hw1_question1

January 26, 2025

[1]:
```
import math
import numpy as np
import pandas as pd
```

[2]:
```
even = np.arange(2, 1001, 2) #returns numbers starting at 2 not including 1001␣
↪by 2, question 1 (1)
```

[3]:
```
print(even)
```

```
[   2    4    6    8   10   12   14   16   18   20   22   24   26   28
   30   32   34   36   38   40   42   44   46   48   50   52   54   56
   58   60   62   64   66   68   70   72   74   76   78   80   82   84
   86   88   90   92   94   96   98  100  102  104  106  108  110  112
  114  116  118  120  122  124  126  128  130  132  134  136  138  140
  142  144  146  148  150  152  154  156  158  160  162  164  166  168
  170  172  174  176  178  180  182  184  186  188  190  192  194  196
  198  200  202  204  206  208  210  212  214  216  218  220  222  224
  226  228  230  232  234  236  238  240  242  244  246  248  250  252
  254  256  258  260  262  264  266  268  270  272  274  276  278  280
  282  284  286  288  290  292  294  296  298  300  302  304  306  308
  310  312  314  316  318  320  322  324  326  328  330  332  334  336
  338  340  342  344  346  348  350  352  354  356  358  360  362  364
  366  368  370  372  374  376  378  380  382  384  386  388  390  392
  394  396  398  400  402  404  406  408  410  412  414  416  418  420
  422  424  426  428  430  432  434  436  438  440  442  444  446  448
  450  452  454  456  458  460  462  464  466  468  470  472  474  476
  478  480  482  484  486  488  490  492  494  496  498  500  502  504
  506  508  510  512  514  516  518  520  522  524  526  528  530  532
  534  536  538  540  542  544  546  548  550  552  554  556  558  560
  562  564  566  568  570  572  574  576  578  580  582  584  586  588
  590  592  594  596  598  600  602  604  606  608  610  612  614  616
  618  620  622  624  626  628  630  632  634  636  638  640  642  644
  646  648  650  652  654  656  658  660  662  664  666  668  670  672
  674  676  678  680  682  684  686  688  690  692  694  696  698  700
  702  704  706  708  710  712  714  716  718  720  722  724  726  728
  730  732  734  736  738  740  742  744  746  748  750  752  754  756
  758  760  762  764  766  768  770  772  774  776  778  780  782  784
  786  788  790  792  794  796  798  800  802  804  806  808  810  812
```

```
 814  816  818  820  822  824  826  828  830  832  834  836  838  840
 842  844  846  848  850  852  854  856  858  860  862  864  866  868
 870  872  874  876  878  880  882  884  886  888  890  892  894  896
 898  900  902  904  906  908  910  912  914  916  918  920  922  924
 926  928  930  932  934  936  938  940  942  944  946  948  950  952
 954  956  958  960  962  964  966  968  970  972  974  976  978  980
 982  984  986  988  990  992  994  996  998 1000]
```

[4]:
```
#Show length of array/vector
len(even)
```

[4]:
```
500
```

[5]:
```
#Reshape array/vector for data frame
even_reshape = even.reshape(100,5)
```

[6]:
```
#create reshaped even data frame
even_df = pd.DataFrame(even_reshape)
```

[7]:
```
even_df
```

[7]:

	0	1	2	3	4
0	2	4	6	8	10
1	12	14	16	18	20
2	22	24	26	28	30
3	32	34	36	38	40
4	42	44	46	48	50
..	…	…	…	…	…
95	952	954	956	958	960
96	962	964	966	968	970
97	972	974	976	978	980
98	982	984	986	988	990
99	992	994	996	998	1000

[100 rows x 5 columns]

[8]:
```
odd = np.arange(1,1000,2) #returns numbers starting at 1 not including 1000 by␣
 ↪2, question 1 part (2)
```

[9]:
```
print(odd)
```

```
[  1   3   5   7   9  11  13  15  17  19  21  23  25  27  29  31  33  35
  37  39  41  43  45  47  49  51  53  55  57  59  61  63  65  67  69  71
  73  75  77  79  81  83  85  87  89  91  93  95  97  99 101 103 105 107
 109 111 113 115 117 119 121 123 125 127 129 131 133 135 137 139 141 143
 145 147 149 151 153 155 157 159 161 163 165 167 169 171 173 175 177 179
 181 183 185 187 189 191 193 195 197 199 201 203 205 207 209 211 213 215
 217 219 221 223 225 227 229 231 233 235 237 239 241 243 245 247 249 251
```

```
253 255 257 259 261 263 265 267 269 271 273 275 277 279 281 283 285 287
289 291 293 295 297 299 301 303 305 307 309 311 313 315 317 319 321 323
325 327 329 331 333 335 337 339 341 343 345 347 349 351 353 355 357 359
361 363 365 367 369 371 373 375 377 379 381 383 385 387 389 391 393 395
397 399 401 403 405 407 409 411 413 415 417 419 421 423 425 427 429 431
433 435 437 439 441 443 445 447 449 451 453 455 457 459 461 463 465 467
469 471 473 475 477 479 481 483 485 487 489 491 493 495 497 499 501 503
505 507 509 511 513 515 517 519 521 523 525 527 529 531 533 535 537 539
541 543 545 547 549 551 553 555 557 559 561 563 565 567 569 571 573 575
577 579 581 583 585 587 589 591 593 595 597 599 601 603 605 607 609 611
613 615 617 619 621 623 625 627 629 631 633 635 637 639 641 643 645 647
649 651 653 655 657 659 661 663 665 667 669 671 673 675 677 679 681 683
685 687 689 691 693 695 697 699 701 703 705 707 709 711 713 715 717 719
721 723 725 727 729 731 733 735 737 739 741 743 745 747 749 751 753 755
757 759 761 763 765 767 769 771 773 775 777 779 781 783 785 787 789 791
793 795 797 799 801 803 805 807 809 811 813 815 817 819 821 823 825 827
829 831 833 835 837 839 841 843 845 847 849 851 853 855 857 859 861 863
865 867 869 871 873 875 877 879 881 883 885 887 889 891 893 895 897 899
901 903 905 907 909 911 913 915 917 919 921 923 925 927 929 931 933 935
937 939 941 943 945 947 949 951 953 955 957 959 961 963 965 967 969 971
973 975 977 979 981 983 985 987 989 991 993 995 997 999]
```

[10]:
```
#Show length of array/vector
len(odd)
```

[10]:
```
500
```

[11]:
```
#Reshape array/vector for data frame
odd_reshape = odd.reshape(100,5)
```

[12]:
```
#create reshaped odd data frame
odd_df = pd.DataFrame(odd_reshape)
```

[13]:
```
odd_df
```

[13]:

	0	1	2	3	4
0	1	3	5	7	9
1	11	13	15	17	19
2	21	23	25	27	29
3	31	33	35	37	39
4	41	43	45	47	49
..	…	…	…	…	…
95	951	953	955	957	959
96	961	963	965	967	969
97	971	973	975	977	979
98	981	983	985	987	989
99	991	993	995	997	999

```
[100 rows x 5 columns]
```

[14]:
```
#Question 1 part 3
```

[15]:
```
tf_list  = ["TRUE", "FALSE"]
```

[16]:
```
tf_vector = np.array((tf_list * (500 // len(tf_list)))[:500]) #needed ChatGPT␣
 ↪to determine how to limit list to 500 alternating
```

[17]:
```
print(tf_vector)
```

```
['TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE'
 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE'
 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE'
 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE'
 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE'
 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE'
 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE'
 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE'
 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE'
 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE'
 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE'
 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE'
 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE'
 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE'
 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE'
 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE'
 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE'
 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE'
 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE'
 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE'
 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE'
 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE'
 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE'
 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE'
 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE'
 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE'
 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE'
 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE'
 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE'
 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE'
 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE'
 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE'
 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE'
 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE'
 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE'
```

```
 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE'
 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE'
 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE'
 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE'
 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE'
 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE'
 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE'
 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE'
 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE'
 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE'
 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE'
 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE'
 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE'
 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE'
 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE'
 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE'
 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE'
 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE'
 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE'
 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE' 'TRUE'
 'FALSE' 'TRUE' 'FALSE' 'TRUE' 'FALSE']
```

[18]:
```python
#Create TRUE/FALSE data frame
tf_df = pd.DataFrame(tf_vector)
```

[19]:
```python
tf_df
```

[19]:
```
         0
0     TRUE
1    FALSE
2     TRUE
3    FALSE
4     TRUE
..     ...
495  FALSE
496   TRUE
497  FALSE
498   TRUE
499  FALSE

[500 rows x 1 columns]
```

[20]:
```python
rb_list = ["Red", "Blue"]
```

[21]:
```python
rb_vector = np.array((rb_list * (500 // len(rb_list))) [:500]) #needed ChatGPT␣
↪to determine how to limit list to 500 alternating
```

[22]:
```python
print(rb_vector)
```

```
['Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red'
 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue'
 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red'
 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue'
 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red'
 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue'
 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red'
 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue'
 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red'
 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue'
 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red'
 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue'
 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red'
 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue'
 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red'
 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue'
 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red'
 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue'
 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red'
 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue'
 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red'
 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue'
 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red'
 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue'
 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red'
 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue'
 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red'
 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue'
 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red'
 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue'
 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red'
 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue'
 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red'
 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue'
 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red'
 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue'
 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red'
 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue'
 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red'
 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue'
 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red'
 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue'
 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red'
 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue'
 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red' 'Blue' 'Red'
```

```
 'Blue' 'Red' 'Blue' 'Red' 'Blue']
```

[23]:
```
#Separate Red and Blue into two categories for data frame
Red = rb_vector[::2]
Blue = rb_vector[1::2]
```

[24]:
```
#Create Red/Blue data frame
rb_df = pd.DataFrame({"Red":Red, "Blue":Blue})
```

[25]:
```
rb_df
```

[25]:
```
     Red  Blue
0    Red  Blue
1    Red  Blue
2    Red  Blue
3    Red  Blue
4    Red  Blue
..   …    …
245  Red  Blue
246  Red  Blue
247  Red  Blue
248  Red  Blue
249  Red  Blue

[250 rows x 2 columns]
```

[26]:
```
#Combine Odd and Even data frames
combined_df = pd.DataFrame({"Odd":odd, "Even":even})
combined_df
```

[26]:
```
     Odd  Even
0      1     2
1      3     4
2      5     6
3      7     8
4      9    10
..   …    …
495  991   992
496  993   994
497  995   996
498  997   998
499  999  1000

[500 rows x 2 columns]
```

[27]:
```
# To combine all, needed to give column names to each of the vectors. Columns␣
 ↪indicated by "" followed by : to indicate variable name.
```

```
# Question 1 part (4)
```

[28]:
```
combined_all_df = pd.DataFrame({"Odd":odd, "Even":even, "True and False":
 ↪tf_vector, "Red and Blue":rb_vector})
combined_all_df
```

[28]:

	Odd	Even	True and False	Red and Blue
0	1	2	TRUE	Red
1	3	4	FALSE	Blue
2	5	6	TRUE	Red
3	7	8	FALSE	Blue
4	9	10	TRUE	Red
..	…	…	…	…
495	991	992	FALSE	Blue
496	993	994	TRUE	Red
497	995	996	FALSE	Blue
498	997	998	TRUE	Red
499	999	1000	FALSE	Blue

[500 rows x 4 columns]

[29]:
```
#Question 1 part (5)
```

[30]:
```
for i in range(99, 501, 100): #used for loops in a prior Python class
    print(even[i])
```

```
200
400
600
800
1000
```

[31]:
```
even_100 = even[99] #validate results of for loop by index
print(even_100)
```

```
200
```

[32]:
```
#Question 1 part (6)
```

[33]:
```
for i in range(101, 500, 100): #used for loops in a prior Python class
    print(odd[i])
print(odd[499])
```

```
203
403
603
803
999
```

[34]:
```
odd_401 = odd[401] #validate results of for loop by index
print(odd_401)
```

```
803
```

[35]:
```
#Question 1 part (7)
```

[36]:
```
for i in even:
    if i <23 or i >=451:
        print(i)
```

```
2
4
6
8
10
12
14
16
18
20
22
452
454
456
458
460
462
464
466
468
470
472
474
476
478
480
482
484
486
488
490
492
494
496
498
500
502
```

```
984
986
988
990
992
994
996
998
1000
```

[37]:
```
#Question 1 part (8)
```

[38]:
```
for i in odd:
    if i > 52 and i<= 300:
        print(i)
```

```
53
55
57
59
61
63
65
67
69
71
73
75
77
79
81
83
85
87
89
91
93
95
97
99
101
103
105
107
109
111
113
115
```

117
119
121
123
125
127
129
131
133
135
137
139
141
143
145
147
149
151
153
155
157
159
161
163
165
167
169
171
173
175
177
179
181
183
185
187
189
191
193
195
197
199
201
203
205
207
209
211

213
215
217
219
221
223
225
227
229
231
233
235
237
239
241
243
245
247
249
251
253
255
257
259
261
263
265
267
269
271
273
275
277
279
281
283
285
287
289
291
293
295
297
299

[39]:
```
#Question 1 part (9) #Referenced Stack Overflow to determine np.unique needed␣
↪to be used and W3 Schools to use correct syntax
```

[40]:
```
#Create vector of factors from TRUE/FALSE array/vector
tf_factor = np.unique(tf_vector)
print(tf_factor)
```

```
['FALSE' 'TRUE']
```

[41]:
```
#Question 1 part (10) #Referenced Stack Overflow to determine np.unique needed to be used and W3 Schools to use
#correct syntax
```

[42]:
```
#Create vector of factors from Red/Blue array/vector
rb_factor = np.unique(rb_vector)
print(rb_factor)
```

```
['Blue' 'Red']
```

[43]:
```
#Question 1 part (11)
```

[44]:
```
#Array/vector adding elements of even and odd arrays/vectors
totals = (even + odd)
print(totals)
```

```
[   3    7   11   15   19   23   27   31   35   39   43   47   51   55
   59   63   67   71   75   79   83   87   91   95   99  103  107  111
  115  119  123  127  131  135  139  143  147  151  155  159  163  167
  171  175  179  183  187  191  195  199  203  207  211  215  219  223
  227  231  235  239  243  247  251  255  259  263  267  271  275  279
  283  287  291  295  299  303  307  311  315  319  323  327  331  335
  339  343  347  351  355  359  363  367  371  375  379  383  387  391
  395  399  403  407  411  415  419  423  427  431  435  439  443  447
  451  455  459  463  467  471  475  479  483  487  491  495  499  503
  507  511  515  519  523  527  531  535  539  543  547  551  555  559
  563  567  571  575  579  583  587  591  595  599  603  607  611  615
  619  623  627  631  635  639  643  647  651  655  659  663  667  671
  675  679  683  687  691  695  699  703  707  711  715  719  723  727
  731  735  739  743  747  751  755  759  763  767  771  775  779  783
  787  791  795  799  803  807  811  815  819  823  827  831  835  839
  843  847  851  855  859  863  867  871  875  879  883  887  891  895
  899  903  907  911  915  919  923  927  931  935  939  943  947  951
  955  959  963  967  971  975  979  983  987  991  995  999 1003 1007
 1011 1015 1019 1023 1027 1031 1035 1039 1043 1047 1051 1055 1059 1063
 1067 1071 1075 1079 1083 1087 1091 1095 1099 1103 1107 1111 1115 1119
 1123 1127 1131 1135 1139 1143 1147 1151 1155 1159 1163 1167 1171 1175
 1179 1183 1187 1191 1195 1199 1203 1207 1211 1215 1219 1223 1227 1231
 1235 1239 1243 1247 1251 1255 1259 1263 1267 1271 1275 1279 1283 1287
 1291 1295 1299 1303 1307 1311 1315 1319 1323 1327 1331 1335 1339 1343
 1347 1351 1355 1359 1363 1367 1371 1375 1379 1383 1387 1391 1395 1399
 1403 1407 1411 1415 1419 1423 1427 1431 1435 1439 1443 1447 1451 1455
```

```
1459 1463 1467 1471 1475 1479 1483 1487 1491 1495 1499 1503 1507 1511
1515 1519 1523 1527 1531 1535 1539 1543 1547 1551 1555 1559 1563 1567
1571 1575 1579 1583 1587 1591 1595 1599 1603 1607 1611 1615 1619 1623
1627 1631 1635 1639 1643 1647 1651 1655 1659 1663 1667 1671 1675 1679
1683 1687 1691 1695 1699 1703 1707 1711 1715 1719 1723 1727 1731 1735
1739 1743 1747 1751 1755 1759 1763 1767 1771 1775 1779 1783 1787 1791
1795 1799 1803 1807 1811 1815 1819 1823 1827 1831 1835 1839 1843 1847
1851 1855 1859 1863 1867 1871 1875 1879 1883 1887 1891 1895 1899 1903
1907 1911 1915 1919 1923 1927 1931 1935 1939 1943 1947 1951 1955 1959
1963 1967 1971 1975 1979 1983 1987 1991 1995 1999]
```

[45]:
```
#Question 1 part (12), used ChatGPT to determine what function to use to␣
↪exclude the center values.
#Also used ChatGPT to validate the excluded numbers were correct
```

[46]:
```
smoosh = np.concatenate((totals[:199],totals[400:]))
print(smoosh)
```

```
[   3    7   11   15   19   23   27   31   35   39   43   47   51   55
   59   63   67   71   75   79   83   87   91   95   99  103  107  111
  115  119  123  127  131  135  139  143  147  151  155  159  163  167
  171  175  179  183  187  191  195  199  203  207  211  215  219  223
  227  231  235  239  243  247  251  255  259  263  267  271  275  279
  283  287  291  295  299  303  307  311  315  319  323  327  331  335
  339  343  347  351  355  359  363  367  371  375  379  383  387  391
  395  399  403  407  411  415  419  423  427  431  435  439  443  447
  451  455  459  463  467  471  475  479  483  487  491  495  499  503
  507  511  515  519  523  527  531  535  539  543  547  551  555  559
  563  567  571  575  579  583  587  591  595  599  603  607  611  615
  619  623  627  631  635  639  643  647  651  655  659  663  667  671
  675  679  683  687  691  695  699  703  707  711  715  719  723  727
  731  735  739  743  747  751  755  759  763  767  771  775  779  783
  787  791  795 1603 1607 1611 1615 1619 1623 1627 1631 1635 1639 1643
 1647 1651 1655 1659 1663 1667 1671 1675 1679 1683 1687 1691 1695 1699
 1703 1707 1711 1715 1719 1723 1727 1731 1735 1739 1743 1747 1751 1755
 1759 1763 1767 1771 1775 1779 1783 1787 1791 1795 1799 1803 1807 1811
 1815 1819 1823 1827 1831 1835 1839 1843 1847 1851 1855 1859 1863 1867
 1871 1875 1879 1883 1887 1891 1895 1899 1903 1907 1911 1915 1919 1923
 1927 1931 1935 1939 1943 1947 1951 1955 1959 1963 1967 1971 1975 1979
 1983 1987 1991 1995 1999]
```

[47]:
```
#Verified length of included elements
len(smoosh)
```

[47]: 299

[48]:
```
#Question 1 part (13) #Used ChatGPT for pd.series definition and prior␣
↪knowledge of Python functions
```

[49]:
```
def describe_array(arr):
    df = pd.Series(arr)
    df_summary = df.describe()
    print(f'Mean: {np.mean(arr)}')
    print(f'Variance: {np.var(arr)}')
    print(f'Standard Deviation: {np.std(arr)}')
    print(f'Statistics Summary: \n{df_summary}')
    print()

describe_array(even)
describe_array(odd)
```

```
Mean: 501.0
Variance: 83333.0
Standard Deviation: 288.6745572439663
Statistics Summary:
count     500.000000
mean      501.000000
std       288.963666
min         2.000000
25%       251.500000
50%       501.000000
75%       750.500000
max      1000.000000
dtype: float64

Mean: 500.0
Variance: 83333.0
Standard Deviation: 288.6745572439663
Statistics Summary:
count    500.000000
mean     500.000000
std      288.963666
min        1.000000
25%      250.500000
50%      500.000000
75%      749.500000
max      999.000000
dtype: float64
```

[ ]:

hw1_question2

January 26, 2025

[73]:
```
import pandas as pd
import numpy as np
import math
import statistics
import matplotlib.pyplot as plt
```

[3]:
```
#Import data
df = pd.read_csv('/Users/helenamabey/Downloads/forestfires2.csv' )
```

[51]:
```
df
```

[51]:

	X	Y	month	mon_num	day	day_num	FFMC	DMC	DC	ISI	temp	RH	\
0	7	5	mar	3	fri	6	86.2	26.2	94.3	5.1	8.2	51	\
1	7	4	oct	10	tue	3	90.6	35.4	669.1	6.7	18.0	33	
2	7	4	oct	10	sat	7	90.6	43.7	686.9	6.7	14.6	33	
3	8	6	mar	3	fri	6	91.7	33.3	77.5	9.0	8.3	97	
4	8	6	mar	3	sun	1	89.3	51.3	102.2	9.6	11.4	99	
..	..	..	…	…	…	…	…	…	…	…	…	..	
512	4	3	aug	8	sun	1	81.6	56.7	665.6	1.9	27.8	32	
513	2	4	aug	8	sun	1	81.6	56.7	665.6	1.9	21.9	71	
514	7	4	aug	8	sun	1	81.6	56.7	665.6	1.9	21.2	70	
515	1	4	aug	8	sat	7	94.4	146.0	614.7	11.3	25.6	42	
516	6	3	nov	11	tue	3	79.5	3.0	106.7	1.1	11.8	31	

	wind	rain	area
0	6.7	0.0	0.00
1	0.9	0.0	0.00
2	1.3	0.0	0.00
3	4.0	0.2	0.00
4	1.8	0.0	0.00
..	…	…	…
512	2.7	0.0	6.44
513	5.8	0.0	54.29
514	6.7	0.0	11.16
515	4.0	0.0	0.00
516	4.5	0.0	0.00

```
[517 rows x 15 columns]
```

[21]:
```
#Examine data types
print(df.dtypes)
```

```
X            int64
Y            int64
month       object
mon_num      int64
day         object
day_num      int64
FFMC       float64
DMC        float64
DC         float64
ISI        float64
temp       float64
RH           int64
wind       float64
rain       float64
area       float64
dtype: object
```

[31]:
```
# Temp stats
# length, sum, mean, variance, standard deviation, range, log
temp_length = len(df["temp"])
temp_sum = sum(df["temp"])
temp_variance = statistics.variance(df["temp"])
temp_standard_deviation = statistics.stdev(df["temp"])
temp_range = max(df["temp"]) - min(df["temp"])
temp_log = [math.log(x) for x in df["temp"]]

print(f'temp Length: {temp_length}')
print(f'temp Sum: {temp_sum}')
print(f'temp Variance: {temp_variance}')
print(f'temp Standard Deviation: {temp_standard_deviation}')
print(f'temp Range: {temp_range}')
print(f'temp Log: {temp_log}')
df["temp"].describe()
```

```
temp Length: 517
temp Sum: 9765.699999999999
temp Variance: 33.71689795030963
temp Standard Deviation: 5.806625349573505
temp Range: 31.099999999999998
temp Log: [2.1041341542702074, 2.8903717578961645, 2.681021528714291,
2.1162555148025524, 2.4336133554004498, 3.100092288878234, 3.1822118404966093,
2.0794415416798357, 2.5726122302071057, 3.126760535960395, 2.8791984572980396,
2.9601050959108397, 2.833213344056216, 3.0587070727153796, 3.2733640101522705,
```

3.131136910560194, 2.714694743820879, 2.8154087194227095, 2.766319109226186,
2.2300144001592104, 2.9069010598473755, 2.9496883350525844, 3.044522437723423,
2.970414465569701, 3.1654750481410856, 2.791165107812717, 2.9444389791664403,
2.9652730660692823, 3.407841924380824, 3.126760535960395, 3.2347491740244907,
2.4159137783010487, 3.0252910757955354, 2.8735646395797834, 3.054001181677967,
2.9014215940827497, 3.077312260546414, 2.424802725718295, 2.8791984572980396,
2.6461747973841225, 3.1484533605716547, 2.91235066461494, 2.8094026953624978,
2.975529566236472, 2.5572273113676265, 3.254242968705492, 2.6878474937846906,
3.1354942159291497, 2.468099531471619, 2.3978952727983707, 3.0349529867072724,
3.068052935133617, 3.0155349008501706, 3.0155349008501706, 2.8678989020441064,
3.3214324131932926, 2.8791984572980396, 2.624668592163159, 2.631888840136646,
2.509599262378372, 2.4423470353692043, 1.7047480922384253, 2.9338568698359038,
3.0349529867072724, 3.139832617527748, 2.9231615807191558, 3.1354942159291497,
2.975529566236472, 2.975529566236472, 2.8449093838194073, 2.760009940032921,
2.8735646395797834, 2.747270914255491, 2.8507065015037334, 3.3178157727231046,
1.9021075263969205, 2.7536607123542622, 2.1162555148025524, 2.6878474937846906,
3.0726933146901194, 2.970414465569701, 2.884800712846709, 2.9231615807191558,
2.8094026953624978, 3.005682604407159, 3.068052935133617, 3.2347491740244907,
3.109060958860994, 3.2308043957334744, 2.856470206220483, 2.6878474937846906,
2.856470206220483, 3.0349529867072724, 2.9014215940827497, 3.152736022363656,
2.8791984572980396, 2.5416019934645457, 2.856470206220483, 2.451005098112319,
2.9856819377004897, 2.9856819377004897, 2.667228206581955, 3.0007198150650303,
3.1822118404966093, 1.667706820558076, 2.5416019934645457, 2.9014215940827497,
3.0633909220278057, 3.0106208860477417, 2.856470206220483, 2.617395832834079,
2.9338568698359038, 3.126760535960395, 2.9391619220655967, 2.760009940032921,
2.740840023925201, 2.451005098112319, 2.7212954278522306, 2.3608540011180215,
2.975529566236472, 2.33214389523559, 2.8390784635086144, 3.1135153092103742,
2.884800712846709, 2.9856819377004897, 3.0252910757955354, 2.1972245773362196,
2.8449093838194073, 2.766319109226186, 2.7343675094195836, 2.7343675094195836,
2.6390573296152584, 2.3608540011180215, 2.8678989020441064, 2.7013612129514133,
2.8678989020441064, 2.8449093838194073, 2.747270914255491, 2.8903717578961645,
3.077312260546414, 3.086486636822455, 3.1484533605716547, 3.054001181677967,
2.8094026953624978, 3.169685580677429, 3.3105430133940246, 2.580216829592325,
3.186352633162641, 2.856470206220483, 3.1654750481410856, 3.1441522786722644,
3.2108436531709366, 3.202746442938317, 3.0007198150650303, 3.3877743613300146,
2.797281334830153, 3.353406717825807, 2.91235066461494, 3.0204248861443626,
2.9444389791664403, 2.7788192719904172, 3.0106208860477417, 2.7212954278522306,
2.8791984572980396, 2.8791984572980396, 1.667706820558076, 2.8094026953624978,
3.152736022363656, 2.681021528714291, 3.0301337002713233, 3.086486636822455,
2.856470206220483, 3.0007198150650303, 2.8735646395797834, 2.653241964607215,
3.0106208860477417, 1.7578579175523736, 2.954910279033736, 2.9069010598473755,
2.667228206581955, 3.173878458937465, 2.9496883350525844, 2.517696472610991,
2.8213788864092133, 3.0349529867072724, 2.8678989020441064, 2.4423470353692043,
3.044522437723423, 2.5877640352277083, 2.4423470353692043, 2.4595888418037104,
3.186352633162641, 3.202746442938317, 3.190476350346503, 3.202746442938317,
3.1570004211501135, 1.7578579175523736, 3.068052935133617, 2.631888840136646,
3.1179499062782403, 3.0726933146901194, 2.517696472610991, 2.174751721484161,
3.005682604407159, 2.714694743820879, 3.095577608523707, 3.131136910560194,

3.0301337002713233, 2.975529566236472, 3.1441522786722644, 2.91235066461494,
1.62924053973028, 3.0007198150650303, 2.3978952727983707, 2.833213344056216,
2.833213344056216, 2.8273136219290276, 2.517696472610991, 2.9652730660692823,
2.7212954278522306, 2.785011242238338, 2.9231615807191558, 2.3978952727983707,
2.5952547069568657, 2.7343675094195836, 3.131136910560194, 2.7788192719904172,
3.0007198150650303, 3.342861804649192, 2.797281334830153, 3.2733640101522705,
3.3250360206965914, 2.928523523860541, 3.190476350346503, 2.8735646395797834,
2.975529566236472, 2.9014215940827497, 2.9338568698359038, 3.2228678461377385,
2.5952547069568657, 2.7212954278522306, 2.8154087194227095, 2.7343675094195836,
3.086486636822455, 3.109060958860994, 3.288401887516811, 3.246490991901174,
3.0301337002713233, 3.3568971227655755, 3.077312260546414, 3.288401887516811,
3.1780538303479458, 3.095577608523707, 3.0633909220278057, 2.9391619220655967,
3.104586678466073, 3.173878458937465, 3.0633909220278057, 3.0252910757955354,
3.1654750481410856, 3.342861804649192, 2.4159137783010487, 3.0633909220278057,
2.9601050959108397, 3.0819099697950434, 3.095577608523707, 2.9652730660692823,
3.1654750481410856, 3.044522437723423, 2.9496883350525844, 3.0819099697950434,
3.0007198150650303, 3.005682604407159, 1.5686159179138452, 1.62924053973028,
1.62924053973028, 1.5260563034950492, 1.5260563034950492, 1.5260563034950492,
1.5260563034950492, 0.7884573603642703, 1.62924053973028, 1.4350845252893227,
2.174751721484161, 2.0149030205422647, 3.152736022363656, 2.533696813957432,
3.095577608523707, 3.186352633162641, 3.190476350346503, 2.928523523860541,
3.2308043957334744, 3.131136910560194, 3.292126286607793, 2.8390784635086144,
3.100092288878234, 2.6602595372658615, 2.7343675094195836, 2.975529566236472,
2.3608540011180215, 3.0301337002713233, 2.9496883350525844, 2.954910279033736,
2.954910279033736, 2.424802725718295, 2.9444389791664403, 2.8390784635086144,
3.169685580677429, 2.772588722239781, 3.214867803470662, 3.2308043957334744,
3.2108436531709366, 2.501435951739211, 3.190476350346503, 2.9806186357439426,
2.917770732084279, 2.9231615807191558, 2.954910279033736, 3.0726933146901194,
3.0726933146901194, 2.9391619220655967, 2.8213788864092133, 2.8213788864092133,
2.5572273113676265, 2.617395832834079, 3.186352633162641, 3.1822118404966093,
3.054001181677967, 2.9806186357439426, 3.1570004211501135, 3.186352633162641,
3.068052935133617, 2.8390784635086144, 2.89591193827178, 2.8903717578961645,
2.2823823856765264, 2.9601050959108397, 3.1354942159291497, 3.122364924487357,
3.0155349008501706, 2.9601050959108397, 2.7536607123542622, 3.0252910757955354,
2.766319109226186, 2.501435951739211, 2.8213788864092133, 3.0587070727153796,
2.312535423847214, 2.856470206220483, 2.5494451709255714, 2.312535423847214,
2.7343675094195836, 3.0252910757955354, 2.9856819377004897, 2.928523523860541,
3.0349529867072724, 3.0349529867072724, 2.766319109226186, 2.9806186357439426,
3.0492730404820207, 2.91235066461494, 2.8507065015037334, 2.7212954278522306,
2.766319109226186, 3.0492730404820207, 2.975529566236472, 2.766319109226186,
2.797281334830153, 2.8213788864092133, 2.624668592163159, 2.624668592163159,
2.653241964607215, 2.341805806147327, 3.0106208860477417, 2.33214389523559,
2.7343675094195836, 3.0492730404820207, 3.086486636822455, 2.163323025660538,
1.6486586255873816, 2.9601050959108397, 2.785011242238338, 3.339321977944068,
3.0204248861443626, 3.0587070727153796, 3.039749158970765, 3.0252910757955354,
2.451005098112319, 3.1484533605716547, 3.1484533605716547, 2.0149030205422647,
3.0301337002713233, 3.086486636822455, 2.7212954278522306, 1.667706820558076,
2.312535423847214, 3.0155349008501706, 3.190476350346503, 3.254242968705492,

```
3.332204510175204, 3.332204510175204, 3.126760535960395, 3.2188758248682006,
3.0587070727153796, 3.0819099697950434, 3.32862668882732, 2.833213344056216,
2.653241964607215, 2.990719731730447, 3.152736022363656, 2.6878474937846906,
2.1041341542702074, 3.126760535960395, 3.2733640101522705, 3.1822118404966093,
3.3141860046725258, 3.269568939183719, 2.624668592163159, 3.214867803470662,
3.2108436531709366, 3.265759410767051, 3.427514689979529, 3.3775875160230218,
3.104586678466073, 3.292126286607793, 3.0155349008501706, 3.0155349008501706,
3.32862668882732, 3.265759410767051, 3.202746442938317, 2.9652730660692823,
3.1484533605716547, 3.173878458937465, 3.039749158970765, 3.100092288878234,
3.169685580677429, 3.288401887516811, 2.653241964607215, 3.1612467120315646,
2.9496883350525844, 2.785011242238338, 3.2386784521643803, 2.388762789235098,
2.6946271807700692, 2.785011242238338, 2.8507065015037334, 2.9496883350525844,
2.186051276738094, 2.3513752571634776, 2.9601050959108397, 3.152736022363656,
2.468099531471619, 2.8735646395797834, 2.856470206220483, 2.8213788864092133,
2.884800712846709, 2.8094026953624978, 2.990719731730447, 2.9391619220655967,
2.740840023925201, 2.9391619220655967, 2.9391619220655967, 2.6741486494265287,
1.5260563034950492, 1.62924053973028, 1.5260563034950492, 2.322387720290225,
2.4159137783010487, 2.5877640352277083, 2.617395832834079, 2.8678989020441064,
2.8903717578961645, 2.6602595372658615, 3.1986731175506815, 3.2733640101522705,
3.122364924487357, 3.3032169733019514, 3.261935314328648, 2.9014215940827497,
3.1179499062782403, 3.407841924380824, 3.407841924380824, 3.152736022363656,
3.4339872044851463, 3.4995332823830174, 3.4210000089583352, 3.1822118404966093,
3.2733640101522705, 2.9652730660692823, 3.0252910757955354, 3.3568971227655755,
3.4781584227982836, 3.4781584227982836, 3.3141860046725258, 3.427514689979529,
3.173878458937465, 3.484312288372662, 3.475067230228611, 3.505557396986398,
3.3068867021909143, 3.0726933146901194, 3.0726933146901194, 3.0301337002713233,
3.374168709274236, 3.3638415951183864, 3.2846635654062037, 2.917770732084279,
3.254242968705492, 3.254242968705492, 3.0492730404820207, 2.9014215940827497,
3.3250360206965914, 3.3250360206965914, 3.086486636822455, 3.054001181677967,
3.242592351485517, 2.468099531471619]
```

[31]:
```
count    517.000000
mean      18.889168
std        5.806625
min        2.200000
25%       15.500000
50%       19.300000
75%       22.800000
max       33.300000
Name: temp, dtype: float64
```

[33]:
```
# Wind stats
# length, sum, mean, variance, standard deviation, range, log
wind_length = len(df["wind"])
wind_sum = sum(df["wind"])
wind_variance = statistics.variance(df["wind"])
wind_standard_deviation = statistics.stdev(df["wind"])
```

```
wind_range = max(df["wind"]) - min(df["wind"])
wind_log = [math.log(x) for x in df["wind"]]

print(f'wind Length: {wind_length}')
print(f'wind Sum: {wind_sum}')
print(f'wind Variance: {wind_variance}')
print(f'wind Standard Deviation: {wind_standard_deviation}')
print(f'wind Range: {wind_range}')
print(f'wind Log: {wind_log}')
df["wind"].describe()
```

```
wind Length: 517
wind Sum: 2077.1000000000004
wind Variance: 3.210019042478221
wind Standard Deviation: 1.7916526009464617
wind Range: 9.0
wind Log: [1.9021075263969205, -0.10536051565782628, 0.26236426446749106,
1.3862943611198906, 0.5877866649021191, 1.6863989535702288, 1.1314021114911006,
0.7884573603642703, 1.6863989535702288, 1.3862943611198906, 1.9740810260220096,
1.3862943611198906, 1.9021075263969205, 0.7884573603642703, 1.5040773967762742,
1.6863989535702288, 1.6863989535702288, 1.589235205116581, 1.3862943611198906,
1.5040773967762742, 0.9932517730102834, 0.9932517730102834, 1.5040773967762742,
1.7578579175523736, 1.7578579175523736, 1.6863989535702288, 1.7578579175523736,
0.26236426446749106, 0.9932517730102834, 1.2809338454620642, 1.2809338454620642,
2.028148247292285, 0.5877866649021191, 1.2809338454620642, 0.9932517730102834,
1.5040773967762742, 1.5040773967762742, 1.6863989535702288, 1.3862943611198906,
0.9932517730102834, 1.1314021114911006, 1.9021075263969205, 1.6863989535702288,
0.9932517730102834, 1.589235205116581, 1.3862943611198906, 1.2809338454620642,
1.1314021114911006, 0.5877866649021191, 1.7578579175523736, 0.26236426446749106,
0.7884573603642703, 1.589235205116581, 1.589235205116581, 1.2809338454620642,
0.7884573603642703, 1.589235205116581, 0.9932517730102834, 1.6863989535702288,
-0.10536051565782628, 1.7578579175523736, 1.840549633397487, 1.589235205116581,
0.9932517730102834, 1.1314021114911006, 1.5040773967762742, 1.5040773967762742,
1.6863989535702288, 1.840549633397487, 1.5040773967762742, 2.028148247292285,
1.2809338454620642, 1.840549633397487, 1.5040773967762742, 0.26236426446749106,
1.1314021114911006, 1.1314021114911006, 1.1314021114911006, 0.9932517730102834,
1.9021075263969205, 1.840549633397487, 0.7884573603642703, 1.5040773967762742,
-0.10536051565782628, 1.2809338454620642, -0.10536051565782628,
0.7884573603642703, 0.7884573603642703, 1.2809338454620642, 1.589235205116581,
1.7578579175523736, 1.6863989535702288, 1.840549633397487, 1.589235205116581,
0.9932517730102834, 0.26236426446749106, 0.5877866649021191, 1.6863989535702288,
1.5040773967762742, 1.6863989535702288, 1.6863989535702288, 1.6863989535702288,
1.3862943611198906, 1.5040773967762742, 1.1314021114911006, 1.840549633397487,
1.1314021114911006, 1.1314021114911006, 1.1314021114911006, 1.6863989535702288,
1.7578579175523736, 1.5040773967762742, 1.2809338454620642, 0.9932517730102834,
2.028148247292285, 1.840549633397487, 1.840549633397487, 1.589235205116581,
1.3862943611198906, 1.1314021114911006, 0.7884573603642703, 1.6863989535702288,
```

1.6863989535702288, 1.1314021114911006, 1.6863989535702288, 1.6863989535702288,
0.7884573603642703, 1.1314021114911006, 1.2809338454620642,
-0.10536051565782628, 0.9932517730102834, 1.1314021114911006, 1.589235205116581,
1.1314021114911006, 0.9932517730102834, 1.7578579175523736, 0.26236426446749106,
1.1314021114911006, 0.9932517730102834, 0.7884573603642703, 0.5877866649021191,
1.5040773967762742, 2.186051276738094, 1.6863989535702288, 1.6863989535702288,
1.3862943611198906, 1.6863989535702288, 1.2809338454620642, 1.9021075263969205,
1.5040773967762742, 1.6863989535702288, 0.7884573603642703, 0.5877866649021191,
1.589235205116581, 0.9932517730102834, 0.26236426446749106, 0.7884573603642703,
1.2809338454620642, 1.3862943611198906, 1.7578579175523736, 1.1314021114911006,
1.3862943611198906, 2.1400661634962708, 0.5877866649021191, 0.7884573603642703,
1.5040773967762742, -0.10536051565782628, 1.5040773967762742,
2.2407096892759584, 0.7884573603642703, 0.5877866649021191, 1.3862943611198906,
1.6863989535702288, 0.7884573603642703, 0.5877866649021191, 1.589235205116581,
1.7578579175523736, 0.9932517730102834, 0.7884573603642703, 1.6863989535702288,
1.9021075263969205, 1.3862943611198906, 0.7884573603642703, 1.5040773967762742,
1.589235205116581, 1.1314021114911006, 1.7578579175523736, 0.7884573603642703,
-0.10536051565782628, 1.3862943611198906, 1.3862943611198906,
0.9932517730102834, 1.5040773967762742, 1.3862943611198906, 1.5040773967762742,
1.6863989535702288, 1.7578579175523736, -0.10536051565782628, 1.840549633397487,
1.2809338454620642, 0.7884573603642703, 1.2809338454620642, 0.7884573603642703,
0.9932517730102834, 1.3862943611198906, 0.5877866649021191, 1.9740810260220096,
0.7884573603642703, 1.840549633397487, 1.589235205116581, 1.1314021114911006,
1.7578579175523736, 1.589235205116581, 1.7578579175523736, 1.589235205116581,
1.589235205116581, 0.26236426446749106, 1.840549633397487, 0.26236426446749106,
1.1314021114911006, 1.1314021114911006, 1.2809338454620642, 1.7578579175523736,
1.2809338454620642, 1.5040773967762742, 1.589235205116581, 1.3862943611198906,
1.5040773967762742, 1.1314021114911006, 1.3862943611198906, 1.5040773967762742,
1.1314021114911006, 0.9932517730102834, 1.1314021114911006, 1.1314021114911006,
1.7578579175523736, 0.5877866649021191, 0.7884573603642703, 1.3862943611198906,
0.5877866649021191, 0.9932517730102834, 1.1314021114911006, 1.3862943611198906,
2.028148247292285, 2.028148247292285, 1.840549633397487, 1.6863989535702288,
0.7884573603642703, 0.9932517730102834, -0.916290731874155, 1.1314021114911006,
1.1314021114911006, 1.2809338454620642, 0.9932517730102834, 1.1314021114911006,
1.3862943611198906, 0.7884573603642703, 0.9932517730102834,
-0.10536051565782628, 0.5877866649021191, 1.3862943611198906, 2.028148247292285,
1.1314021114911006, 1.2809338454620642, 1.1314021114911006, 2.028148247292285,
1.3862943611198906, 1.1314021114911006, 1.1314021114911006, 0.9932517730102834,
1.1314021114911006, 1.5040773967762742, 1.3862943611198906, 2.1400661634962708,
2.0794415416798357, 1.589235205116581, 2.1400661634962708, 2.1400661634962708,
2.1400661634962708, 2.1400661634962708, 1.589235205116581, 2.1400661634962708,
1.3862943611198906, 1.1314021114911006, 2.0794415416798357, 1.840549633397487,
2.028148247292285, 0.9932517730102834, 0.5877866649021191, 0.5877866649021191,
0.5877866649021191, -0.10536051565782628, 0.26236426446749106,
1.6863989535702288, 1.2809338454620642, 0.26236426446749106, 0.5877866649021191,
0.7884573603642703, 1.589235205116581, 0.9932517730102834, 1.589235205116581,
1.6863989535702288, 1.5040773967762742, 1.5040773967762742, 1.589235205116581,
0.7884573603642703, 1.6863989535702288, 1.2809338454620642, 0.5877866649021191,

0.7884573603642703, 0.9932517730102834, 0.5877866649021191, 1.840549633397487,
1.589235205116581, 0.5877866649021191, 0.9932517730102834, 1.7578579175523736,
1.589235205116581, 0.7884573603642703, 1.840549633397487, 1.9740810260220096,
1.3862943611198906, 1.3862943611198906, 0.9932517730102834, 0.5877866649021191,
1.1314021114911006, 1.1314021114911006, 0.7884573603642703, 0.5877866649021191,
1.3862943611198906, 1.1314021114911006, 1.5040773967762742, 0.7884573603642703,
1.1314021114911006, 1.6863989535702288, 0.5877866649021191, 0.7884573603642703,
0.7884573603642703, 0.7884573603642703, 0.5877866649021191, 0.7884573603642703,
0.7884573603642703, 0.5877866649021191, 1.5040773967762742, 1.589235205116581,
1.1314021114911006, 0.7884573603642703, 1.2809338454620642, 1.5040773967762742,
1.2809338454620642, 1.2809338454620642, 1.840549633397487, 1.2809338454620642,
0.9932517730102834, 0.7884573603642703, 1.589235205116581, 1.589235205116581,
1.2809338454620642, 0.9932517730102834, 0.7884573603642703, 0.7884573603642703,
1.3862943611198906, 1.1314021114911006, 0.7884573603642703, 0.9932517730102834,
1.1314021114911006, 1.6863989535702288, 1.2809338454620642, 1.589235205116581,
2.028148247292285, 2.028148247292285, 1.3862943611198906, -0.10536051565782628,
0.9932517730102834, 1.3862943611198906, 1.589235205116581, 0.7884573603642703,
0.7884573603642703, 1.7578579175523736, -0.10536051565782628,
1.9740810260220096, 0.9932517730102834, 0.5877866649021191, 0.9932517730102834,
1.5040773967762742, 0.7884573603642703, 1.6863989535702288, 1.6863989535702288,
1.1314021114911006, 1.1314021114911006, 1.840549633397487, 0.9932517730102834,
1.3862943611198906, 2.028148247292285, 0.5877866649021191, 0.5877866649021191,
1.589235205116581, 1.2809338454620642, 1.1314021114911006, 1.5040773967762742,
1.5040773967762742, 1.3862943611198906, 1.3862943611198906, 1.2809338454620642,
0.7884573603642703, 0.7884573603642703, 1.589235205116581, 1.3862943611198906,
1.1314021114911006, 1.6863989535702288, 0.9932517730102834, 2.2407096892759584,
1.5040773967762742, 1.2809338454620642, 1.3862943611198906, 1.589235205116581,
1.1314021114911006, 1.7578579175523736, 1.6863989535702288, 1.3862943611198906,
1.5040773967762742, 1.5040773967762742, 1.2809338454620642, 1.3862943611198906,
1.2809338454620642, 0.7884573603642703, 0.7884573603642703, 0.7884573603642703,
1.7578579175523736, 1.3862943611198906, 1.2809338454620642, 1.3862943611198906,
1.9021075263969205, 1.589235205116581, 1.2809338454620642, 0.5877866649021191,
0.26236426446749106, 0.9932517730102834, 1.3862943611198906, 0.7884573603642703,
1.2809338454620642, 0.5877866649021191, 1.1314021114911006, 2.0794415416798357,
1.2809338454620642, 1.5040773967762742, 0.7884573603642703, 2.0794415416798357,
1.3862943611198906, 1.589235205116581, 1.6863989535702288, 1.589235205116581,
1.3862943611198906, 1.1314021114911006, 1.1314021114911006, 0.9932517730102834,
0.9932517730102834, 1.3862943611198906, 1.589235205116581, 2.0794415416798357,
1.589235205116581, 1.589235205116581, 2.028148247292285, 1.840549633397487,
1.6863989535702288, -0.10536051565782628, 1.7578579175523736,
1.6863989535702288, 1.2809338454620642, 2.2407096892759584, 1.7578579175523736,
1.3862943611198906, 1.3862943611198906, 1.1314021114911006, 0.9932517730102834,
2.2407096892759584, 0.26236426446749106, 1.3862943611198906, 1.5040773967762742,
1.589235205116581, 1.5040773967762742, 1.589235205116581, 1.7578579175523736,
1.6863989535702288, 1.3862943611198906, 1.2809338454620642, 1.840549633397487,
1.2809338454620642, 2.028148247292285, 0.26236426446749106, 1.3862943611198906,
1.5040773967762742, 0.7884573603642703, 1.5040773967762742, 1.589235205116581,
0.7884573603642703, 1.1314021114911006, 0.7884573603642703, 0.9932517730102834,

```
1.589235205116581, 1.589235205116581, 1.589235205116581, 1.589235205116581,
1.589235205116581, 1.589235205116581, 0.5877866649021191, 2.1400661634962708,
1.2809338454620642, 1.2809338454620642, 2.028148247292285, 1.6863989535702288,
0.9932517730102834, 0.9932517730102834, 1.7578579175523736, 1.9021075263969205,
1.3862943611198906, 1.5040773967762742]
```

[33]:
```
count    517.000000
mean       4.017602
std        1.791653
min        0.400000
25%        2.700000
50%        4.000000
75%        4.900000
max        9.400000
Name: wind, dtype: float64
```

[59]:
```python
# Rain stats
# length, sum, mean, variance, standard deviation, range, log
# Used ChatGPT to help with the logs because there were many 0 values to␣
 ↪exclude in order to get results
rain_length = len(df["rain"])
rain_sum = sum(df["rain"])
rain_variance = statistics.variance(df["rain"])
rain_standard_deviation = statistics.stdev(df["rain"])
rain_range = max(df["rain"]) - min(df["rain"])
valid_rain = df[df["rain"] > 0]["rain"]
rain_log = np.log(valid_rain)

print(f'rain Length: {rain_length}')
print(f'rain Sum: {rain_sum}')
print(f'rain Variance: {rain_variance}')
print(f'rain Standard Deviation: {rain_standard_deviation}')
print(f'rain Range: {rain_range}')
print(f'rain Log: \n{rain_log}')
df["rain"].describe()
```

```
rain Length: 517
rain Sum: 11.200000000000003
rain Variance: 0.08759180123851079
rain Standard Deviation: 0.295959120890894
rain Range: 6.4
rain Log:
3     -1.609438
243    0.000000
286   -1.609438
499    1.856298
500   -0.223144
501   -0.223144
```

```
502   -0.916291
509    0.336472
Name: rain, dtype: float64
```

[59]:
```
count    517.000000
mean       0.021663
std        0.295959
min        0.000000
25%        0.000000
50%        0.000000
75%        0.000000
max        6.400000
Name: rain, dtype: float64
```

[37]:
```python
# Area stats
# length, sum, mean, variance, standard deviation, range, log
area_length = len(df["area"])
area_sum = sum(df["area"])
area_variance = statistics.variance(df["area"])
area_standard_deviation = statistics.stdev(df["area"])
area_range = max(df["area"]) - min(df["area"])
area_log = [math.log(x) for x in df["temp"]]

print(f'area Length: {area_length}')
print(f'area Sum: {area_sum}')
print(f'area Variance: {area_variance}')
print(f'area Standard Deviation: {area_standard_deviation}')
print(f'area Range: {area_range}')
print(f'area Log: {area_log}')
df["area"].describe()
```

```
area Length: 517
area Sum: 6642.049999999998
area Variance: 4052.063224823444
area Standard Deviation: 63.65581846794089
area Range: 1090.84
area Log: [2.1041341542702074, 2.8903717578961645, 2.681021528714291,
2.1162555148025524, 2.4336133554004498, 3.100092288878234, 3.1822118404966093,
2.0794415416798357, 2.5726122302071057, 3.126760535960395, 2.8791984572980396,
2.9601050959108397, 2.833213344056216, 3.0587070727153796, 3.2733640101522705,
3.131136910560194, 2.714694743820879, 2.8154087194227095, 2.766319109226186,
2.2300144001592104, 2.9069010598473755, 2.9496883350525844, 3.044522437723423,
2.970414465569701, 3.1654750481410856, 2.791165107812717, 2.9444389791664403,
2.9652730660692823, 3.407841924380824, 3.126760535960395, 3.2347491740244907,
2.4159137783010487, 3.0252910757955354, 2.8735646395797834, 3.054001181677967,
2.9014215940827497, 3.077312260546414, 2.424802725718295, 2.8791984572980396,
2.6461747973841225, 3.1484533605716547, 2.91235066461494, 2.8094026953624978,
2.975529566236472, 2.5572273113676265, 3.254242968705492, 2.6878474937846906,
```

3.1354942159291497, 2.468099531471619, 2.3978952727983707, 3.0349529867072724,
3.068052935133617, 3.0155349008501706, 3.0155349008501706, 2.8678989020441064,
3.3214324131932926, 2.8791984572980396, 2.624668592163159, 2.631888840136646,
2.509599262378372, 2.4423470353692043, 1.7047480922384253, 2.9338568698359038,
3.0349529867072724, 3.139832617527748, 2.9231615807191558, 3.1354942159291497,
2.975529566236472, 2.975529566236472, 2.8449093838194073, 2.760009940032921,
2.8735646395797834, 2.747270914255491, 2.8507065015037334, 3.3178157727231046,
1.9021075263969205, 2.7536607123542622, 2.1162555148025524, 2.6878474937846906,
3.0726933146901194, 2.970414465569701, 2.884800712846709, 2.9231615807191558,
2.8094026953624978, 3.005682604407159, 3.068052935133617, 3.2347491740244907,
3.109060958860994, 3.2308043957334744, 2.856470206220483, 2.6878474937846906,
2.856470206220483, 3.0349529867072724, 2.9014215940827497, 3.152736022363656,
2.8791984572980396, 2.5416019934645457, 2.856470206220483, 2.451005098112319,
2.9856819377004897, 2.9856819377004897, 2.667228206581955, 3.0007198150650303,
3.1822118404966093, 1.667706820558076, 2.5416019934645457, 2.9014215940827497,
3.0633909220278057, 3.0106208860477417, 2.856470206220483, 2.617395832834079,
2.9338568698359038, 3.126760535960395, 2.9391619220655967, 2.760009940032921,
2.740840023925201, 2.451005098112319, 2.7212954278522306, 2.3608540011180215,
2.975529566236472, 2.33214389523559, 2.8390784635086144, 3.1135153092103742,
2.884800712846709, 2.9856819377004897, 3.0252910757955354, 2.1972245773362196,
2.8449093838194073, 2.766319109226186, 2.7343675094195836, 2.7343675094195836,
2.6390573296152584, 2.3608540011180215, 2.8678989020441064, 2.7013612129514133,
2.8678989020441064, 2.8449093838194073, 2.747270914255491, 2.8903717578961645,
3.077312260546414, 3.086486636822455, 3.1484533605716547, 3.054001181677967,
2.8094026953624978, 3.169685580677429, 3.3105430133940246, 2.580216829592325,
3.186352633162641, 2.856470206220483, 3.1654750481410856, 3.1441522786722644,
3.2108436531709366, 3.202746442938317, 3.0007198150650303, 3.3877743613300146,
2.797281334830153, 3.353406717825807, 2.91235066461494, 3.0204248861443626,
2.9444389791664403, 2.7788192719904172, 3.0106208860477417, 2.7212954278522306,
2.8791984572980396, 2.8791984572980396, 1.667706820558076, 2.8094026953624978,
3.152736022363656, 2.681021528714291, 3.0301337002713233, 3.086486636822455,
2.856470206220483, 3.0007198150650303, 2.8735646395797834, 2.653241964607215,
3.0106208860477417, 1.7578579175523736, 2.954910279033736, 2.9069010598473755,
2.667228206581955, 3.173878458937465, 2.9496883350525844, 2.517696472610991,
2.8213788864092133, 3.0349529867072724, 2.8678989020441064, 2.4423470353692043,
3.044522437723423, 2.5877640352277083, 2.4423470353692043, 2.4595888418037104,
3.186352633162641, 3.202746442938317, 3.190476350346503, 3.202746442938317,
3.1570004211501135, 1.7578579175523736, 3.068052935133617, 2.631888840136646,
3.1179499062782403, 3.0726933146901194, 2.517696472610991, 2.174751721484161,
3.005682604407159, 2.714694743820879, 3.095577608523707, 3.131136910560194,
3.0301337002713233, 2.975529566236472, 3.1441522786722644, 2.91235066461494,
1.62924053973028, 3.0007198150650303, 2.3978952727983707, 2.833213344056216,
2.833213344056216, 2.8273136219290276, 2.517696472610991, 2.9652730660692823,
2.7212954278522306, 2.785011242238338, 2.9231615807191558, 2.3978952727983707,
2.5952547069568657, 2.7343675094195836, 3.131136910560194, 2.7788192719904172,
3.0007198150650303, 3.342861804649192, 2.797281334830153, 3.2733640101522705,
3.3250360206965914, 2.928523523860541, 3.190476350346503, 2.8735646395797834,
2.975529566236472, 2.9014215940827497, 2.9338568698359038, 3.2228678461377385,

2.5952547069568657, 2.7212954278522306, 2.8154087194227095, 2.7343675094195836,
3.086486636822455, 3.109060958860994, 3.288401887516811, 3.246490991901174,
3.0301337002713233, 3.3568971227655755, 3.077312260546414, 3.288401887516811,
3.1780538303479458, 3.095577608523707, 3.0633909220278057, 2.9391619220655967,
3.104586678466073, 3.173878458937465, 3.0633909220278057, 3.0252910757955354,
3.1654750481410856, 3.342861804649192, 2.4159137783010487, 3.0633909220278057,
2.9601050959108397, 3.0819099697950434, 3.095577608523707, 2.9652730660692823,
3.1654750481410856, 3.044522437723423, 2.9496883350525844, 3.0819099697950434,
3.0007198150650303, 3.005682604407159, 1.5686159179138452, 1.62924053973028,
1.62924053973028, 1.5260563034950492, 1.5260563034950492, 1.5260563034950492,
1.5260563034950492, 0.7884573603642703, 1.62924053973028, 1.4350845252893227,
2.174751721484161, 2.0149030205422647, 3.152736022363656, 2.533696813957432,
3.095577608523707, 3.186352633162641, 3.190476350346503, 2.928523523860541,
3.2308043957334744, 3.131136910560194, 3.292126286607793, 2.8390784635086144,
3.100092288878234, 2.6602595372658615, 2.7343675094195836, 2.975529566236472,
2.3608540011180215, 3.0301337002713233, 2.9496883350525844, 2.954910279033736,
2.954910279033736, 2.424802725718295, 2.9444389791664403, 2.8390784635086144,
3.169685580677429, 2.772588722239781, 3.214867803470662, 3.2308043957334744,
3.2108436531709366, 2.501435951739211, 3.190476350346503, 2.9806186357439426,
2.917770732084279, 2.9231615807191558, 2.954910279033736, 3.0726933146901194,
3.0726933146901194, 2.9391619220655967, 2.8213788864092133, 2.8213788864092133,
2.5572273113676265, 2.617395832834079, 3.186352633162641, 3.1822118404966093,
3.054001181677967, 2.9806186357439426, 3.1570004211501135, 3.186352633162641,
3.068052935133617, 2.8390784635086144, 2.89591193827178, 2.8903717578961645,
2.2823823856765264, 2.9601050959108397, 3.1354942159291497, 3.122364924487357,
3.0155349008501706, 2.9601050959108397, 2.7536607123542622, 3.0252910757955354,
2.766319109226186, 2.501435951739211, 2.8213788864092133, 3.0587070727153796,
2.312535423847214, 2.856470206220483, 2.5494451709255714, 2.312535423847214,
2.7343675094195836, 3.0252910757955354, 2.9856819377004897, 2.928523523860541,
3.0349529867072724, 3.0349529867072724, 2.766319109226186, 2.9806186357439426,
3.0492730404820207, 2.91235066461494, 2.8507065015037334, 2.7212954278522306,
2.766319109226186, 3.0492730404820207, 2.975529566236472, 2.766319109226186,
2.797281334830153, 2.8213788864092133, 2.624668592163159, 2.624668592163159,
2.653241964607215, 2.341805806147327, 3.0106208860477417, 2.33214389523559,
2.7343675094195836, 3.0492730404820207, 3.086486636822455, 2.163323025660538,
1.6486586255873816, 2.9601050959108397, 2.785011242238338, 3.339321977944068,
3.0204248861443626, 3.0587070727153796, 3.039749158970765, 3.0252910757955354,
2.451005098112319, 3.1484533605716547, 3.1484533605716547, 2.0149030205422647,
3.0301337002713233, 3.086486636822455, 2.7212954278522306, 1.667706820558076,
2.312535423847214, 3.0155349008501706, 3.190476350346503, 3.254242968705492,
3.332204510175204, 3.332204510175204, 3.126760535960395, 3.2188758248682006,
3.0587070727153796, 3.0819099697950434, 3.32862668882732, 2.833213344056216,
2.653241964607215, 2.990719731730447, 3.152736022363656, 2.6878474937846906,
2.1041341542702074, 3.126760535960395, 3.2733640101522705, 3.1822118404966093,
3.3141860046725258, 3.269568939183719, 2.624668592163159, 3.214867803470662,
3.2108436531709366, 3.265759410767051, 3.427514689979529, 3.3775875160230218,
3.104586678466073, 3.292126286607793, 3.0155349008501706, 3.0155349008501706,
3.32862668882732, 3.265759410767051, 3.202746442938317, 2.9652730660692823,

```
3.1484533605716547, 3.173878458937465, 3.039749158970765, 3.100092288878234,
3.169685580677429, 3.288401887516811, 2.653241964607215, 3.1612467120315646,
2.9496883350525844, 2.785011242238338, 3.2386784521643803, 2.388762789235098,
2.6946271807700692, 2.785011242238338, 2.8507065015037334, 2.9496883350525844,
2.186051276738094, 2.3513752571634776, 2.9601050959108397, 3.152736022363656,
2.468099531471619, 2.8735646395797834, 2.856470206220483, 2.8213788864092133,
2.884800712846709, 2.8094026953624978, 2.990719731730447, 2.9391619220655967,
2.740840023925201, 2.9391619220655967, 2.9391619220655967, 2.6741486494265287,
1.5260563034950492, 1.62924053973028, 1.5260563034950492, 2.322387720290225,
2.4159137783010487, 2.5877640352277083, 2.617395832834079, 2.8678989020441064,
2.8903717578961645, 2.6602595372658615, 3.1986731175506815, 3.2733640101522705,
3.122364924487357, 3.3032169733019514, 3.261935314328648, 2.9014215940827497,
3.1179499062782403, 3.407841924380824, 3.407841924380824, 3.152736022363656,
3.4339872044851463, 3.4995332823830174, 3.4210000089583352, 3.1822118404966093,
3.2733640101522705, 2.9652730660692823, 3.0252910757955354, 3.3568971227655755,
3.4781584227982836, 3.4781584227982836, 3.3141860046725258, 3.427514689979529,
3.173878458937465, 3.484312288372662, 3.475067230228611, 3.505557396986398,
3.3068867021909143, 3.0726933146901194, 3.0726933146901194, 3.0301337002713233,
3.374168709274236, 3.3638415951183864, 3.2846635654062037, 2.917770732084279,
3.254242968705492, 3.254242968705492, 3.0492730404820207, 2.9014215940827497,
3.3250360206965914, 3.3250360206965914, 3.086486636822455, 3.054001181677967,
3.242592351485517, 2.468099531471619]
```

[37]:
```
count     517.000000
mean       12.847292
std        63.655818
min         0.000000
25%         0.000000
50%         0.520000
75%         6.570000
max      1090.840000
Name: area, dtype: float64
```

[61]:
```
#Reviewed data frame
df
```

[61]:

	X	Y	month	mon_num	day	day_num	FFMC	DMC	DC	ISI	temp	RH	\
0	7	5	mar	3	fri	6	86.2	26.2	94.3	5.1	8.2	51	\
1	7	4	oct	10	tue	3	90.6	35.4	669.1	6.7	18.0	33	
2	7	4	oct	10	sat	7	90.6	43.7	686.9	6.7	14.6	33	
3	8	6	mar	3	fri	6	91.7	33.3	77.5	9.0	8.3	97	
4	8	6	mar	3	sun	1	89.3	51.3	102.2	9.6	11.4	99	
..	..	..	…	…	…	…	…	…	…	…	…	..	
512	4	3	aug	8	sun	1	81.6	56.7	665.6	1.9	27.8	32	
513	2	4	aug	8	sun	1	81.6	56.7	665.6	1.9	21.9	71	
514	7	4	aug	8	sun	1	81.6	56.7	665.6	1.9	21.2	70	
515	1	4	aug	8	sat	7	94.4	146.0	614.7	11.3	25.6	42	

```
516  6  3   nov       11  tue        3  79.5    3.0  106.7   1.1  11.8  31

     wind  rain   area
0     6.7   0.0   0.00
1     0.9   0.0   0.00
2     1.3   0.0   0.00
3     4.0   0.2   0.00
4     1.8   0.0   0.00
..    …   …    …
512   2.7   0.0   6.44
513   5.8   0.0  54.29
514   6.7   0.0  11.16
515   4.0   0.0   0.00
516   4.5   0.0   0.00

[517 rows x 15 columns]
```

[65]:
```
#Examined data types
print(df.dtypes)
```

```
X            int64
Y            int64
month       object
mon_num      int64
day         object
day_num      int64
FFMC       float64
DMC        float64
DC         float64
ISI        float64
temp       float64
RH           int64
wind       float64
rain       float64
area       float64
dtype: object
```

[67]:
```
# Neither month or day was catergorical. Referenced ChatGPT for method to␣
 ↪change from object to category, verified by .dtypes
df["month"] = pd.Categorical(df["month"], categories=["jan", "feb", "mar",␣
 ↪"apr", "may", "jun", "jul", "aug", "sep", "oct", "nov", "dec"], ordered=True)

df["day"] = pd.Categorical(df["day"], categories=["mon", "tue", "wed", "thu",␣
 ↪"fri", "sat", "sun"], ordered=True)
```

[69]:
```
df
```

[69]:

	X	Y	month	mon_num	day	day_num	FFMC	DMC	DC	ISI	temp	RH	\
0	7	5	mar	3	fri	6	86.2	26.2	94.3	5.1	8.2	51	
1	7	4	oct	10	tue	3	90.6	35.4	669.1	6.7	18.0	33	
2	7	4	oct	10	sat	7	90.6	43.7	686.9	6.7	14.6	33	
3	8	6	mar	3	fri	6	91.7	33.3	77.5	9.0	8.3	97	
4	8	6	mar	3	sun	1	89.3	51.3	102.2	9.6	11.4	99	
..	..	..	…	…	…	…	…	…	…	…	…	..	
512	4	3	aug	8	sun	1	81.6	56.7	665.6	1.9	27.8	32	
513	2	4	aug	8	sun	1	81.6	56.7	665.6	1.9	21.9	71	
514	7	4	aug	8	sun	1	81.6	56.7	665.6	1.9	21.2	70	
515	1	4	aug	8	sat	7	94.4	146.0	614.7	11.3	25.6	42	
516	6	3	nov	11	tue	3	79.5	3.0	106.7	1.1	11.8	31	

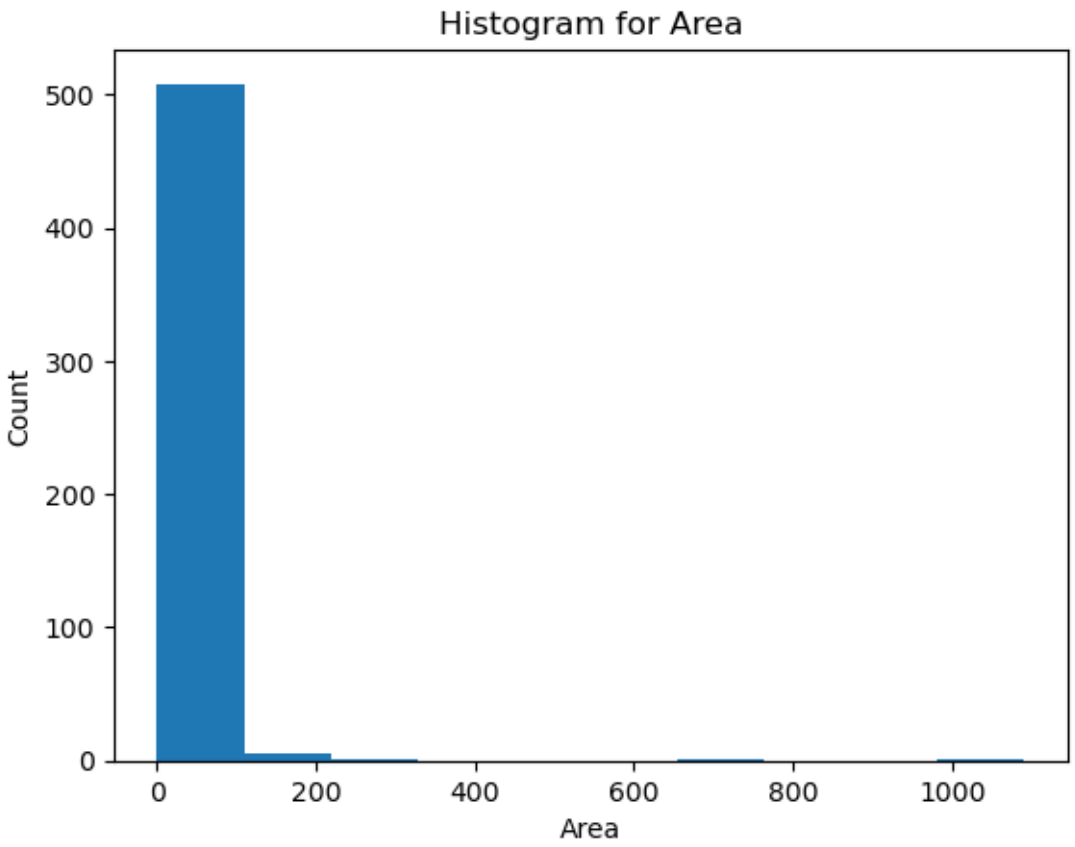
	wind	rain	area
0	6.7	0.0	0.00
1	0.9	0.0	0.00
2	1.3	0.0	0.00
3	4.0	0.2	0.00
4	1.8	0.0	0.00
..	…	…	…
512	2.7	0.0	6.44
513	5.8	0.0	54.29
514	6.7	0.0	11.16
515	4.0	0.0	0.00
516	4.5	0.0	0.00

[517 rows x 15 columns]

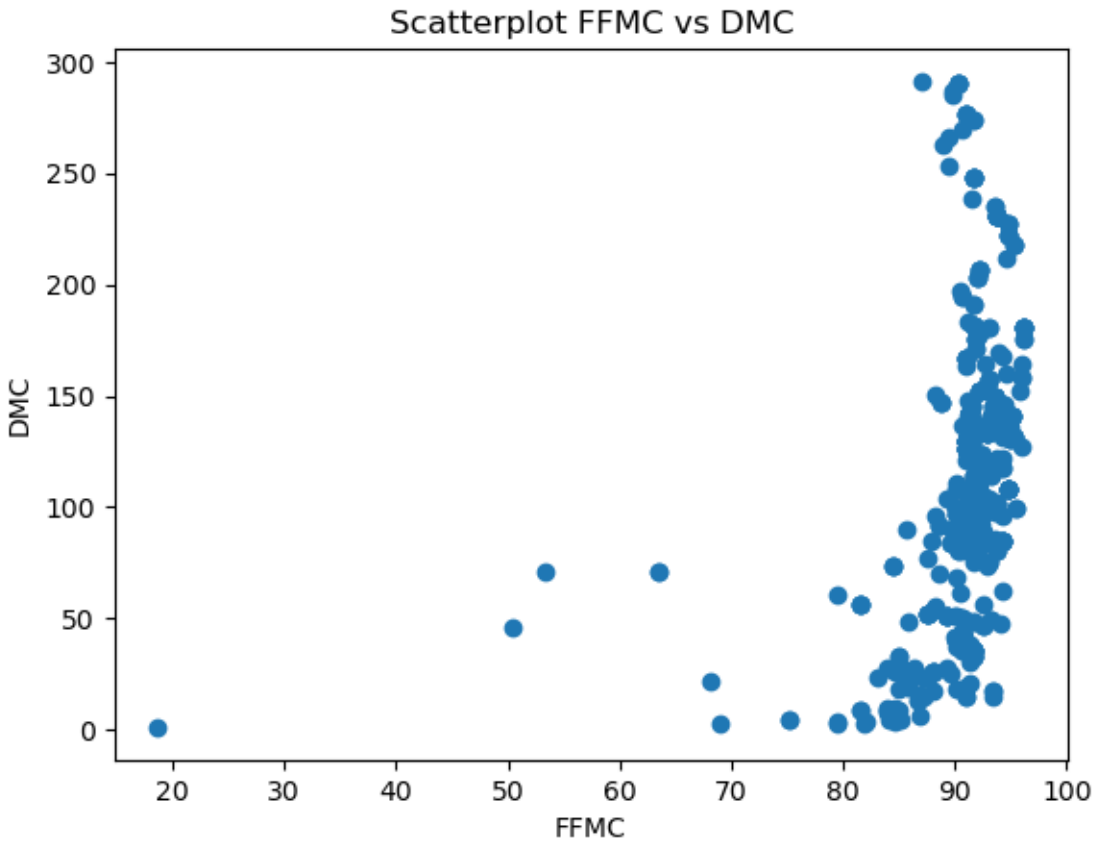
[71]:
```
#Reviewed updated data types
print(df.dtypes)
```

```
X             int64
Y             int64
month      category
mon_num       int64
day        category
day_num       int64
FFMC        float64
DMC         float64
DC          float64
ISI         float64
temp        float64
RH            int64
wind        float64
rain        float64
area        float64
dtype: object
```

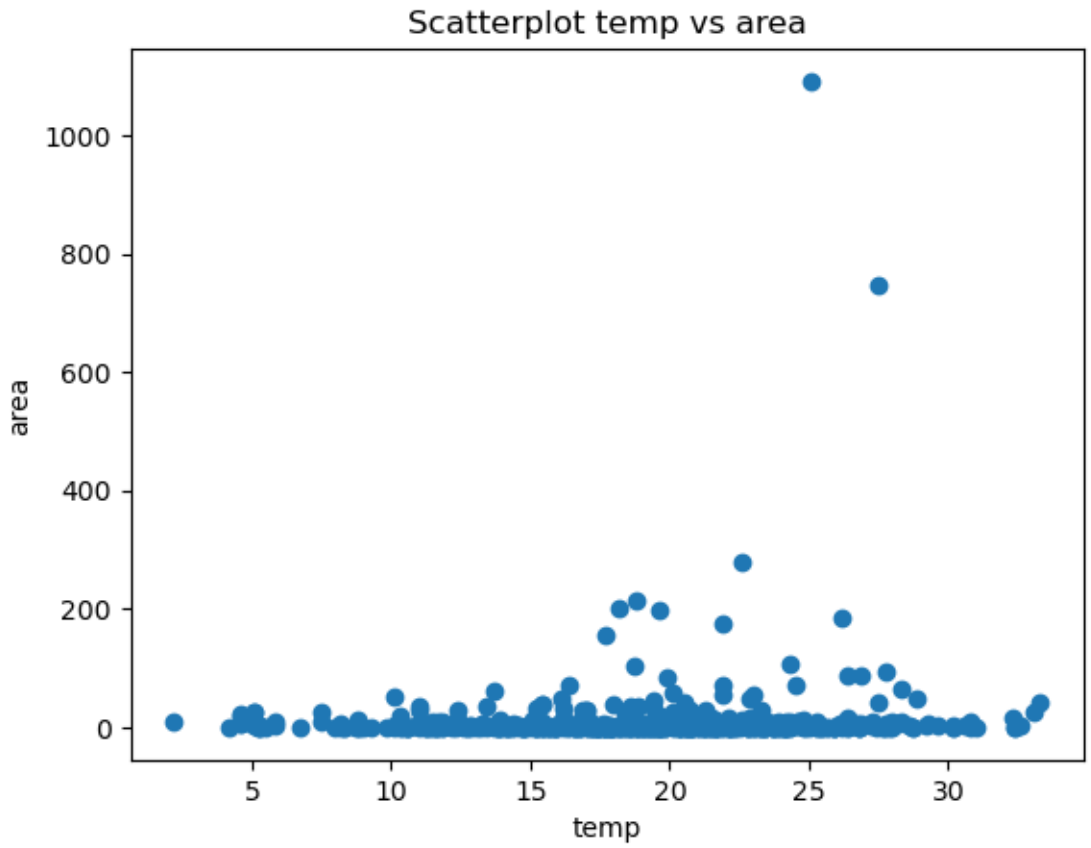
[113]:
```
#Area histogram. This distribution is not normal. There is no 'bell curve' type␣
↪distribution. This is right-skewed.
plt.hist(df['area'])
plt.title("Histogram for Area")
plt.xlabel("Area")
plt.ylabel("Count")
plt.show()
```



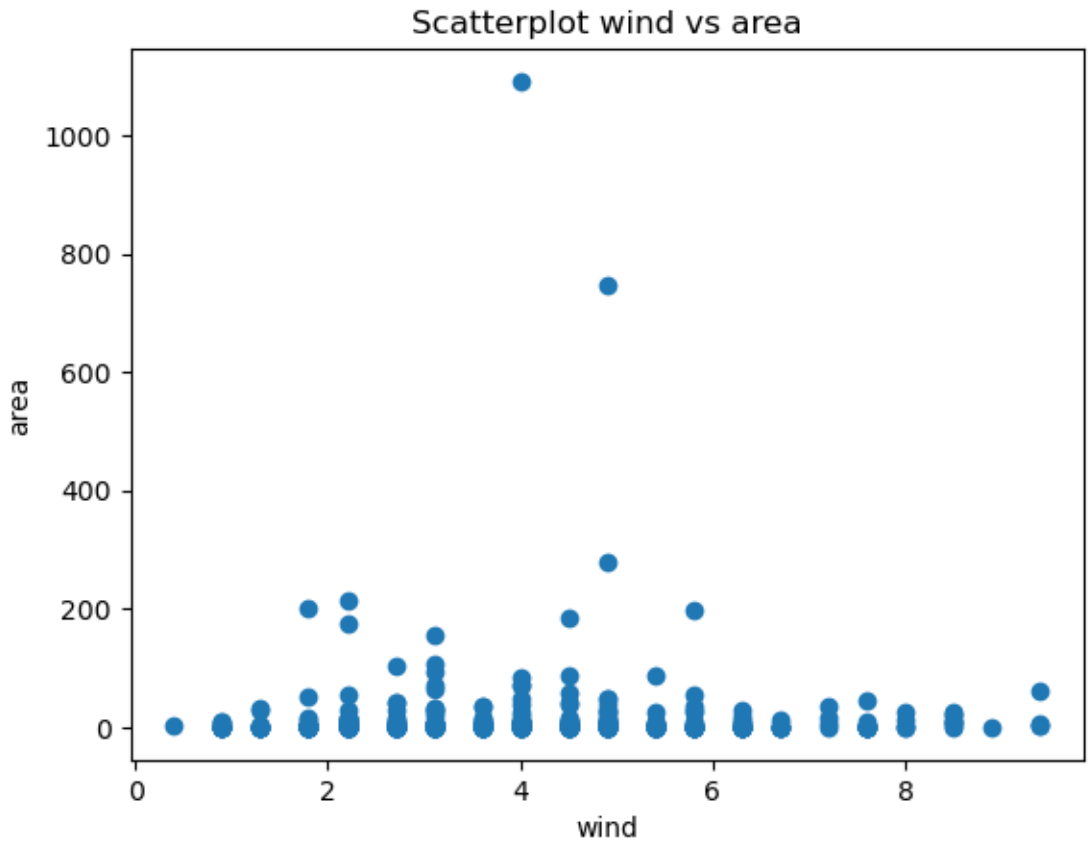
[115]:
```
#Scatterplot FFMC vs DMC
plt.scatter(df['FFMC'], df['DMC'])
plt.title("Scatterplot FFMC vs DMC")
plt.xlabel('FFMC')
plt.ylabel('DMC')
plt.show()
```



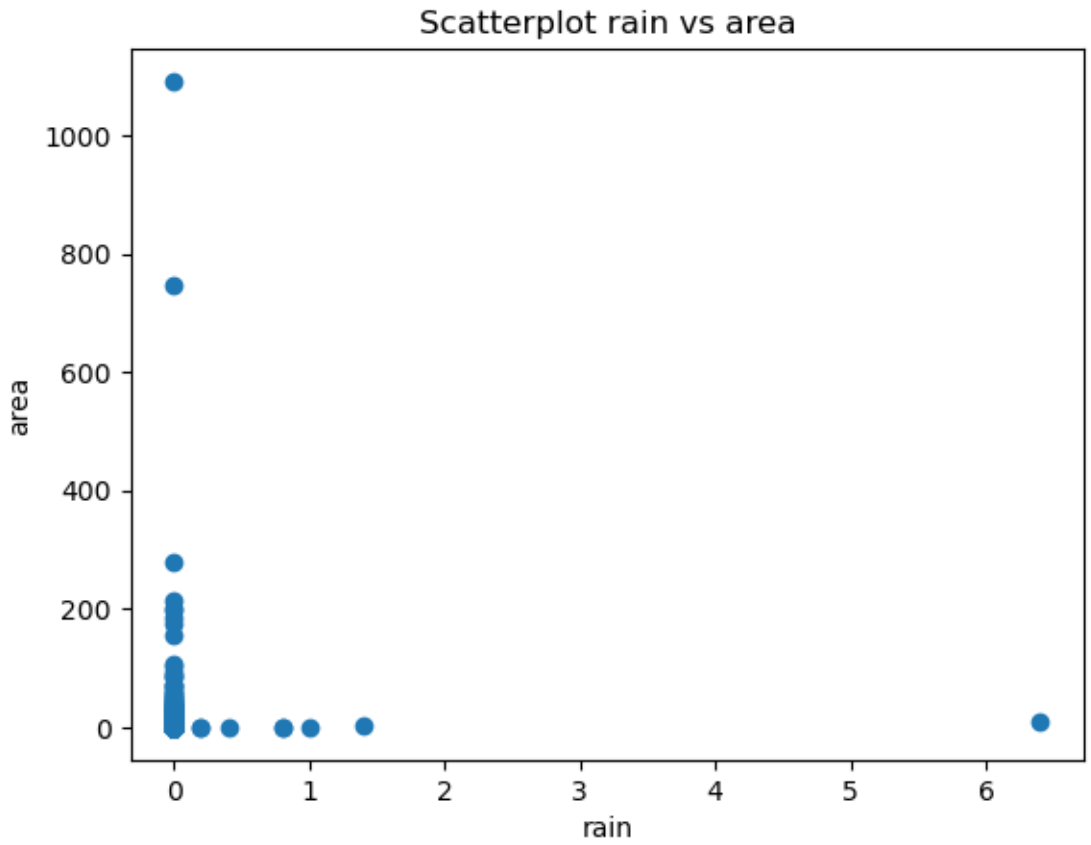
[117]:
```
#Scatterplot temp vs area
plt.scatter(df['temp'], df['area'])
plt.title("Scatterplot temp vs area")
plt.xlabel('temp')
plt.ylabel('area')
plt.show()
```



[119]:
```
#Scaaterplot wind vs area
plt.scatter(df['wind'], df['area'])
plt.title("Scatterplot wind vs area")
plt.xlabel('wind')
plt.ylabel('area')
plt.show()
```



[121]:
```
#Scatterplot rain vs area
#When there is less rain, more and larger fires occur
plt.scatter(df['rain'], df['area'])
plt.title("Scatterplot rain vs area")
plt.xlabel('rain')
plt.ylabel('area')
plt.show()
```



[101]:
```
#Data frame statistics
df["temp"].describe()
```

[101]:
```
count    517.000000
mean      18.889168
std        5.806625
min        2.200000
25%       15.500000
50%       19.300000
75%       22.800000
max       33.300000
Name: temp, dtype: float64
```

[125]:
```
#Minimum and Maximum temps in celsius
min_temp = min(df["temp"])
max_temp = max(df["temp"])
print(f'Minimum temp in celsius: {min_temp}')
print(f'Maximum temp in celsius: {max_temp}')
```

```
Minimum temp in celsius: 2.2
```

```
Maximum temp in celsius: 33.3
```

[127]:
```python
#Minimum and Maximum temps in fahrenheit
min_temp_f = ((9/5) * min_temp) + 32
max_temp_f = ((9/5) * max_temp) + 32
print(f'Minimum temp in fahrenheit: {min_temp_f}')
print(f'Maximum temp in fahrenheit: {max_temp_f}')
```

```
Minimum temp in fahrenheit: 35.96
Maximum temp in fahrenheit: 91.94
```

[ ]:
```python

```

hw1_question3

January 26, 2025

[1]:
```
import pandas as pd
import numpy as np
import math
import statistics
```

[3]:
```
#Create dictionary of data used to create a data frame
data = {'Alcohol Consumption':[0,0.5,1.5,4,7],'Malformation Present':
 ↪[48,38,None,1,1],'Total':[17114,14502,793,None,38]}
data
```

[3]:
```
{'Alcohol Consumption': [0, 0.5, 1.5, 4, 7],
 'Malformation Present': [48, 38, None, 1, 1],
 'Total': [17114, 14502, 793, None, 38]}
```

[11]:
```
#Create data frame with the data
df = pd.DataFrame(data)
df
```

[11]:

	Alcohol Consumption	Malformation Present	Total
0	0.0	48.0	17114.0
1	0.5	38.0	14502.0
2	1.5	NaN	793.0
3	4.0	1.0	NaN
4	7.0	1.0	38.0

[15]:
```
#Total missing values count in the data frame
nulls = df.isnull().sum().sum()
print(nulls)
```

```
2
```

[19]:
```
#Mean of Malformation Present column excluding missing value
column_mean = df['Malformation Present'].mean()
print(column_mean)
```

```
22.0
```

[ ]:
```

```

hw1_question4

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	**Original**	**Clean**	**Discrepancies**		**Sorted Clean**								
2	Apple	Apple			Apple								
3	Samsung	Samsung			Apple								
4	Appel	Apple	Typographical error		Apple								
5	Nokia	Nokia			Apple								
6	Blackberry	Blackberry			Apple								
7	HTC	HTC			Apple								
8	Apple	Apple			Blackberry								
9	Samsung	Samsung			Blueberry								
10	HTC	HTC			HTC								
11	LG	LG			HTC								
12	Blueberry	Blueberry	Could be 'Blackberry but cannot assume		LG								
13	Samsung	Samsung			Motorola								
14	Samsung	Samsung			NaN								
15	APPLE	Apple	Fixed capitalization		Nokia								
16	Motorola	Motorola			Samsung								
17	Apple	Apple			Samsung								
18	Samsun	Samsung	Typographical error		Samsung								
19	Apple	Apple			Samsung								
20	Samsung	Samsung			Samsung								
21	NaN	NaN			Samsung								

a. Column B is the cleaned data with discrepancies listed in column C. The data was sorted for viewer ease in column D.

b. It was noted that there were 20 respondants but there were only 19 items listed. A NaN was added for the null value. It is unknown what this value should be based on the data provided.

Homework 1 Question 5

a. The population of interest is bank executives of all financial institutions.
b. A sample of 163 bank executives provided information on their institutions as a sampling of financial institutions because it would not be feasible to survey executives of all financial institutions. They provided information on the importance of boosting profitability and identifying growth areas.
c. The parameter of the survey is the percentage of all bank executives within financial institutions and their preference on methods of increasing profitability and growth.
d. The executives provided the following information on boosting profitability and identifying growth areas:
    a. 55% of respondents stated that they plan on a spending increase on customer experience initiatives
    b. A customer relationship management (CRM) solution was noted as the most important omnichannel strategy to implement
    c. 41% of respondents stated that digital banking enhancements is the most anticipated strategy to improve customer experience

hw1_question6

January 26, 2025

[35]:
```
import pandas as pd
import numpy as np
import math
import statistics
import matplotlib.pyplot as plt
```

[3]:
```
df = pd.read_csv('/Users/helenamabey/Downloads/retirement_funds-1.csv' )
df
```

[3]:

	Fund Number	Market Cap	Type	Assets	Turnover Ratio	Beta	SD	\
0	RF001	Large	Growth	309.9	12.21	1.15	18.72	\
1	RF002	Large	Growth	23.3	0.00	2.19	35.72	
2	RF003	Large	Growth	141.5	147.00	2.24	36.69	
3	RF004	Large	Growth	118.5	5.00	2.24	36.63	
4	RF005	Large	Growth	575.3	121.00	0.89	14.56	
..	…	…	…	…	…	…	…	
311	RF312	Small	Value	73.4	32.86	1.19	19.60	
312	RF313	Small	Value	1053.5	12.00	1.16	19.42	
313	RF314	Small	Value	48.2	201.00	1.23	20.16	
314	RF315	Small	Value	65.1	16.72	1.20	19.36	
315	RF316	Small	Value	71.3	14.00	0.84	13.79	

	Risk	1YrReturn%	3YrReturn%	5YrReturn%	10YrReturn%	Expense Ratio	\
0	Low	28.99	24.26	11.06	8.97	1.22	\
1	High	33.40	22.72	-4.89	0.02	1.90	
2	High	33.98	21.91	1.53	12.55	1.92	
3	High	33.78	21.89	1.57	12.69	1.73	
4	Low	21.62	16.47	9.40	10.30	1.41	
..	…	…	…	…	…	…	
311	Average	12.47	8.88	6.08	9.56	1.09	
312	Average	13.83	8.72	2.34	9.90	1.10	
313	Average	15.79	8.58	1.51	4.24	1.53	
314	Average	15.30	7.43	3.46	9.16	1.71	
315	Low	4.83	7.12	4.41	9.80	1.27	

	Star Rating
0	Four

```
1           Two
2           Two
3           Two
4          Five
..          …
311         Two
312       Three
313         One
314       Three
315        Four

[316 rows x 14 columns]
```

[5]:
```
filtered_df = df[['Market Cap','Risk','Star Rating']]
filtered_df
```

[5]:
```
    Market Cap     Risk Star Rating
0        Large      Low        Four
1        Large     High         Two
2        Large     High         Two
3        Large     High         Two
4        Large      Low        Five
..         …       …          …
311      Small  Average         Two
312      Small  Average       Three
313      Small  Average         One
314      Small  Average       Three
315      Small      Low        Four

[316 rows x 3 columns]
```

[87]:
```
#Used ChaptGPT to get all the fundamentals of creating a pivot table in pandas␣
 ↪(next few cells included)
df_pivot = pd.pivot_table(
    df,
    values='Fund Number',
    index=['Market Cap', 'Risk'],
    columns='Star Rating',
    aggfunc='count',
    fill_value=0
)

df_pivot
```

[87]:
```
Star Rating         Five  Four  One  Three  Two
Market Cap Risk
Large      Average     1     2    3      4    4
```

```
            High        0     0    1      0    3
            Low        10    42    6     54   23
Mid-Cap     Average     2     7    3     13   14
            Low         6    25    1     16    4
Small       Average     1     6    3     18   10
            High        0     1    6      1    1
            Low         3    15    0      4    3
```

[89]:
```
#Reordered columns to logical order
df_pivot = df_pivot[['Five', 'Four', 'Three', 'Two', 'One']]
df_pivot
```

[89]:
```
Star Rating          Five  Four  Three  Two  One
Market Cap Risk
Large       Average     1     2      4    4    3
            High        0     0      0    3    1
            Low        10    42     54   23    6
Mid-Cap     Average     2     7     13   14    3
            Low         6    25     16    4    1
Small       Average     1     6     18   10    3
            High        0     1      1    1    6
            Low         3    15      4    3    0
```

[91]:
```
#Reordered index to logical order
df_pivot = df_pivot.sort_index(level='Risk', ascending=False)

df_pivot = df_pivot.reindex(index=pd.MultiIndex.from_product(
    [df_pivot.index.get_level_values('Market Cap').unique(),
     ['High', 'Average', 'Low']],
    names=['Market Cap', 'Risk']))

df_pivot
```

[91]:
```
Star Rating          Five  Four  Three   Two  One
Market Cap Risk
Small       High      0.0   1.0    1.0   1.0  6.0
            Average   1.0   6.0   18.0  10.0  3.0
            Low       3.0  15.0    4.0   3.0  0.0
Mid-Cap     High      NaN   NaN    NaN   NaN  NaN
            Average   2.0   7.0   13.0  14.0  3.0
            Low       6.0  25.0   16.0   4.0  1.0
Large       High      0.0   0.0    0.0   3.0  1.0
            Average   1.0   2.0    4.0   4.0  3.0
            Low      10.0  42.0   54.0  23.0  6.0
```

[93]:
```
#Removed NaN values and replaced with 0 to pull numbers back to integers
#Final Pivot Table for tabulation
```

```
df_pivot = df_pivot.fillna(0).astype(int)
df_pivot
```

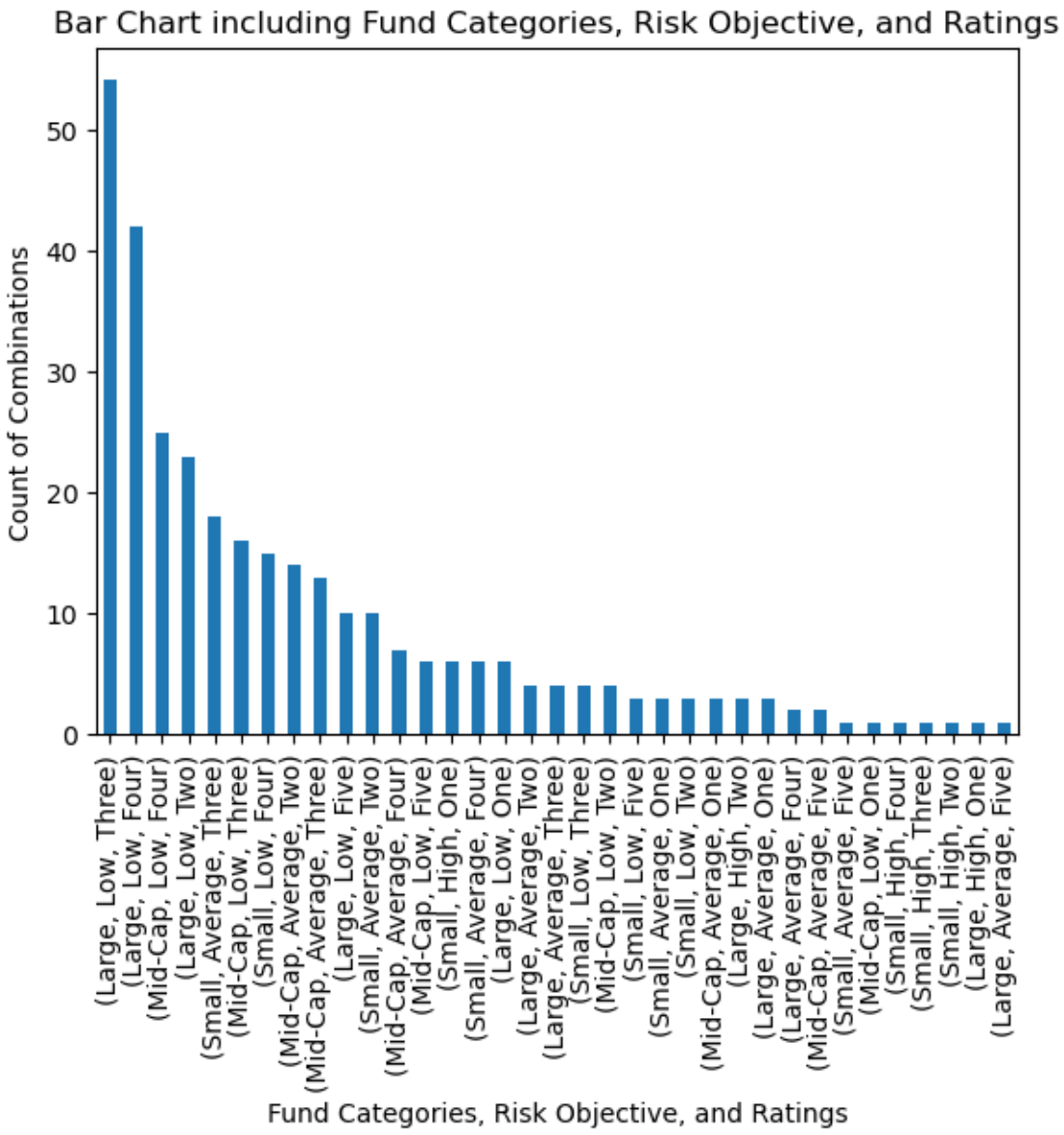
[93]:

Star Rating		Five	Four	Three	Two	One
Market Cap	Risk					
Small	High	0	1	1	1	6
	Average	1	6	18	10	3
	Low	3	15	4	3	0
Mid-Cap	High	0	0	0	0	0
	Average	2	7	13	14	3
	Low	6	25	16	4	1
Large	High	0	0	0	3	1
	Average	1	2	4	4	3
	Low	10	42	54	23	6

0.0.1 What conclusions can be reached concerning differences among the categories of funds, risk objective, and ratings?

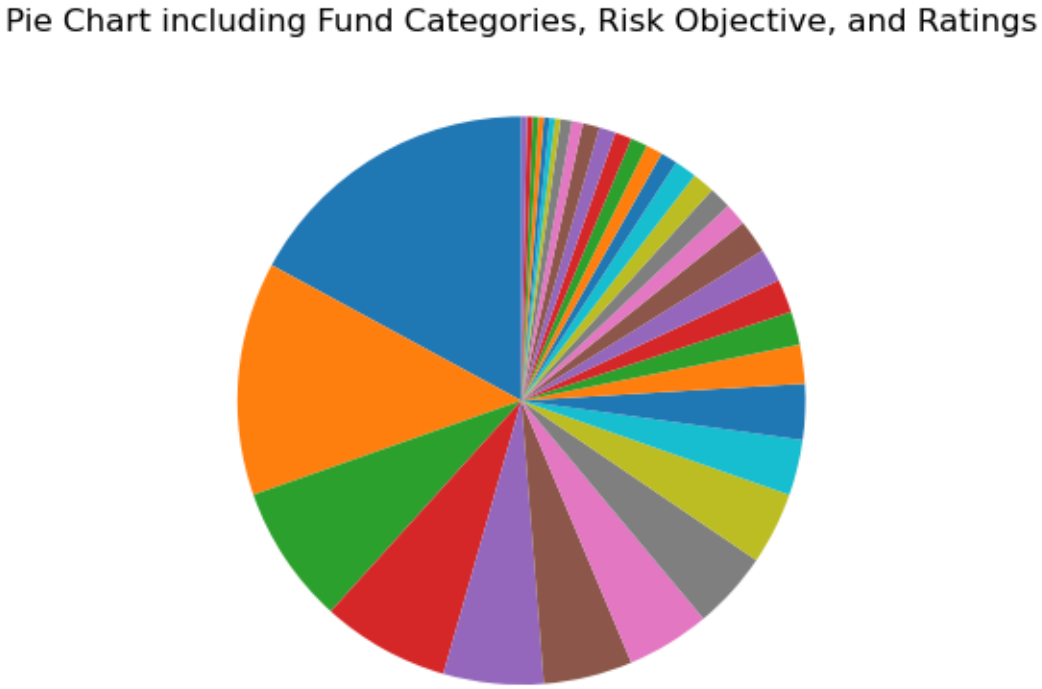
By reviewing the Retirement Funds information on categories of funds, risk objectives , and ratings, we can see that there are very few high risk funds included in this review. The majority of funds are low risk regardless of Market Cap. Most of the funds are rating moderately, between 2 and 4 with few outliers with a high rating of 5 and even fewer with a low rating of 1.

[193]:
```
#Bar Chart with three values. If you order by greatest on the left, you can␣
 ↪show the highest percentage clearly
aggregated_data = df.groupby(['Market Cap', 'Risk', 'Star Rating']).size()
aggregated_data = aggregated_data.sort_values(ascending=False)

aggregated_data.plot(kind='bar')
plt.title('Bar Chart including Fund Categories, Risk Objective, and Ratings')
plt.xlabel('Fund Categories, Risk Objective, and Ratings')
plt.ylabel('Count of Combinations')
plt.show()
```

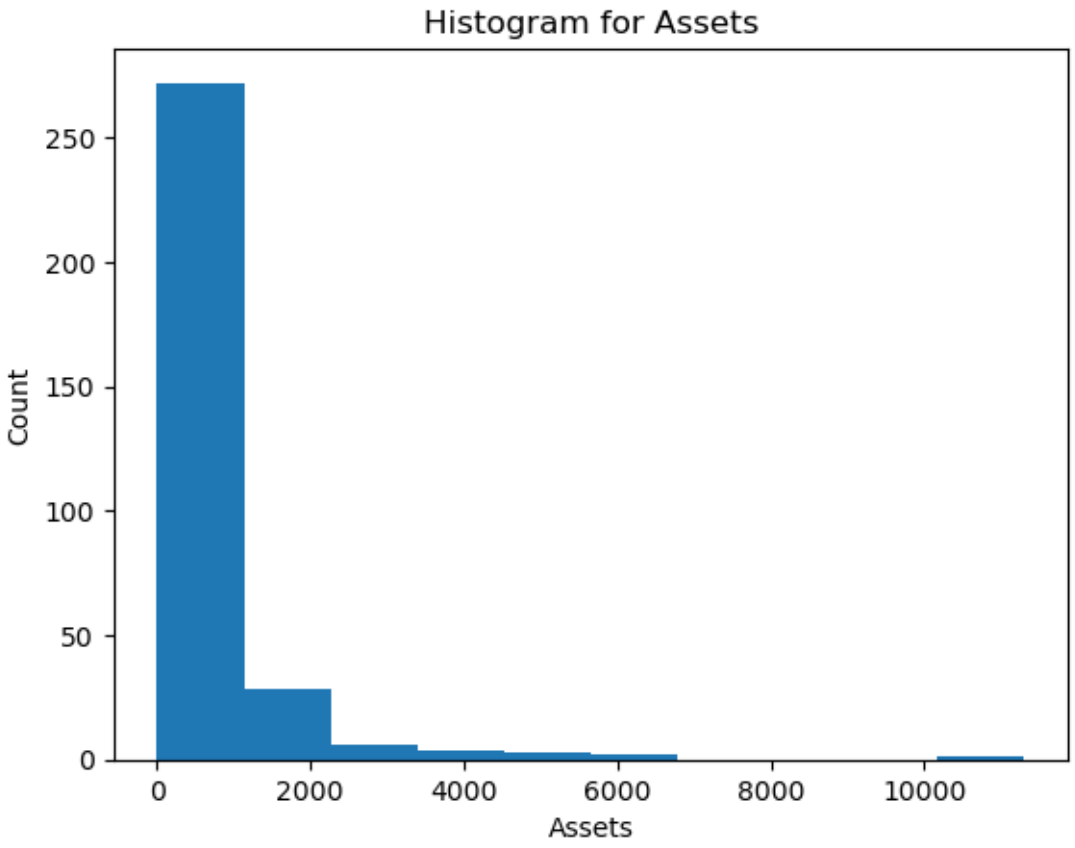


[169]:
```
#Pie Chart with three values. I had to remove all labels and the legend to make␣
 ↪this even readable.
#Pie charts should not be used for more than three or four values or they␣
 ↪become meaningless.
aggregated_data.plot(
    kind='pie',
    #autopct='%1.1f%%',
    startangle=90,
    legend=False,
    labels=None
)
```

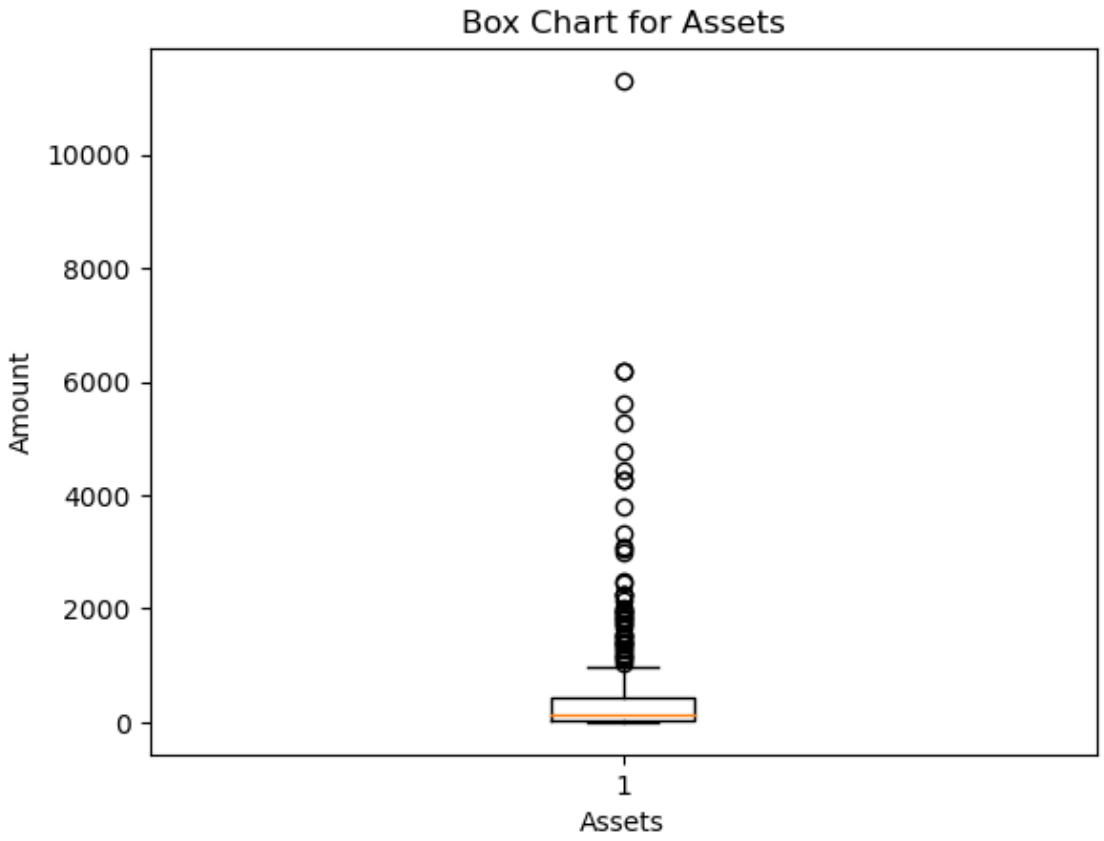
```
plt.title('Pie Chart including Fund Categories, Risk Objective, and Ratings')
plt.show()
```



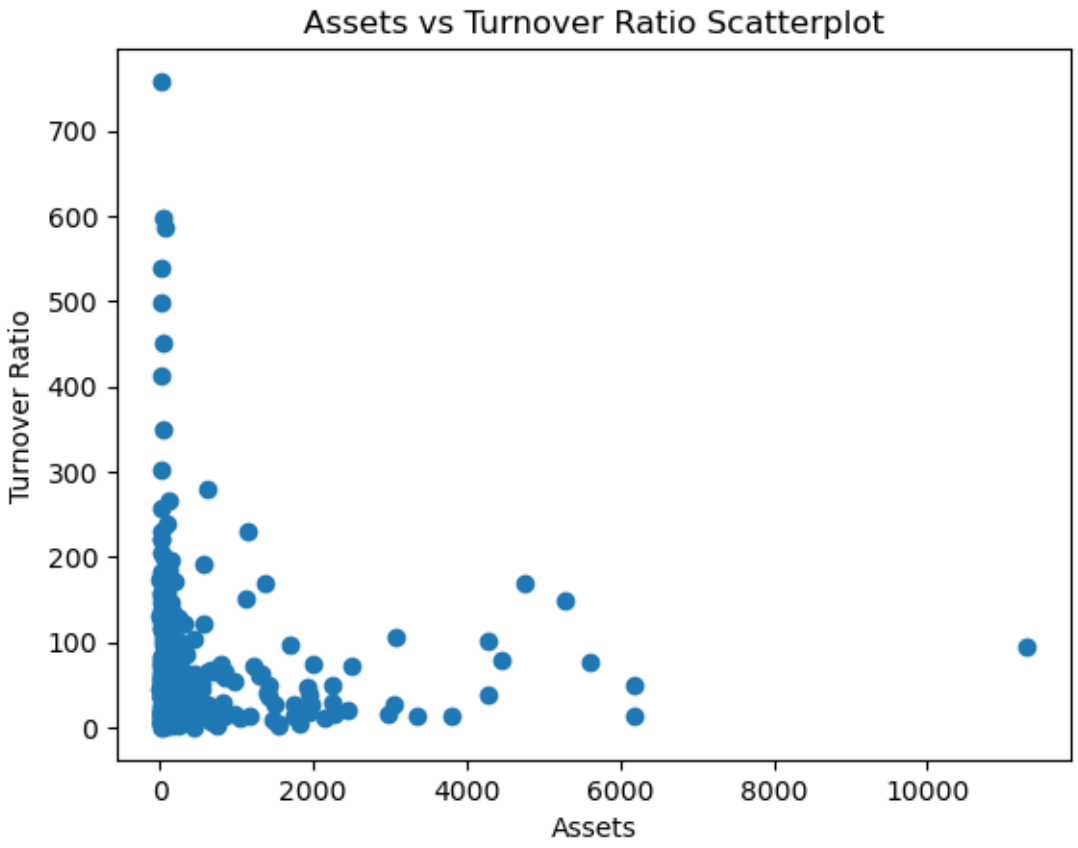
[171]:
```
#Histogram for Assets
plt.hist(df['Assets'])
plt.title("Histogram for Assets")
plt.xlabel("Assets")
plt.ylabel("Count")
plt.show()
```



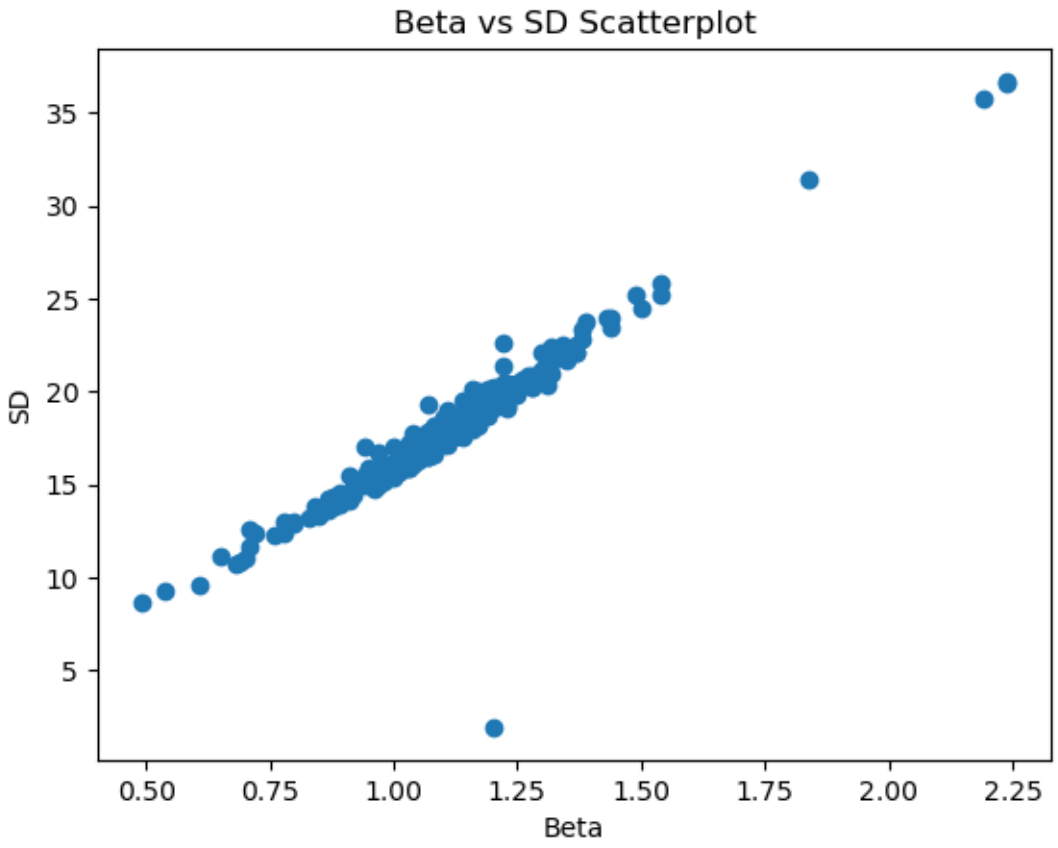
[199]:
```
#Box Chart for Assets
plt.boxplot(df['Assets'])
plt.title("Box Chart for Assets")
plt.xlabel("Assets")
plt.ylabel("Amount")
#plt.ylim(-50, 6000)
plt.show()
```



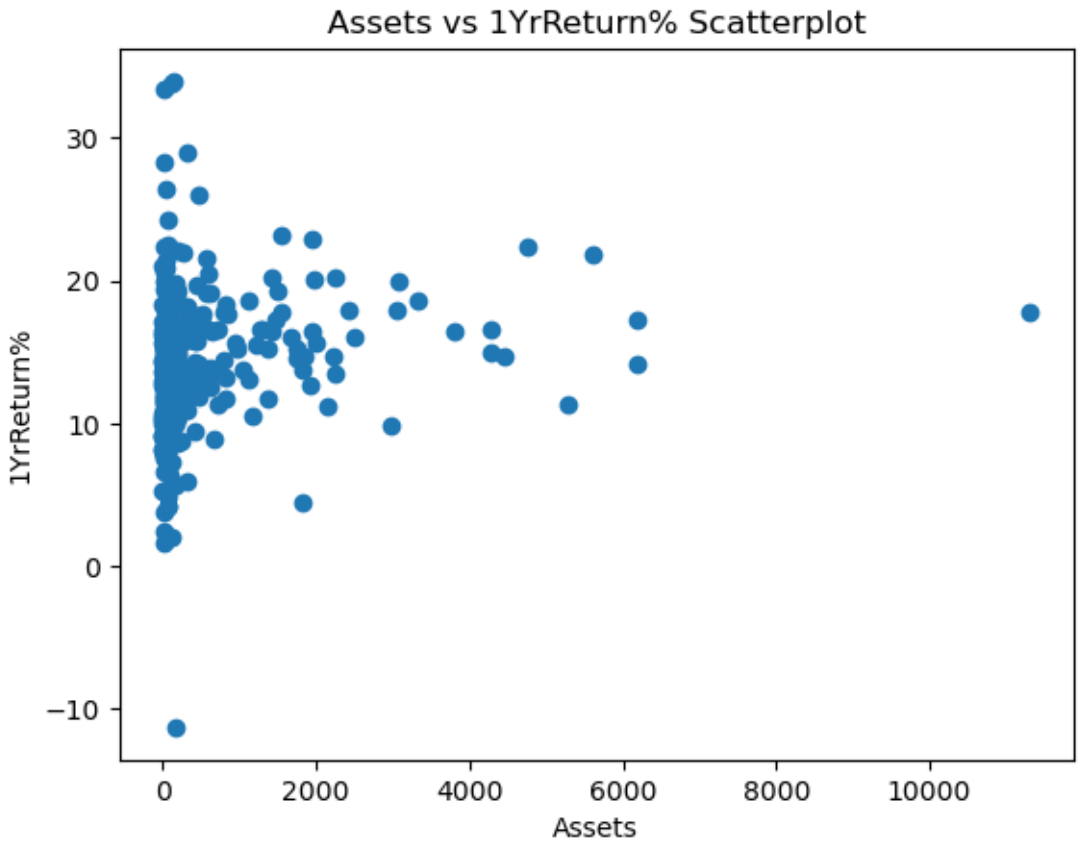
[179]:
```
#The next charts are a combination of the numeric variables in multiple pair␣
 ↪combinations.
plt.scatter(df['Assets'], df['Turnover Ratio'])
plt.title("Assets vs Turnover Ratio Scatterplot")
plt.xlabel("Assets")
plt.ylabel("Turnover Ratio")
plt.show()
```



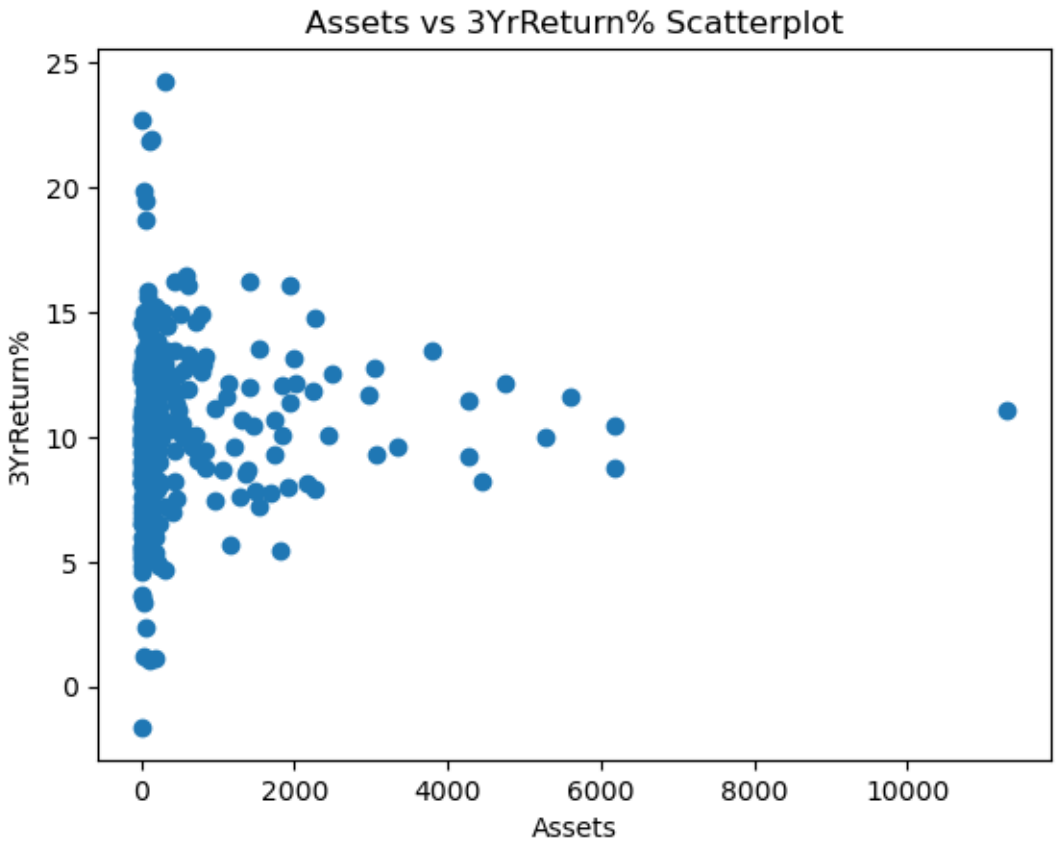
[181]:
```
plt.scatter(df['Beta'], df['SD'])
plt.title("Beta vs SD Scatterplot")
plt.xlabel("Beta")
plt.ylabel("SD")
plt.show()
```



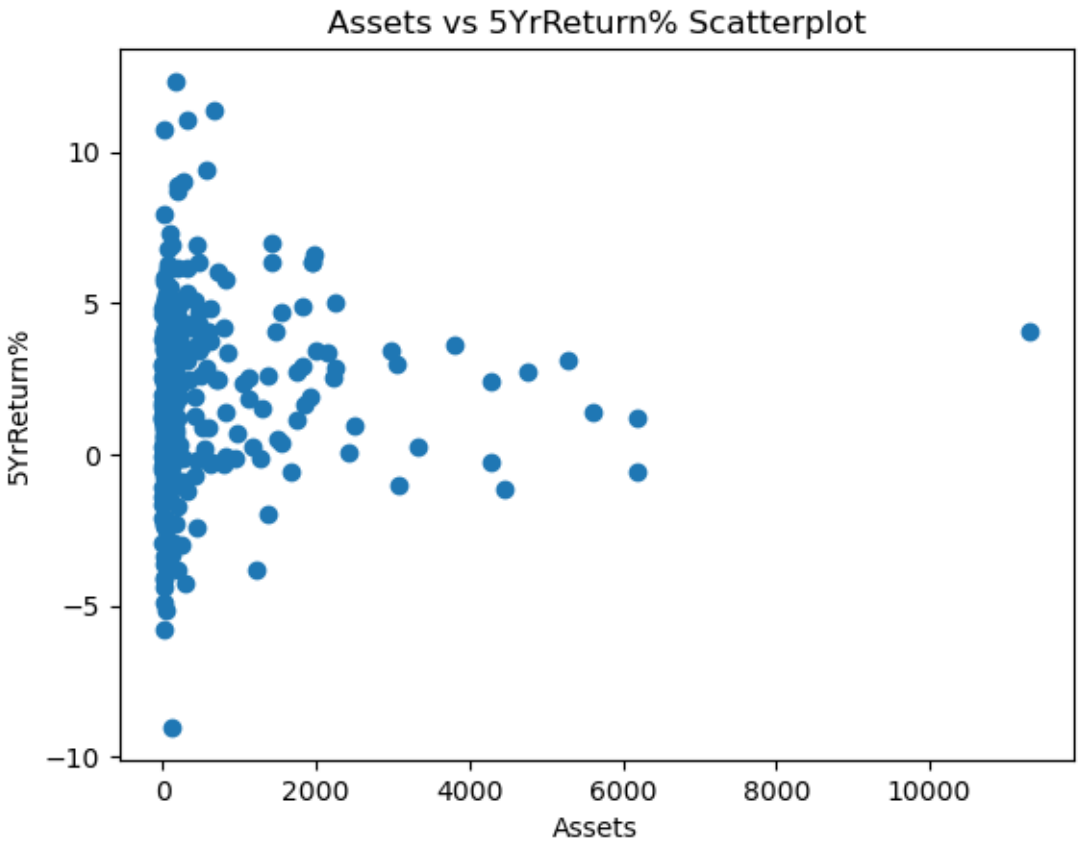
[183]:
```
plt.scatter(df['Assets'], df['1YrReturn%'])
plt.title("Assets vs 1YrReturn% Scatterplot")
plt.xlabel("Assets")
plt.ylabel("1YrReturn%")
plt.show()
```



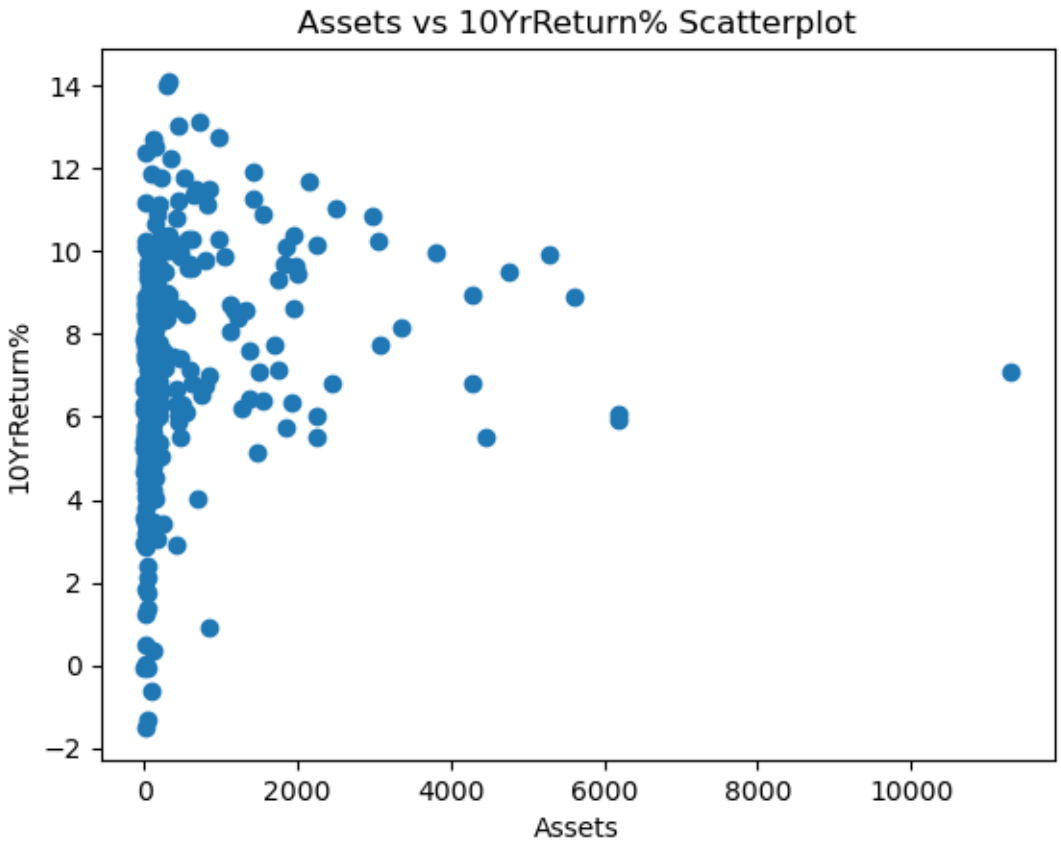
[185]:
```
plt.scatter(df['Assets'], df['3YrReturn%'])
plt.title("Assets vs 3YrReturn% Scatterplot")
plt.xlabel("Assets")
plt.ylabel("3YrReturn%")
plt.show()
```



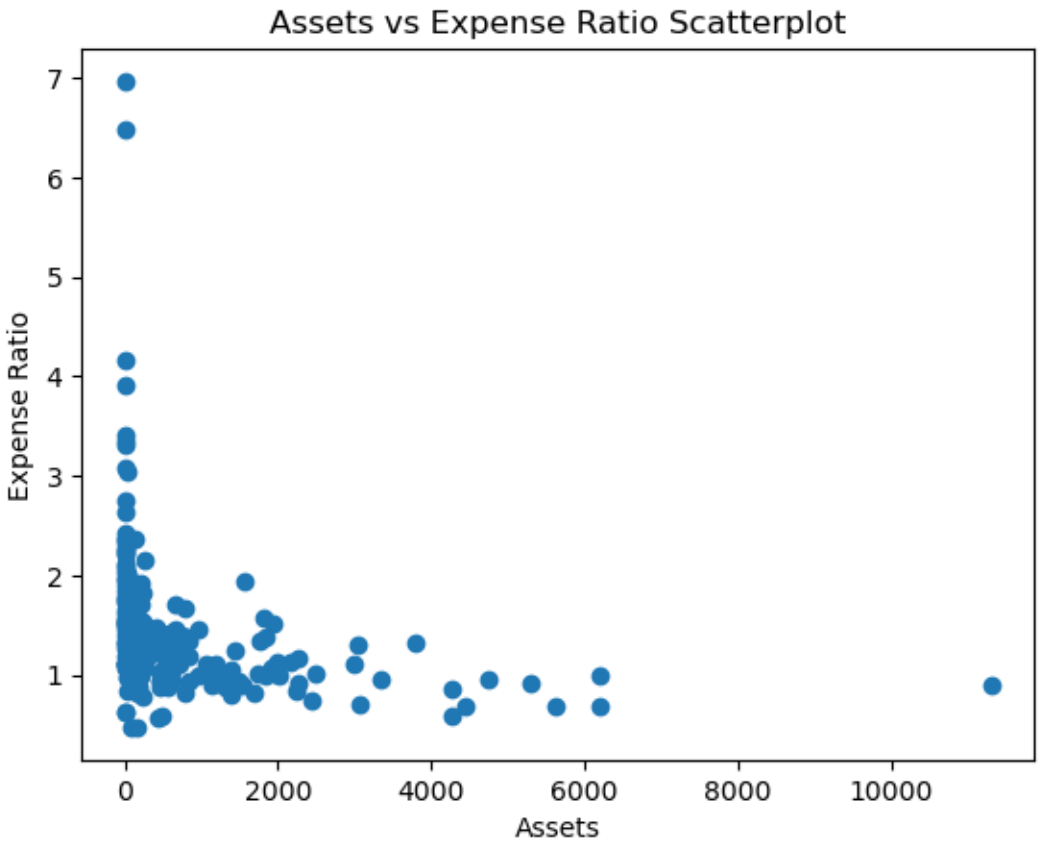
[187]:
```
plt.scatter(df['Assets'], df['5YrReturn%'])
plt.title("Assets vs 5YrReturn% Scatterplot")
plt.xlabel("Assets")
plt.ylabel("5YrReturn%")
plt.show()
```



[201]:
```
plt.scatter(df['Assets'], df['10YrReturn%'])
plt.title("Assets vs 10YrReturn% Scatterplot")
plt.xlabel("Assets")
plt.ylabel("10YrReturn%")
plt.show()
```



[203]:
```
plt.scatter(df['Assets'], df['Expense Ratio'])
plt.title("Assets vs Expense Ratio Scatterplot")
plt.xlabel("Assets")
plt.ylabel("Expense Ratio")
plt.show()
```



0.0.2 What conclusions can be reached on retirement funds can be reached based on the above charts?

By reviewing the bar chart for categories of funds, risk objective, and ratings we can see that many of the funds reviewed were large cap, low risk, with a moderate rating of four or five. By sorting the bar chart from highest to lowest (not ascending), this can be seen at a glance. The pie chart does not provide insight because the number of combinations was so great that the pie slices were indistinguishable. It can still be seen that there were two large segments but, because of the number of slices, labels were removed to allow a view of the chart. In a review of the assets, the histogram shows a solid right-skew with the majority of the assets falling below 2000. There is a single outlier at 10000. A similar trend can be seen in the box and whisker chart. This chart is skewed because of the one outlier making it challenging to obtain detailed information about the results. A few numeric pairs were run as scatterplots to see if there was any correlation of the data. It does appear that there is a relationship between asset amount and the turnover ratio. The lower the asset amount, the more likely there is to be a higher turnover percentage. As the asset amount grows, the turnover ratio stabilizes. Another numeric pair that appears to have a direct relationship are Beta and SD. As Beta increases, SD increases at a constaint rate. The assets in ratio to the 1, 3, 5, and 10 year return percentages each

follow a similar pattern based on asset size. This pattern carries over regardless of the number of years. The percentages increase and decease over time but the pattern remains. The outlier pattern also remains contstant along side the majority.

[ ]:

hw1_question7

January 26, 2025

[1]:
```
import pandas as pd
import numpy as np
import math
import statistics
import matplotlib.pyplot as plt
```

[9]:
```
df = pd.read_excel('/Users/helenamabey/Downloads/CEO-Compensation.xlsx' )
df
```

[9]:

	Company	Compensation ($millions)	Return in 2012 (%)
0	Abbott Laboratories	19.0	19.0
1	Adobe Systems	12.0	26.0
2	AES	7.7	-10.0
3	Aflac	11.2	26.0
4	Agilent Technologies	10.1	-2.0
..	…	…	…
165	Weyerhaeuser	6.4	52.0
166	Whirlpool	12.9	119.0
167	Whole Foods Market	1.3	50.0
168	Wisconsin Energy	9.1	9.0
169	XL Group	9.4	29.0

[170 rows x 3 columns]

[ ]:
```
#Used ChatGPT for definitions of frequency and percentage distribution as well␣
↪as gathering syntax to gather the data
```

[15]:
```
#Frequency distribution
comp_freq = df['Compensation ($millions)'].value_counts()
print(comp_freq)
```

```
Compensation ($millions)
10.3    5
8.4     3
8.6     3
9.1     3
7.9     3
       ..
```

```
8.8     1
29.7    1
5.9     1
8.3     1
9.4     1
Name: count, Length: 115, dtype: int64
```

[17]:
```
#Frequency distribution
return_freq = df['Return in 2012 (%)'].value_counts()
print(return_freq)
```

```
Return in 2012 (%)
 19.0     8
 15.0     8
 29.0     6
 26.0     5
 24.0     5
         ..
-28.0     1
 107.0    1
 89.0     1
-15.0     1
 119.0    1
Name: count, Length: 79, dtype: int64
```

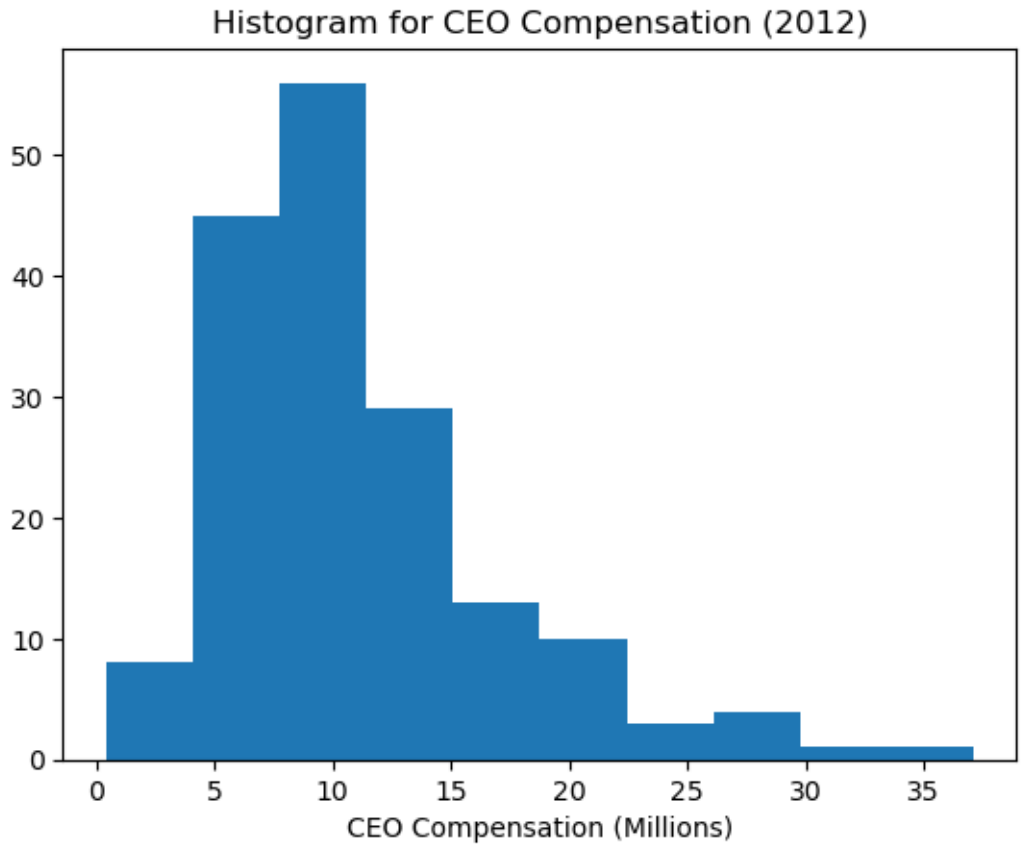
[19]:
```
# Percentage distribution
comp_perc = df['Compensation ($millions)'].value_counts(normalize=True) * 100

print(comp_perc)
```

```
Compensation ($millions)
10.3    2.941176
8.4     1.764706
8.6     1.764706
9.1     1.764706
7.9     1.764706
          …
8.8     0.588235
29.7    0.588235
5.9     0.588235
8.3     0.588235
9.4     0.588235
Name: proportion, Length: 115, dtype: float64
```

[21]:
```
# Percentage distribution
return_perc = df['Return in 2012 (%)'].value_counts(normalize=True) * 100

print(return_perc)
```

```
Return in 2012 (%)
 19.0     4.705882
 15.0     4.705882
 29.0     3.529412
 26.0     2.941176
 24.0     2.941176
            …
-28.0     0.588235
 107.0    0.588235
 89.0     0.588235
-15.0     0.588235
 119.0    0.588235
Name: proportion, Length: 79, dtype: float64
```

[141]:
```
#Histogram of Compensation (Millions)
plt.hist(df['Compensation ($millions)'])
plt.title('Histogram for CEO Compensation (2012)')
plt.xlabel("CEO Compensation (Millions)")
plt.show()
```



[148]:
```
#Histogram of Investment Return (%)
plt.hist(df['Return in 2012 (%)'])
plt.title('Histogram for Investment Return (2012)')
plt.xlabel("Investment Return (%)")
plt.show()
```

[ ]:
```
#Used ChatGPT for definition of a percentage polygon. Code was provided for a␣
 ↪general example and adjusted for
#this specific use case. The following cells are for the percentage polygons␣
 ↪for compensation and investment return.
```

[69]:
```
#CEO Compensation (Millions) percentage polygon creation
comp_bins = [0,7,14,21,28,35,42]
df['Comp_Bins'] = pd.cut(df['Compensation ($millions)'], bins=comp_bins)

comp_frequency = df['Comp_Bins'].value_counts().sort_index()
comp_percentage = (comp_frequency / comp_frequency.sum()) * 100
```

[71]:
```
print(comp_frequency)
```

```
Comp_Bins
(0, 7]      43
(7, 14]     93
(14, 21]    22
(21, 28]     7
(28, 35]     4
(35, 42]     1
Name: count, dtype: int64
```

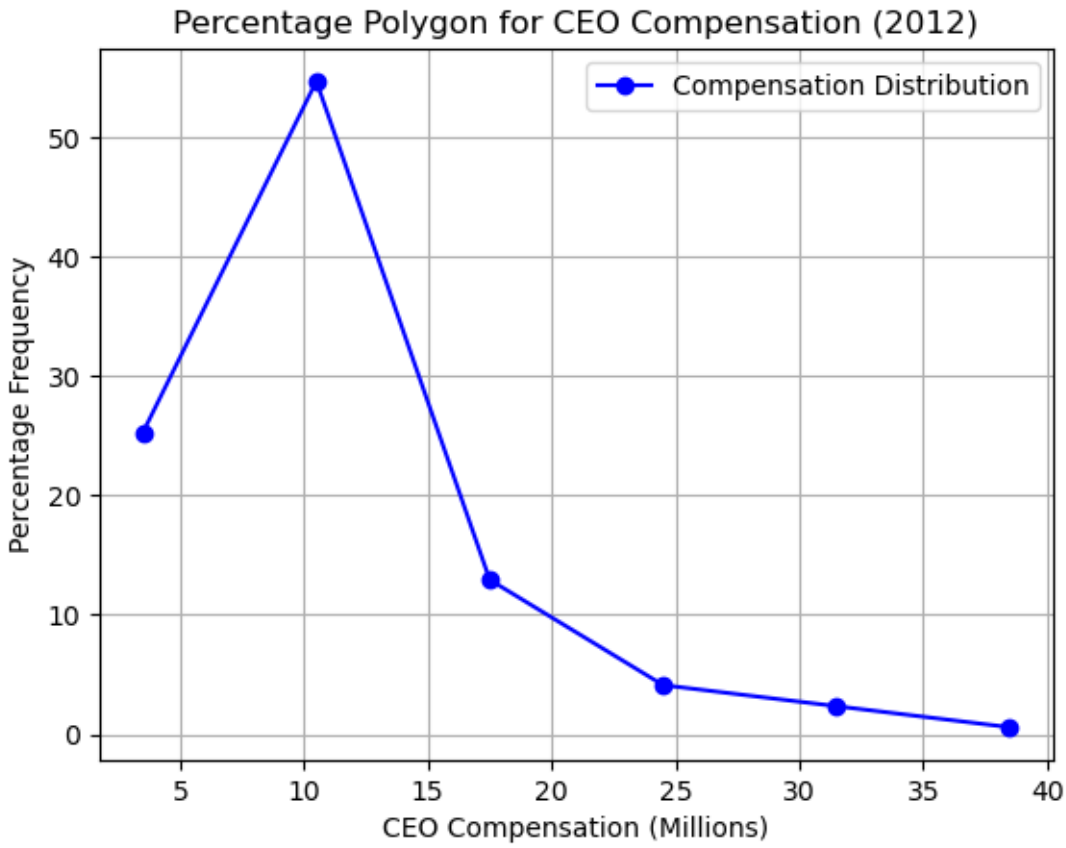
[73]:
```
print(comp_percentage)
```

```
Comp_Bins
(0, 7]      25.294118
(7, 14]     54.705882
(14, 21]    12.941176
(21, 28]     4.117647
(28, 35]     2.352941
(35, 42]     0.588235
Name: count, dtype: float64
```

[75]:
```
comp_bin_mid = [interval.mid for interval in comp_frequency.index]
print(comp_bin_mid)
```

```
[3.5, 10.5, 17.5, 24.5, 31.5, 38.5]
```

[139]:
```
plt.plot(comp_bin_mid, comp_percentage, marker='o', linestyle='-',␣
  ↪color='blue', label='Compensation Distribution')

plt.title('Percentage Polygon for CEO Compensation (2012)')
plt.xlabel('CEO Compensation (Millions)')
plt.ylabel('Percentage Frequency')
plt.grid(True)
plt.legend()
plt.show()
```



[59]:
```
#Investment Return (%) percentage polygon creation
return_bins = [-56,-28,0,28,56,84,112,140]
df['Return_Bins'] = pd.cut(df['Return in 2012 (%)'], bins=return_bins)

ret_frequency = df['Return_Bins'].value_counts().sort_index()
ret_percentage = (ret_frequency / ret_frequency.sum()) * 100
```

[61]:
```
print(ret_frequency)
```

```
Return_Bins
(-56, -28]     4
(-28, 0]      26
(0, 28]       94
(28, 56]      28
(56, 84]      14
(84, 112]      3
(112, 140]     1
Name: count, dtype: int64
```

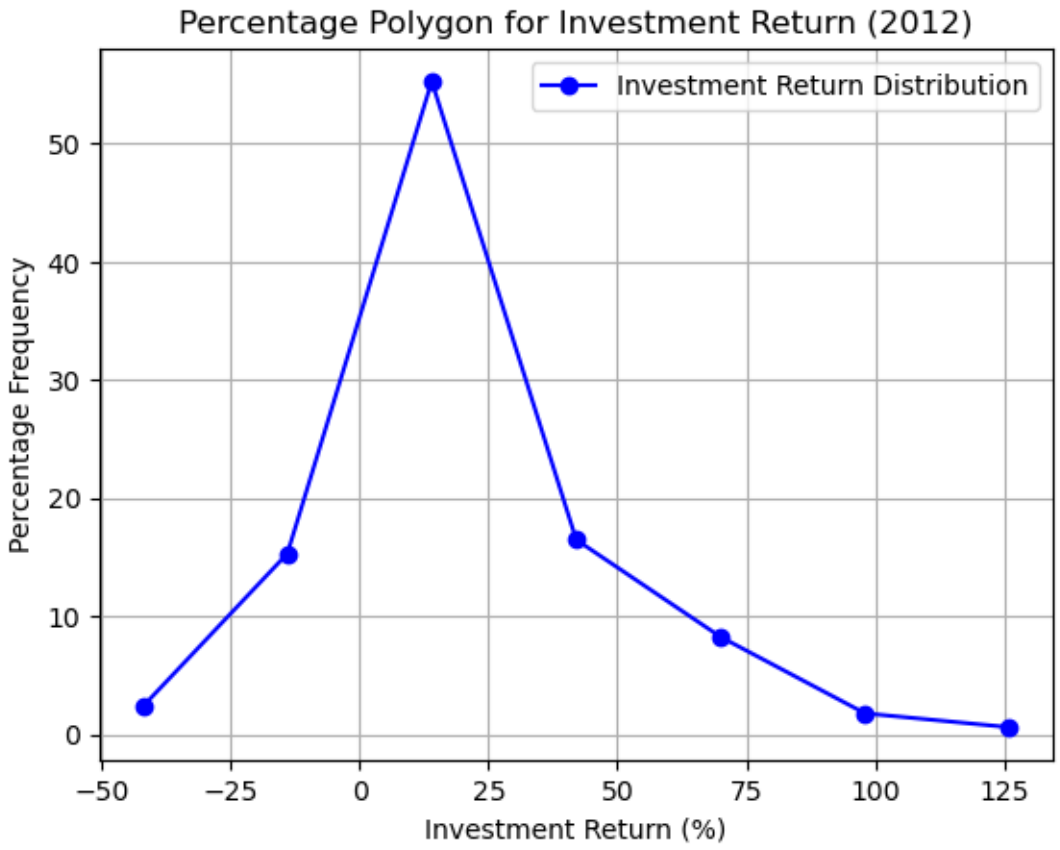
[63]:
```
print(ret_percentage)
```

```
Return_Bins
(-56, -28]     2.352941
(-28, 0]      15.294118
(0, 28]       55.294118
(28, 56]      16.470588
(56, 84]       8.235294
(84, 112]      1.764706
(112, 140]     0.588235
Name: count, dtype: float64
```

[65]:
```
ret_bin_mid = [interval.mid for interval in ret_frequency.index]
print(ret_bin_mid)
```

```
[-42.0, -14.0, 14.0, 42.0, 70.0, 98.0, 126.0]
```

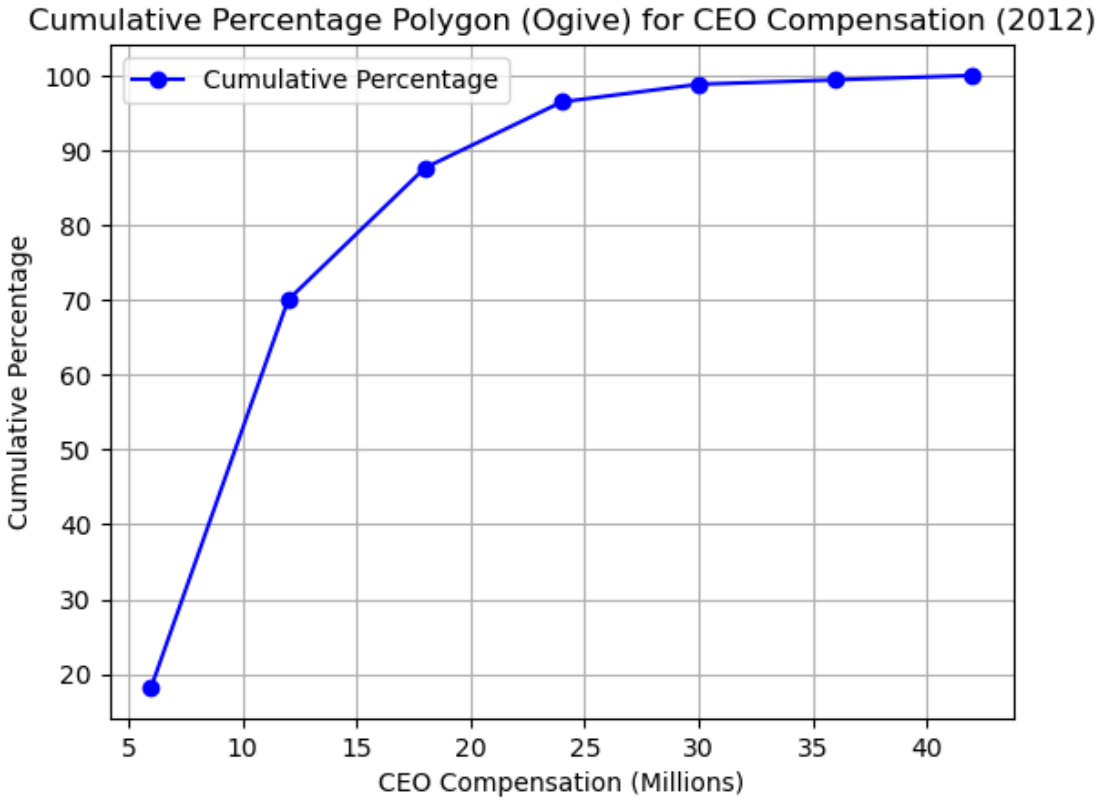
[146]:
```
plt.plot(ret_bin_mid, ret_percentage, marker='o', linestyle='-', color='blue',␣
  ↪label='Investment Return Distribution')

plt.title('Percentage Polygon for Investment Return (2012)')
plt.xlabel('Investment Return (%)')
plt.ylabel('Percentage Frequency')
plt.grid(True)
plt.legend()
plt.show()
```



[ ]:
```
#Used ChatGPT for definition of a cumulative percentage polygon (ogive). Code␣
 ↪was provided for a general example and adjusted for
#this specific use case. The following cells are for the cumulative percentage␣
 ↪polygons (ogives) for compensation and investment return.
#Used similar bins and labels as the percentage polygons for comparison.
```

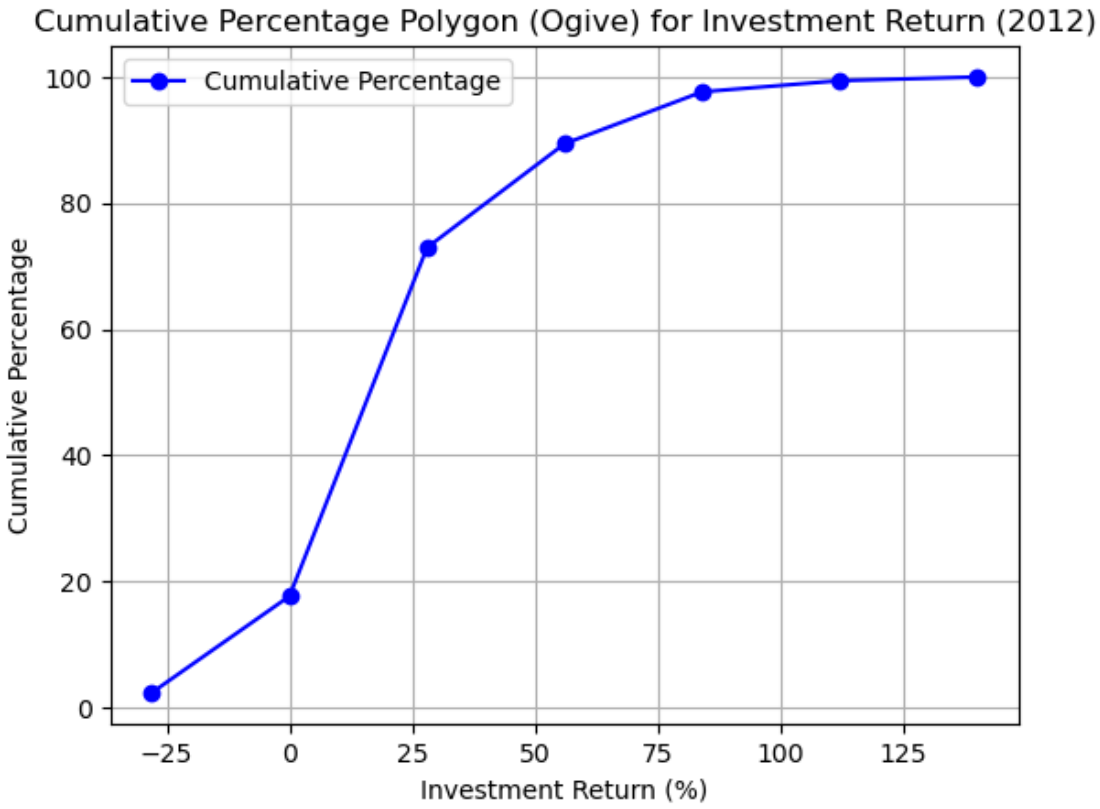
[137]:
```
#CEO Compensation (Millions) cumulative percentage polygon (ogive) creation
ogive_comp_bins = [0, 6, 12, 18, 24, 30, 36, 42]
ogive_comp_labels = ['0-6', '6-12', '12-18', '18-24', '24-30', '30-36', '36-42']

df['Compensation_Ogive_Comp_Bins'] = pd.cut(df['Compensation ($millions)'],␣
 ↪bins=ogive_comp_bins, labels=ogive_comp_labels, include_lowest=True)

ogive_comp_frequency = df['Compensation_Ogive_Comp_Bins'].value_counts().
 ↪sort_index()
cumulative_ogive_comp_frequency = ogive_comp_frequency.cumsum()

total_ogive_comp_frequency = ogive_comp_frequency.sum()
cumulative_ogive_comp_percentage = (cumulative_ogive_comp_frequency /␣
 ↪total_ogive_comp_frequency) * 100
```

```
ogive_comp_upper_boundaries = ogive_comp_bins[1:]

plt.plot(ogive_comp_upper_boundaries, cumulative_ogive_comp_percentage,␣
 ↪marker='o', linestyle='-', color='blue', label='Cumulative Percentage')

plt.title('Cumulative Percentage Polygon (Ogive) for CEO Compensation (2012)')
plt.xlabel('CEO Compensation (Millions)')
plt.ylabel('Cumulative Percentage')
plt.grid(True)
plt.legend()
plt.show()
```



[144]:
```
#Investment Return (%) cumulative percentage polygon (ogive) creation
ogive_return_bins = [-56,-28,0,28,56,84,112,140]
ogive_return_labels = ['-56 - -28', '-28-0', '0-28', '28-56', '56-84',␣
 ↪'84-112', '112-140']

df['Return_Ogive_Bins'] = pd.cut(df['Return in 2012 (%)'],␣
 ↪bins=ogive_return_bins, labels=ogive_return_labels, include_lowest=True)
```

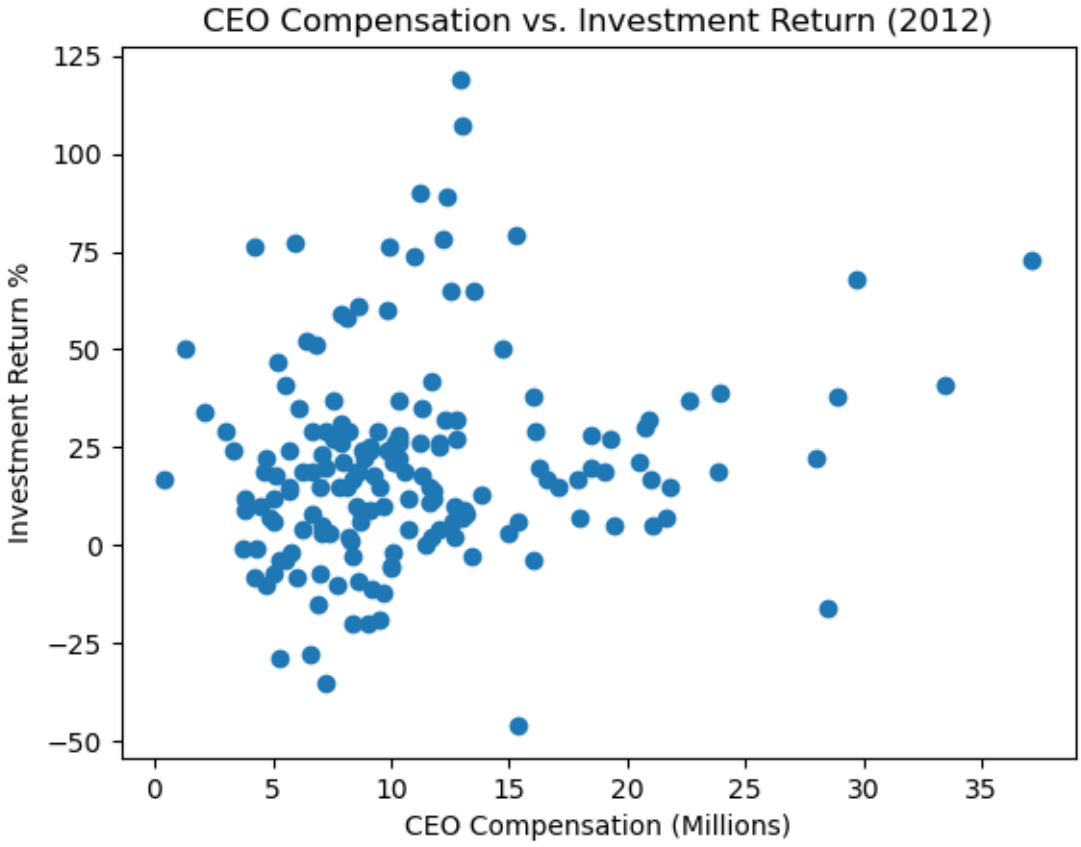
```
ogive_return_frequency = df['Return_Ogive_Bins'].value_counts().sort_index()
cumulative_ogive_return_frequency = ogive_return_frequency.cumsum()

total_ogive_return_frequency = ogive_return_frequency.sum()
cumulative_ogive_return_percentage = (cumulative_ogive_return_frequency /␣
 ↪total_ogive_return_frequency) * 100

ogive_return_upper_boundaries = ogive_return_bins[1:]

plt.plot(ogive_return_upper_boundaries, cumulative_ogive_return_percentage,␣
 ↪marker='o', linestyle='-', color='blue', label='Cumulative Percentage')

plt.title('Cumulative Percentage Polygon (Ogive) for Investment Return (2012)')
plt.xlabel('Investment Return (%)')
plt.ylabel('Cumulative Percentage')
plt.grid(True)
plt.legend()
plt.show()
```



0.0.1 CEO Compensation conclusions

In a review of the CEO compensation and investment return percentage data a few trends are presented. The CEO Compensation has slight right-skew but is generally normally distributed. The investment return percentages are even more normally distributed. The majority of CEOs make less than $14M with a sharp decline after reaching that amount. Over 75% of companies report a CEO with compensation under this amount. The investment return percentage has a similar peak under 25% with over 70% of companies reporting an investment return percentage under this amount. These trends are more clearly outlined in the cumulative percentage polygon charts. The climb is very steep and reaches a near-max percentage within the first third of the graph for CEO compensation and within the first half of the graph for investment return percentage.

[135]:
```
#CEO Compensation (Millions) vs Investment Return (%) scatterplot
plt.scatter(df['Compensation ($millions)'], df['Return in 2012 (%)'])
plt.title('CEO Compensation vs. Investment Return (2012)')
plt.xlabel('CEO Compensation (Millions)')
plt.ylabel('Investment Return %')
plt.show()
```



0.0.2 Relationship between CEO compensation and investment return percentage in 2012

The scatterplot solidifies the prior conclusions that the majority of CEO compensation is under \$14M and most investment return percentages are under 25%. There is a small trend that could be investigated for CEO compensation around \$15M where the investment return percentage has an increase. There may be other variables contributing to these outliers. There is a small population of outliers in the CEO compensation above \$25M that all have an investment return percentage above 25%. This too could be attributed to additional variables not included in the given data set.

[ ]:

AI Statement Homework 1

During this exercise, I utilized multiple sources online to assist in learning and understanding definitions and appropriate Python coding. In my notebooks, I cited references to ChatGPT as they were used and the purpose for which they were used. Outside of ChatGPT, I utilized webpage resources including W3 Schools and Stack Overflow forums for suggestions on syntax and coding assistance.