there are any missing values in the dataset. Replace missing values in the horsepower column with the column's median.

[151]:
```
df.isna().any()
```

[151]:
```
mpg             False
cylinders       False
displacement    False
horsepower       True
weight          False
acceleration    False
model year      False
origin          False
car name        False
dtype: bool
```

[152]:
```
df['horsepower'] = df['horsepower'].fillna(df['horsepower'].mean())
```

[153]:
```
df.isna().any()
```

[153]:
```
mpg             False
cylinders       False
displacement    False
horsepower      False
weight          False
acceleration    False
model year      False
origin          False
car name        False
dtype: bool
```

2.4 4. Data Transformation

Add a new column power_to_weight_ratio, calculated as horsepower/weight.

[154]:
```
df['power_to_weight_ratio'] = df['horsepower']/df['weight']
```

[155]:
```
df
```

[155]:
```
      mpg  cylinders  displacement  horsepower  weight  acceleration  \
0    18.0          8         307.0       130.0    3504          12.0
1    15.0          8         350.0       165.0    3693          11.5
2    18.0          8         318.0       150.0    3436          11.0
3    16.0          8         304.0       150.0    3433          12.0
4    17.0          8         302.0       140.0    3449          10.5
..    …          …           …          …       …             …
393  27.0          4         140.0        86.0    2790          15.6
```

```
394  44.0          4         97.0        52.0    2130           24.6
395  32.0          4        135.0        84.0    2295           11.6
396  28.0          4        120.0        79.0    2625           18.6
397  31.0          4        119.0        82.0    2720           19.4

     model year  origin                   car name  power_to_weight_ratio
0            70       1  chevrolet chevelle malibu               0.037100
1            70       1          buick skylark 320               0.044679
2            70       1         plymouth satellite               0.043655
3            70       1              amc rebel sst               0.043694
4            70       1                ford torino               0.040591
..          …      …                         …                    …
393          82       1            ford mustang gl               0.030824
394          82       2                  vw pickup               0.024413
395          82       1              dodge rampage               0.036601
396          82       1                ford ranger               0.030095
397          82       1                 chevy s-10               0.030147

[398 rows x 10 columns]
```

2.5 5. Group By

Group the cars by origin and calculate the mean mpg for each group.

[156]:
```python
group = df.groupby(['origin'])
```

[157]:
```python
group['mpg'].mean()
```

[157]:
```
origin
1    20.083534
2    27.891429
3    30.450633
Name: mpg, dtype: float64
```

2.6 6. Sorting

Sort the DataFrame by mpg in descending order and display the top 10 cars withthe highest mpg.

[158]:
```python
df.sort_values(by='mpg', ascending=False).head(10)
```

[158]:
```
      mpg  cylinders  displacement  horsepower  weight  acceleration  \
322  46.6          4          86.0   65.000000    2110          17.9
329  44.6          4          91.0   67.000000    1850          13.8
325  44.3          4          90.0   48.000000    2085          21.7
394  44.0          4          97.0   52.000000    2130          24.6
326  43.4          4          90.0   48.000000    2335          23.7
244  43.1          4          90.0   48.000000    1985          21.5
309  41.5          4          98.0   76.000000    2144          14.7
```

```
330  40.9          4         85.0  104.469388    1835          17.3
324  40.8          4         85.0   65.000000    2110          19.2
247  39.4          4         85.0   70.000000    2070          18.6

     model year  origin                          car name  \
322          80       3                         mazda glc
329          80       3               honda civic 1500 gl
325          80       2              vw rabbit c (diesel)
394          82       2                         vw pickup
326          80       2                vw dasher (diesel)
244          78       2  volkswagen rabbit custom diesel
309          80       2                         vw rabbit
330          80       2              renault lecar deluxe
324          80       3                        datsun 210
247          78       3                    datsun b210 gx

     power_to_weight_ratio
322               0.030806
329               0.036216
325               0.023022
394               0.024413
326               0.020557
244               0.024181
309               0.035448
330               0.056932
324               0.030806
247               0.033816
```

2.7 7. Apply Function

Create a new column performance_score using a custom function:
`def performance_score(row):`
`return row['mpg'] * row['acceleration'] / row['weight']`
Apply this function to each row and store the result in the new column.

[159]:
```
def performance_score(row):
    return row['mpg']*row['acceleration']/row['weight']
```

[160]:
```
df['performance_score']=performance_score(df)
```

[161]:
```
df
```

[161]:
```
      mpg  cylinders  displacement  horsepower  weight  acceleration  \
0    18.0          8         307.0       130.0    3504          12.0
1    15.0          8         350.0       165.0    3693          11.5
2    18.0          8         318.0       150.0    3436          11.0
3    16.0          8         304.0       150.0    3433          12.0
4    17.0          8         302.0       140.0    3449          10.5
```

```
..   …        …         …        …      …          …
393  27.0          4       140.0        86.0    2790           15.6
394  44.0          4        97.0        52.0    2130           24.6
395  32.0          4       135.0        84.0    2295           11.6
396  28.0          4       120.0        79.0    2625           18.6
397  31.0          4       119.0        82.0    2720           19.4

     model year  origin                   car name  power_to_weight_ratio  \
0            70       1  chevrolet chevelle malibu               0.037100
1            70       1          buick skylark 320               0.044679
2            70       1         plymouth satellite               0.043655
3            70       1              amc rebel sst               0.043694
4            70       1                ford torino               0.040591
..          …      …                        …                    …
393          82       1            ford mustang gl               0.030824
394          82       2                  vw pickup               0.024413
395          82       1              dodge rampage               0.036601
396          82       1                ford ranger               0.030095
397          82       1                 chevy s-10               0.030147

     performance_score
0             0.061644
1             0.046710
2             0.057625
3             0.055928
4             0.051754
..                 …
393           0.150968
394           0.508169
395           0.161743
396           0.198400
397           0.221103

[398 rows x 11 columns]
```

2.8 8. Visualization Preparation

Generate a summary DataFrame with: - Average mpg, weight, and horsepower for each model_year.

[162]:
```python
summary_df = df.groupby('model year')[['mpg','weight','horsepower']].mean()
```

[163]:
```python
summary_df
```

[163]:
```
                  mpg       weight  horsepower
model year
70          17.689655  3372.793103  147.827586
71          21.250000  2995.428571  106.945335
72          18.714286  3237.714286  120.178571
```

```
73          17.100000  3419.025000  130.475000
74          22.703704  2877.925926   94.609977
75          20.266667  3176.800000  101.066667
76          21.573529  3078.735294  101.117647
77          23.375000  2997.357143  105.071429
78          24.061111  2861.805556   99.694444
79          25.093103  3055.344828  101.206897
80          33.696552  2436.655172   79.342716
81          30.334483  2522.931034   81.843772
82          31.709677  2453.548387   82.208690
```

2.9 9. Exporting Data

Save a subset of the data containing only mpg, cylinders, horsepower, and weight for cars with mpg > 30 into a CSV file named high_mpg_cars.csv.

[164]:
```python
subset_df = df[df['mpg']>30]
subset_df[['mpg','cylinders','horsepower','weight']]
```

[164]:
```
      mpg  cylinders  horsepower  weight
53   31.0          4        65.0    1773
54   35.0          4        69.0    1613
129  31.0          4        67.0    1950
131  32.0          4        65.0    1836
144  31.0          4        52.0    1649
..    …          …          …     …
390  32.0          4        96.0    2665
391  36.0          4        84.0    2370
394  44.0          4        52.0    2130
395  32.0          4        84.0    2295
397  31.0          4        82.0    2720

[85 rows x 4 columns]
```

[165]:
```python
subset_df[['mpg','cylinders','horsepower','weight']].to_csv('high_mpg_cars.csv')
```

[166]:
```python
verify_df = pd.read_csv('high_mpg_cars.csv')
verify_df.head()
```

[166]:
```
   Unnamed: 0   mpg  cylinders  horsepower  weight
0          53  31.0          4        65.0    1773
1          54  35.0          4        69.0    1613
2         129  31.0          4        67.0    1950
3         131  32.0          4        65.0    1836
4         144  31.0          4        52.0    1649
```

2.10 10. Finding Anomalies

Identify potential outliers in the mpg column using the Interquartile Range (IQR) method. Specifically: - Calculate the IQR for mpg. - Define outliers as values less than Q1 - 1.5 * IQR or greater than Q3 + 1.5 * IQR. - Create a DataFrame of cars classified as outliers, displaying car_name, mpg, and model_year.

[167]:
```python
# IQR for mpg
Q1 = df['mpg'].quantile(0.25)
Q3 = df['mpg'].quantile(0.75)
IQR = Q3-Q1
print("Q1 of mpg = ", Q1)
print("Q3 of mpg = ", Q3)
print("IQR of mpg = ", IQR)
```

```
Q1 of mpg =  17.5
Q3 of mpg =  29.0
IQR of mpg =  11.5
```

[168]:
```python
# Define outliers as values less than Q1 - 1.5 * IQR or greater than Q3 + 1.5 *␣
↪IQR.
outliers = (df['mpg']<(Q1 - 1.5 * IQR)) | (df['mpg']>(Q3 + 1.5 * IQR))
print("Outliers are mpg <",Q1 - 1.5 * IQR, "or mpg >", Q3 + 1.5 * IQR)
```

```
Outliers are mpg < 0.25 or mpg > 46.25
```

[169]:
```python
# Create a DataFrame of cars classified as outliers, displaying car_name, mpg,␣
↪and model_year.
outlier_df = df[outliers]
outlier_df[['car name','mpg','model year']]
```

[169]:

	car name	mpg	model year
322	mazda glc	46.6	80