

```
1  # %%
2
3  # =====
4  # PANDAS EXERCISES
5  # =====
6
7
8  # %%
9
10 # excercise 1
11
12 # importing the panadas
13
14 import numpy as np
15 import pandas as pd
16
17 # %%
18
19 # creating a dataframe fruits
20
21 fruits = pd.DataFrame({'Apples': [30], 'Bananas': [21]})
22
23 fruits
24
25 # %%
26
27 # creating a dataframe fruit_sales
28
29 fruit_sales = pd.DataFrame({"Apples": [35, 41], "Bananas": [
30     21, 34]}, index=["2017 Sales", "2018 Sales"])
31
32 fruit_sales
33
34
35 # %%
36
37
38 # creating a series ingredients
```

```
39 # creating a series ingredients
40 ingredients = pd.Series(
41     {"Flour": "4 Cups", "Milk": "1 Cup", "Eggs": "2 large", "Spam": "1 Can"})
42
43 ingredients
44
45 # %%
46
47
48 # reading the csv dataset of wine reviews into a Dataframe reviews
49
50
51 reviews = pd.read_csv("~/KUBUNTU/DATASET/CSV_FILES/winemag-data_first150k.csv")
52
53 reviews.head()
54
55 # %%
56
57
58 # describe the dataset
59
60 desc = reviews.describe()
61
62
63 # %%
64
65 # create and display a DataFrame called `animals`
66
67 animals = pd.DataFrame({'Cows': [12, 20], 'Goats': [22, 19]}, index=[
68     'Year 1', 'Year 2'])
69
70 animals
71
72
73 # %%
74
75
76 # code to save this DataFrame to disk as a csv file with the name `cows_and_goats.csv`
```

```
77
78 animals.to_csv("~/KUBUNTU/DATASET/CSV_FILES/cows_and_goats.csv")
79
80
81 # %%
82
83 # excercise 2
84
85
86 # select the description column from reviews and assign it to variable desc
87
88 desc = reviews["description"]
89
90 desc
91
92 # %%
93
94
95 # check the type of the variable desc is
96
97
98 print(type(desc))
99
100
101 # %%
102
103
104 # select first column from description column of reviews and assign it to ariable first_description
105
106
107 first_description = reviews["description"][0]
108
109 first_description
110
111 # %%
112
113 # select first row of the data from reviews and assign it to first_row
114
115 first_row = reviews.iloc[0]
```

```
115 first_row = reviews.iloc[0]
116
117 first_row
118
119
120 # %%
121
122
123 # select the first 10 values from the description column in reviews and assign to variable first_descriptions
124
125 first_descriptions = reviews["description"][:10]
126
127 first_descriptions
128
129
130 # %%
131
132
133 # select the records with index labels 1,2,3,5,8 and assign to variable sample_reviews
134
135 sample_reviews = reviews.iloc[[1, 2, 3, 5, 8], :]
136
137
138 sample_reviews
139
140
141 # %%
142
143
144 # create a variable df have country, province, region_1, region_2, columns with index labels 0,1,10,100
145
146 df = reviews[["description", "country", "province",
147               "region_1", "region_2"]].iloc[[0, 1, 10, 100]]
148
149 df
150
151
152 # %%
153
```

```
154
155 # a variable df containing country and variety column first 100 records
156
157 df = reviews[["country", "variety"]].iloc[:100, ]
158
159 df
160
161
162 # %%
163
164
165 # craete a dataframe italian_wines having reviews of wines made in Italy
166
167 italian_wines = reviews.loc[reviews["country"] == "Italy"]
168
169
170 italian_wines
171
172
173 # %%
174
175
176 # create a datafrme top_oceania_wines with all reviews having atleast 95 points out of 100 for wines from Australia and New Zealand
177
178
179 top_oceania_wines = reviews.loc[(reviews["country"].isin(
180     ["Australia", "New Zealand"])) & (reviews["points"] >= 95)]
181
182
183 top_oceania_wines
184
185
186 # %%
187
188
189 # excercse 3
190
191
```

```
192 # find the median of the points column
193
194
195 median_points = reviews["points"].median()
196
197
198 median_points
199
200 # %%
201
202
203 # what countries in dataset
204
205 countries_unique = reviews["country"].unique()
206
207
208 countries_unique
209
210
211 # %%
212
213
214 # create reviews_per_country variable mapping countries to the count of reviews of wines from that country
215
216
217 reviews_per_country = reviews["country"].value_counts()
218
219
220 reviews_per_country
221
222
223 # %%
224
225
226 # create centered_price having version of price column with mean value subtracted
227
228
229 centered_price = reviews["price"] - reviews["price"].mean()
230
```

```
231
232 centered_price
233
234
235 # %%
236
237
238 # create variable bargain_wine with the title of the wine with the highest points-to-price ratio
239
240
241 bargain_idx = (reviews["points"]/reviews["price"]).idxmax()
242
243 bargain_wine = reviews.loc[bargain_idx]["winery"]
244
245 bargain_wine
246
247
248 # %%
249
250
251 # create a series descriptor_counts to count how many the words "tropical" or "fruity" appear
252
253
254 trop = reviews["description"].map(lambda desc: "tropical" in desc).sum()
255
256
257 fruit = reviews["description"].map(lambda desc: "fruity" in desc).sum()
258
259
260 descriptor_counts = pd.Series([trop, fruit], index=["tropical", "fruity"])
261
262
263 descriptor_counts
264
265
266 # %%
267
268
```

```
# create variable star_ratings with number of strs regarding the eachreview in dataset
```

```
def rating(row):
```

```
    if row.country == "Canada":
```

```
        return 3
```

```
    elif row.points >= 95:
```

```
        return 3
```

```
    elif row.points >= 85:
```

```
        return 2
```

```
    else:
```

```
        return 1
```

```
star_ratings = reviews.apply(rating, axis="columns")
```

```
star_ratings
```

```
# %%
```

```
# excercise 4
```

```
# create series whose index is variety and whose value counts how many review each person wrote
```

```
reviews_written = reviews.groupby('variety').variety.count()
```

```
reviews_written
```

```
# %%
```



```

308
309 # Create a Series whose index is wine prices and whose values is the maximum number of points a wine costing that much was given in a review
310
311 best_rating_per_price = reviews.groupby('price')['points'].max().sort_index()
312
313 best_rating_per_price
314
315
316 # %%
317
318 # Create a DataFrame whose index is the variety category from the dataset and whose values are the min and max values thereof.
319
320 price_extremes = reviews.groupby('variety').price.agg([min, max])
321
322
323 price_extremes
324
325 # %%
326
327 # Create a variable sorted_varieties containing a copy of the dataframe from the previous question where varieties are sorted in descending order
328
329 sorted_varieties = price_extremes.sort_values(
330     by=['min', 'max'], ascending=False)
331
332
333 sorted_varieties
334
335
336 # %%
337
338 # Create a Series whose index is reviewers and whose values is the average review score given out by that reviewer.
339
340 reviewer_mean_ratings = reviews.groupby('designation').points.mean()
341
342
343 reviewer_mean_ratings
344
345

```

```
346 # %%
347
348 # Create a Series whose index is a MultiIndex of {country, variety} pairs.
349
350
351 country_variety_counts = reviews.groupby(
352     ['country', 'variety']).size().sort_values(ascending=False)
353
354
355 country_variety_counts
356
357
358 # %%
359
360 # exercise 5
361
362
363 # What is the data type of the points column in the dataset
364
365
366 dtype = reviews.points.dtype
367
368
369 dtype
370
371
372 # %%
373
374
375 # Create a Series from entries in the `points` column, but convert the entries to strings.
376
377
378 point_strings = reviews.points.astype(str)
379
380
381 point_strings
382
383
384 # %%
```

```
384 # %%
385
386
387 # How many reviews in the dataset are missing a price?
388
389
390 n_missing_prices = reviews.price.isnull().sum()
391
392
393 n_missing_prices
394
395
396 # %%
397
398
399 # Create a Series counting the number of times each value occurs in the region_1 field. This field is often missing data, so replace missing values with 'Unknown'.
400
401
402 reviews_per_region = reviews.region_1.fillna(
403     'Unknown').value_counts().sort_values(ascending=False)
404
405
406 reviews_per_region
407
408
409 # %%
410
411 # `region_1` and `region_2` are pretty uninformative names for locale columns in the dataset. Create a copy of `reviews` with these columns renamed to `region` and `locale`.
412
413 renamed = reviews.rename(columns=dict(region_1='region', region_2='locale'))
414
415
416 renamed
417
418
419 # %%
420
421 # Set the index name in the dataset to `wines`.
422
```

```
423  
424 reindexed = reviews.rename_axis('wines', axis='rows')
```

```
425  
426  
427 reindexed
```

```
428  
429 # %%
```

```
430  
431 # =====
```

```
432 # NUMPY EXCERCISES
```

```
433 # =====
```

```
434  
435  
436 # %%
```

```
437  
438 # NumPy Creating Array
```

```
439  
440 # importing the numpy library
```

```
441  
442  
443 # %%
```

```
444  
445 # Create a NumPy ndarray Object
```

```
446  
447  
448 arr = np.array([1, 2, 3, 4, 5])
```

```
449  
450 print(arr)
```

```
451  
452 print(type(arr))
```

```
453  
454  
455 # %%
```

```
456  
457 # Use a tuple to create a NumPy array:
```

```
458  
459  
460 arr = np.array((1, 2, 3, 4, 5))
```

```
462 print(arr)
463
464
465 # %%
466
467 # Create a 0-D array with value 42
468
469
470 arr = np.array(42)
471
472 print(arr)
473
474
475 # %%
476
477 # Create a 1-D array containing the values 1,2,3,4,5:
478
479
480 arr = np.array([1, 2, 3, 4, 5])
481
482 print(arr)
483
484
485 # %%
486
487 # Create a 2-D array containing two arrays with the values 1,2,3 and 4,5,6:
488
489
490 arr = np.array([[1, 2, 3], [4, 5, 6]])
491
492 print(arr)
493
494
495 # %%
496
497 # Create a 3-D array with two 2-D arrays, both containing two arrays with the values 1,2,3 and 4,5,6:
498
499
```

```
500 arr = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])
501
502 print(arr)
503
504
505 # %%
506
507
508 # Check how many dimensions the arrays have:
509
510
511 a = np.array(42)
512 b = np.array([1, 2, 3, 4, 5])
513 c = np.array([[1, 2, 3], [4, 5, 6]])
514 d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])
515
516 print(a.ndim)
517 print(b.ndim)
518 print(c.ndim)
519 print(d.ndim)
520
521
522 # %%
523
524 # Create an array with 5 dimensions and verify that it has 5 dimensions:
525
526
527 arr = np.array([1, 2, 3, 4], ndmin=5)
528
529 print(arr)
530 print('number of dimensions :', arr.ndim)
531
532
533 # %%
534
535 # NumPy Array Indexing
536
537
```

```
# Get the first element from the following array:
```

```
arr = np.array([1, 2, 3, 4])
```

```
print(arr[0])
```

```
# %%
```

```
# Get the second element from the following array.
```

```
arr = np.array([1, 2, 3, 4])
```

```
print(arr[1])
```

```
# %%
```

```
# Get third and fourth elements from the following array and add them.
```

```
arr = np.array([1, 2, 3, 4])
```

```
print(arr[2] + arr[3])
```

```
# %%
```

```
# Access the 2nd element on 1st dim:
```

```
arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
```

```
print('2nd element on 1st dim: ', arr[0, 1])
```

```
# %%
```

```
577 # Access the 5th element on 2nd dim:
578
579
580 arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
581
582 print('5th element on 2nd dim: ', arr[1, 4])
583
584 # %%
585
586 # Access the third element of the second array of the first array:
587
588
589 arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
590
591 print(arr[0, 1, 2])
592
593
594 # %%
595
596 # Print the last element from the 2nd dim:
597
598
599 arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
600
601 print('Last element from 2nd dim: ', arr[1, -1])
602
603
604 # %%
605
606
607 # NumPy Array Slicing
608
609
610 # Slice elements from index 1 to index 5 from the following array:
611
612
613 arr = np.array([1, 2, 3, 4, 5, 6, 7])
614
```



```
615 print(arr[1:5])
616
617
618 # %%
619
620 # Slice elements from index 4 to the end of the array:
621
622
623 arr = np.array([1, 2, 3, 4, 5, 6, 7])
624
625 print(arr[4:])
626
627 # %%
628
629 # Slice elements from the beginning to index 4 (not included):
630
631
632 arr = np.array([1, 2, 3, 4, 5, 6, 7])
633
634 print(arr[:4])
635
636
637 # %%
638
639
640 # Slice from the index 3 from the end to index 1 from the end:
641
642
643 arr = np.array([1, 2, 3, 4, 5, 6, 7])
644
645 print(arr[-3:-1])
646
647
648 # %%
649
650
651 # Return every other element from index 1 to index 5:
652
653
```

```
654 arr = np.array([1, 2, 3, 4, 5, 6, 7])
655
656 print(arr[1:5:2])
657
658
659 # %%
660
661
662 # Return every other element from the entire array:
663
664
665 arr = np.array([1, 2, 3, 4, 5, 6, 7])
666
667 print(arr[::-2])
668
669
670 # %%
671
672 # From the second element, slice elements from index 1 to index 4 (not included):
673
674
675 arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
676
677 print(arr[1, 1:4])
678
679 # %%
680
681 # From both elements, return index 2:
682
683
684 arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
685
686 print(arr[0:2, 2])
687
688
689 # %%
690
691 # From both elements, slice index 1 to index 4 (not included), this will return a 2-D array:
```

```
692
693
694 arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
695
```

```
696 print(arr[0:2, 1:4])
697
698
```

```
699 # %%
700
```

```
701 # NumPy Data Types
702
703
```

```
704 # Get the data type of an array object:
705
706
```

```
707 arr = np.array([1, 2, 3, 4])
708
```

```
709 print(arr.dtype)
710
```

```
711
712 # %%
713
```

```
714 # Get the data type of an array containing strings:
715
716
```

```
717 arr = np.array(['apple', 'banana', 'cherry'])
718
```

```
719 print(arr.dtype)
720
```

```
721
722 # %%
723
```

```
724 # Create an array with data type string:
725
726
```

```
727 print(arr)
728
```

```
728 print(arr.dtype)
729
730
```

```
731 # %%
732
733 # Create an array with data type 4 bytes integer:
734
735
736 arr = np.array([1, 2, 3, 4], dtype='i4')
737
738 print(arr)
739 print(arr.dtype)
740
741
742 # %%
743
744 # Change data type from float to integer by using 'i' as parameter value:
745
746
747 arr = np.array([1.1, 2.1, 3.1])
748
749 newarr = arr.astype('i')
750
751 print(newarr)
752 print(newarr.dtype)
753
754 # %%
755
756 # Change data type from float to integer by using int as parameter value:
757
758
759 arr = np.array([1.1, 2.1, 3.1])
760
761 newarr = arr.astype(int)
762
763 print(newarr)
764 print(newarr.dtype)
765
766
767 # %%
768
```

```
769 # Change data type from integer to boolean:
770
771
772 arr = np.array([1, 0, 3])
773
774 newarr = arr.astype(bool)
775
776 print(newarr)
777 print(newarr.dtype)
778
779
780 # %%
781
782 # NumPy Array Copy vs View
783
784
785 # Make a copy, change the original array, and display both arrays:
786
787
788 arr = np.array([1, 2, 3, 4, 5])
789 x = arr.copy()
790 arr[0] = 42
791
792 print(arr)
793 print(x)
794
795
796 # %%
797
798 # Make a view, change the original array, and display both arrays:
799
800
801 arr = np.array([1, 2, 3, 4, 5])
802 x = arr.view()
803 arr[0] = 42
804
805 print(arr)
806 print(x)
```

```
808
809 # %%
810
811 # Make a view, change the view, and display both arrays:
812
813
814 arr = np.array([1, 2, 3, 4, 5])
815 x = arr.view()
816 x[0] = 31
817
818 print(arr)
819 print(x)
820
821
822 # %%
823
824 # Print the value of the base attribute to check if an array owns it's data or not:
825
826
827 arr = np.array([1, 2, 3, 4, 5])
828
829 x = arr.copy()
830 y = arr.view()
831
832 print(x.base)
833 print(y.base)
834
835
836 # %%
837
838 # NumPy Array Shape
839
840
841 # Print the shape of a 2-D array:
842
843
844 arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
845
```

```
846 print(arr.shape)
847
848
849 # %%
850
851 # Create an array with 5 dimensions using ndmin using a vector with values 1,2,3,4 and verify that last dimension has value 4:
852
853
854 arr = np.array([1, 2, 3, 4], ndmin=5)
855
856 print(arr)
857 print('shape of array :', arr.shape)
858
859
860 # %%
861
862 # NumPy Array Reshaping
863
864
865 # Convert the following 1-D array with 12 elements into a 2-D array.The outermost dimension will have 4 arrays, each with 3 elements:
866
867
868 arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
869
870 newarr = arr.reshape(4, 3)
871
872 print(newarr)
873
874
875 # %%
876
877
878 # Convert the following 1-D array with 12 elements into a 3-D array.The outermost dimension will have 2 arrays that contains 3 arrays, each with
879
880 arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
881
882 newarr = arr.reshape(2, 3, 2)
883
```

```
884 print(newarr)
885
886
887 # %%
888
889 # Check if the returned array is a copy or a view:
890
891
892 arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])
893
894 print(arr.reshape(2, 4).base)
895
896
897 # %%
898
899
900 # Convert 1D array with 8 elements to 3D array with 2x2 elements:
901
902
903 arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])
904
905 newarr = arr.reshape(2, 2, -1)
906
907 print(newarr)
908
909
910 # %%
911
912 # Convert the array into a 1D array:
913
914
915 arr = np.array([[1, 2, 3], [4, 5, 6]])
916
917 newarr = arr.reshape(-1)
918
919 print(newarr)
920
921
922 # %%
```



```
# %%  
923  
924 # NumPy Array Iterating  
925  
926  
927 # Iterate on the elements of the following 1-D array:  
928  
929  
930 arr = np.array([1, 2, 3])  
931  
932 for x in arr:  
933     print(x)  
934  
935  
936 # %%  
937  
938  
939 # Iterate on the elements of the following 2-D array:  
940  
941  
942 arr = np.array([[1, 2, 3], [4, 5, 6]])  
943  
944 for x in arr:  
945     print(x)  
946  
947  
948 # %%  
949  
950 # Iterate on each scalar element of the 2-D array:  
951  
952  
953 arr = np.array([[1, 2, 3], [4, 5, 6]])  
954  
955 for x in arr:  
956     for y in x:  
957         print(y)  
958  
959  
960 # %%
```

```
961 # Iterate on the elements of the following 3-D array:
962
963
964
965 arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
966
967 for x in arr:
968     print(x)
969
970
971 # %%
972
973 # Iterate down to the scalars:
974
975
976 arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
977
978 for x in arr:
979     for y in x:
980         for z in y:
981             print(z)
982
983
984 # %%
985
986 # Iterate through the following 3-D array:
987
988
989 arr = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])
990
991 for x in np.nditer(arr):
992     print(x)
993
994
995 # %%
996
997
998 # Iterate through the array as a string:
999
```

```
1000
1001 arr = np.array([1, 2, 3])
```

```
1002
1003 for x in np.nditer(arr, flags=['buffered'], op_dtypes=['S']):
1004     print(x)
```

```
1005
1006
1007 # %%
```

```
1008
1009 # Iterate through every scalar element of the 2D array skipping 1 element:
```

```
1010
1011
1012 arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
```

```
1013
1014 for x in np.nditer(arr[:, ::2]):
1015     print(x)
```

```
1016
1017
1018 # %%
```

```
1019
1020 # Enumerate on following 1D arrays elements:
```

```
1021
1022
1023 arr = np.array([1, 2, 3])
```

```
1024
1025 for idx, x in np.ndenumerate(arr):
1026     print(idx, x)
```

```
1027
1028
1029 # %%
```

```
1030
1031 # Enumerate on following 2D array's elements:
```

```
1032
1033
1034 arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
```

```
1035
1036 for idx, x in np.ndenumerate(arr):
1037     print(idx, x)
```

```
1038 # %%
1039
1040 # NumPy Joining Array
1041
1042 # Join two arrays
1043
1044
1045 arr1 = np.array([1, 2, 3])
1046
1047 arr2 = np.array([4, 5, 6])
1048
1049 arr = np.concatenate((arr1, arr2))
1050
1051 print(arr)
1052
1053 # %%
1054
1055 # Join two 2-D arrays along rows (axis=1):
1056
1057
1058 arr1 = np.array([[1, 2], [3, 4]])
1059
1060 arr2 = np.array([[5, 6], [7, 8]])
1061
1062 arr = np.concatenate((arr1, arr2), axis=1)
1063
1064 print(arr)
1065
1066 # %%
1067
1068 # NumPy Splitting Array
```

```
1077 # Split the array in 3 parts:
1078
1079
1080 arr = np.array([1, 2, 3, 4, 5, 6])
1081
1082 newarr = np.array_split(arr, 3)
1083
1084 print(newarr)
1085
1086
1087 # %%
1088
1089 # Split the array in 4 parts:
1090
1091
1092 arr = np.array([1, 2, 3, 4, 5, 6])
1093
1094 newarr = np.array_split(arr, 4)
1095
1096 print(newarr)
1097
1098
1099 # %%
1100
1101 # Access the splitted arrays:
1102
1103
1104 arr = np.array([1, 2, 3, 4, 5, 6])
1105
1106 newarr = np.array_split(arr, 3)
1107
1108 print(newarr[0])
1109 print(newarr[1])
1110 print(newarr[2])
1111
1112
1113 # %%
1114
```

```
1115 # Split the 2-D array into three 2-D arrays.
1116
1117
1118 arr = np.array([[1, 2], [3, 4], [5, 6], [7, 8], [9, 10], [11, 12]])
1119
1120 newarr = np.array_split(arr, 3)
1121
1122 print(newarr)
1123
1124
1125 # %%
1126
1127 # Split the 2-D array into three 2-D arrays.
1128
1129
1130 arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [
1131     10, 11, 12], [13, 14, 15], [16, 17, 18]])
1132
1133 newarr = np.array_split(arr, 3)
1134
1135 print(newarr)
1136
1137
1138 # %%
1139
1140 # Split the 2-D array into three 2-D arrays along rows.
1141
1142
1143 arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [
1144     10, 11, 12], [13, 14, 15], [16, 17, 18]])
1145
1146 newarr = np.array_split(arr, 3, axis=1)
1147
1148 print(newarr)
1149
1150
1151 # %%
1152
```

```
# NumPy Searching Arrays
```

```
# Find the indexes where the value is 4:
```

```
arr = np.array([1, 2, 3, 4, 5, 4, 4])
```

```
x = np.where(arr == 4)
```

```
print(x)
```

```
# %%
```

```
# Find the indexes where the values are even:
```

```
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])
```

```
x = np.where(arr % 2 == 0)
```

```
print(x)
```

```
# %%
```

```
# Find the indexes where the values are odd:
```

```
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])
```

```
x = np.where(arr % 2 == 1)
```

```
print(x)
```

```
# %%
```

```
# NumPy Sorting Arrays
```

```
1192 # Sort the array:
1193
1194 arr = np.array([3, 2, 0, 1])
1195
1196 print(np.sort(arr))
1197
1198
1199 # %%
1200
1201 # Sort the array alphabetically:
1202
1203 arr = np.array(['banana', 'cherry', 'apple'])
1204
1205 print(np.sort(arr))
1206
1207
1208 # %%
1209
1210 # Sort a boolean array:
1211
1212 arr = np.array([True, False, True])
1213
1214 print(np.sort(arr))
1215
1216
1217 # %%
1218
1219 # Sort a 2-D array:
1220
1221 arr = np.array([[3, 2, 4], [5, 0, 1]])
1222
1223 print(np.sort(arr))
1224
1225
1226 # %%
1227
1228
1229 # =====
```


1230

EXPERIMENT COMPLETED

1231

=====