In [1]:
```
# import relevant modules
import pandas as pd
```

In [2]:
```
# read grades dataset, save as a pandas dataframe
grades = pd.read_csv('./grades.csv')
```

In [3]:
```
# display first few rows of grades
grades.head()
```

Out[3]:

	exam	student_id	grade
0	1	1	86.0
1	1	2	65.0
2	1	3	70.0
3	1	4	98.0
4	1	5	89.0

In [4]:
```
def lowest_grade(student_id):

    """Find lowest grade across all exams for student with given student_id.
    Treat missing exam grades as zeros."""

    return grades.loc[grades['student_id'] == student_id]['grade'].fillna(0).min()
```

In [5]:
```
# test lowest_grade on student_id 1
assert lowest_grade(1) == 0.0, 'test failed'
print('test passed')
```

```
test passed
```

In [6]:
```
# sequence containing all distinct student ids
student_ids = grades['student_id'].unique()
student_ids
```

Out[6]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10], dtype=int64)
```

In [7]:
```
# apply lowest_grade to each student id
list(map(lowest_grade, student_ids))
```

Out[7]:
```
[0.0, 0.0, 70.0, 0.0, 0.0, 0.0, 75.0, 56.0, 73.0, 75.0]
```

Zip

In [8]:
```
# points scored by each of five players in game1
points_game1 = [50, 40, 60, 70, 80]
```

In [9]:
```
# points scored by each of five players in game2
points_game2 = [76, 81, 53, 92, 67]
```

In [10]:
```
# sequence where points differences will be saved
diffs = []

# iterate through points_game1 and points_game2 at the same time
for x, y in zip(points_game1, points_game2):
    # compute absolute difference in points between game1 and game2
    # add to diffs
    diffs.append(abs(x - y))

diffs
```

Out[10]:
```
[26, 41, 7, 22, 13]
```

Filter

In [11]:
```
def mean_atleast_70(student_id):

    """Compute mean grade across all exams for student with given student_id.
    Treat missing exam grades as zeros.
    If mean grade is atleast 70, return True. Otherwise, return False."""

    mean_grade = grades.loc[grades['student_id'] == student_id]['grade'].fillna(0).mean
    return mean_grade >= 70
```

In [12]:
```
# test mean_grade on student_id 1
assert mean_atleast_70(1) == False, 'test failed'
print('test passed')
```

```
test passed
```

In [13]:
```
# sequence containing all distinct student ids
student_ids = grades['student_id'].unique()
student_ids
```

Out[13]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10], dtype=int64)
```

In [14]:
```
list(filter(mean_atleast_70, student_ids))
```

Out[14]:
```
[3, 5, 7, 8, 9, 10]
```

Numpy

In [16]:
```
# import relevant libraries
import numpy as np
```

In [17]:
```
# create an empty numpy array & save in a variable
array0 = np.array([])
array0
```

Out[17]:
```
array([], dtype=float64)
```

In [18]:
```
# initialize list1 as a Python list
list1 = [1, 2, 3, 4, 5]
```

In [19]:
```
# create a one-dimensional numpy array from list1 & save in a variable
array1 = np.array(list1)
array1
```

Out[19]:
```
array([1, 2, 3, 4, 5])
```

In [20]:
```
# initialize list2 as a nested Python list
list2 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

In [21]:
```
# create a two-dimensional numpy array from list2
array2 = np.array(list2)
array2
```

Out[21]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

In [22]:
```
# create a numpy array containing integers 0 to 4 including 0 and 4
np.arange(5)
```

Out[22]:
```
array([0, 1, 2, 3, 4])
```

In [23]:
```
# create a numpy array containing all even integers between 0 and 10 including 0 and 10
np.arange(0, 11, 2)
```

Out[23]:
```
array([ 0,  2,  4,  6,  8, 10])
```

In [24]:
```
# create a one-dimenstional numpy array containing 5 zeros
np.zeros(5)
```

Out[24]:
```
array([0., 0., 0., 0., 0.])
```

In [25]:
```
# create a two-dimensional numpy array of zeros having 4 rows and 5 columns
np.zeros((4, 5))
```

Out[25]:
```
array([[0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.]])
```

In [26]:
```
# create a one-dimenstional numpy array containing 6 ones
np.ones(6)
```

Out[26]:
```
array([1., 1., 1., 1., 1., 1.])
```

In [27]:
```
# create a two-dimensional numpy array of ones having 4 rows and 6 columns
np.ones((4, 6))
```

Out[27]:
```
array([[1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1.]])
```

In [28]:
```
# create a numpy array of 9 evenly spaced numbers from 1 to 2, including 1 and 2
np.linspace(1, 2, 9)
```

Out[28]:
```
array([1.   , 1.125, 1.25 , 1.375, 1.5  , 1.625, 1.75 , 1.875, 2.   ])
```

In [29]:
```
# create a numpy array of 10 random integers from 20 to 50
np.random.randint(20, 50, 10)
```

Out[29]:
```
array([42, 44, 48, 37, 38, 43, 43, 33, 42, 38])
```

Min_max

In [30]:
```
# import relevant libraries
import numpy as np
```

In [31]:
```
# create a numpy array of 20 random integers from 1 to 50 & save in a variable
array_random = np.random.randint(1, 50, 20)
array_random
```

Out[31]:
```
array([11, 20, 48, 46, 41, 32, 21, 38, 10,  7, 49,  6,  9, 18, 22, 24,  4,
        4, 14, 10])
```

In [32]:
```
# find minimum value in array_random
array_random.min()
```

Out[32]:
```
4
```

In [33]:
```
# find maximum value in array_random
array_random.max()
```

Out[33]:
```
49
```

Indices of min_max

In [34]:
```
# import relevant libraries
```

```
import numpy as np
```

In [35]:
```
# create a numpy array of 20 random integers from 1 to 50 & save in a variable
array_random = np.random.randint(1, 50, 20)
array_random
```

Out[35]:
```
array([45, 49,  5, 24, 39,  9, 45, 22, 36, 15, 46,  8, 25, 15, 36, 48, 12,
       20, 30,  9])
```

In [36]:
```
# find index of minimum value in array_random
array_random.argmin()
```

Out[36]:
```
2
```

In [37]:
```
# find index of maximum value in array_random
array_random.argmax()
```

Out[37]:
```
1
```

Shapes

In [38]:
```
# import relevant libraries
import numpy as np
```

In [39]:
```
array0 = np.array([])
```

In [40]:
```
# find the shape of array0
array0.shape
```

Out[40]:
```
(0,)
```

In [41]:
```
# find the shape of array0
array0.shape
```

Out[41]:
```
(0,)
```

In [42]:
```
array1 = np.array([1, 2, 3, 4, 5])
```

In [43]:
```
# find the shape of array1
array1.shape
```

Out[43]:
```
(5,)
```

In [44]:
```
array2 = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
```

In [45]:
```
# find the shape of array2
array2.shape
```

Out[45]:
```
(3, 3)
```

In [46]:
```
array_zeros = np.zeros((4, 5))
```

In [47]:
```
# find the shape of array_zeros
array_zeros.shape
```

Out[47]:
```
(4, 5)
```

In [48]:
```
array_ones = np.ones((3, 6))
```

In [49]:
```
# find the shape of array_ones
array_ones.shape
```

Out[49]:
```
(3, 6)
```

In [50]:
```
array_r0 = np.random.randint(1, 50, 25)
array_r0
```

Out[50]:
```
array([10, 44, 14, 24, 10, 38, 49,  1, 10, 14, 10, 33, 30, 48, 20, 19, 32,
       30,  3, 19, 26, 21,  2, 40,  8])
```

In [51]:
```
# find the shape of array_r0
array_r0.shape
```

Out[51]:
```
(25,)
```

In [52]:
```
# reshape array_r0
# to create a new multi-dimensional numpy array
# containing array_r0's items with 5 rows & 5 columns
array_r0.reshape((5, 5))
```

Out[52]:
```
array([[10, 44, 14, 24, 10],
       [38, 49,  1, 10, 14],
       [10, 33, 30, 48, 20],
       [19, 32, 30,  3, 19],
       [26, 21,  2, 40,  8]])
```

In [53]:
```
# note that array_r0 itself was not modified
array_r0
```

Out[53]:
```
array([10, 44, 14, 24, 10, 38, 49,  1, 10, 14, 10, 33, 30, 48, 20, 19, 32,
       30,  3, 19, 26, 21,  2, 40,  8])
```

In [54]:
```
array_r1 = np.random.randint(1, 50, 20)
array_r1
```

Out[54]:
```
array([17, 35, 39, 27,  9, 15, 19, 20, 43, 14, 46, 18, 17, 36, 41, 33,  5,
        2, 25, 11])
```

In [55]:
```
# find the shape of array_r1
array_r1.shape
```

Out[55]:
```
(20,)
```

In [56]:
```
# reshape array_r1
# to create a new multi-dimensional numpy array containing array_r1's items
# with 10 rows & 2 columns
array_r1.reshape((10, 2))
```

Out[56]:
```
array([[17, 35],
       [39, 27],
       [ 9, 15],
       [19, 20],
       [43, 14],
       [46, 18],
       [17, 36],
       [41, 33],
       [ 5,  2],
       [25, 11]])
```

In [57]:
```
# reshape array_r1
# to create a new multi-dimensional numpy array containing array_r1's items
# with 2 rows & 10 columns
array_r1.reshape((2, 10))
```

Out[57]:
```
array([[17, 35, 39, 27,  9, 15, 19, 20, 43, 14],
       [46, 18, 17, 36, 41, 33,  5,  2, 25, 11]])
```

In [58]:
```
# reshape array_r1
# to create a new multi-dimensional numpy array containing array_r1's items
# with 4 rows & 5 columns
array_r1.reshape((4, 5))
```

Out[58]:
```
array([[17, 35, 39, 27,  9],
       [15, 19, 20, 43, 14],
       [46, 18, 17, 36, 41],
       [33,  5,  2, 25, 11]])
```

In [59]:
```
# reshape array_r1
# to create a new multi-dimensional numpy array containing array_r1's items
# with 5 rows & 4 columns
array_r1.reshape((5, 4))
```

Out[59]:
```
array([[17, 35, 39, 27],
       [ 9, 15, 19, 20],
       [43, 14, 46, 18],
       [17, 36, 41, 33],
       [ 5,  2, 25, 11]])
```

In [60]:
```
# note that array_r1 itself was not modified
array_r1
```

Out[60]:
```
array([17, 35, 39, 27,  9, 15, 19, 20, 43, 14, 46, 18, 17, 36, 41, 33,  5,
        2, 25, 11])
```

In [61]:
```
array_r2 = np.random.randint(1, 50, (4, 4))
array_r2
```

Out[61]:
```
array([[10, 23, 34, 27],
       [14, 15, 33,  8],
       [31, 46, 36, 46],
       [ 3,  8,  2, 23]])
```

In [62]:
```
# find the shape of array_r2
array_r2.shape
```

Out[62]:
```
(4, 4)
```

In [63]:
```
# reshape array_r2
# to create a new one-dimensional numpy array containing array_r2's items
# with length 16
array_r2.reshape(16)
```

Out[63]:
```
array([10, 23, 34, 27, 14, 15, 33,  8, 31, 46, 36, 46,  3,  8,  2, 23])
```

In [64]:
```
# note that array_r2 itself was not modified
array_r2
```

Out[64]:
```
array([[10, 23, 34, 27],
       [14, 15, 33,  8],
       [31, 46, 36, 46],
       [ 3,  8,  2, 23]])
```

Groups numpy

In [65]:
```
# initialize array0
# as a one-dimensional numpy array containing integers 0 to 4 including 0 and 4
array0 = np.arange(5)
array0
```

Out[65]:
```
array([0, 1, 2, 3, 4])
```

In [66]:
```
# select the item at index 1 from array0
array0[1]
```

Out[66]:
```
1
```

In [67]:
```
# select the items at indices 1 to 3 inclusive from array0
array0[1:4]
```

Out[67]:
```
array([1, 2, 3])
```

In [68]:
```
# select the items at indices 0 to 3 inclusive from array0
array0[:4]
```

Out[68]:
```
array([0, 1, 2, 3])
```

In [69]:
```
# select the items starting at index 3 until the end of array0
array0[3:]
```

Out[69]:
```
array([3, 4])
```

In [70]:
```
# initialize array1 as a three-dimensional numpy array
array1 = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
array1
```

Out[70]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

In [71]:
```
# select the item in row 0 column 0 from array1 using double bracket notation
array1[0][0]
```

Out[71]:
```
1
```

In [72]:
```
# select the item in row 0 column 0 from array1 using single bracket notation
array1[0, 0]
```

Out[72]:
```
1
```

In [73]:
```
# select the item in row 1 column 2 from array1 using double bracket notation
array1[1][2]
```

Out[73]:
```
6
```

In [74]:
```
# select the item in row 1 column 2 from array1 using single bracket notation
array1[1, 2]
```

Out[74]:
```
6
```

In [75]:
```
# select first two items from row 0 of array1
array1[:1, :2]
```

Out[75]:
```
array([[1, 2]])
```

In [76]:
```
# select first two items from every row until row 1 inclusive from array1
array1[:2, :2]
```

Out[76]:
```
array([[1, 2],
       [4, 5]])
```

In [77]:
```
# select first two items from each row of array1
array1[:, :2]
```

Out[77]:
```
array([[1, 2],
       [4, 5],
       [7, 8]])
```

In [78]:
```
# select row 0 from array1
array1[0]
```

Out[78]:
```
array([1, 2, 3])
```

In [79]:
```
# select everything before row 2 from array1
array1[:2]
```

Out[79]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [80]:
```
# initialize array2
# as a one-dimensional numpy array containing all even integers between 0 and 20 inclus
array2 = np.arange(0, 21, 2)
array2
```

Out[80]:
```
array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18, 20])
```

In [81]:
```
# select the items from array2 that are greater than or equal to 12
array2[array2 >= 12]
```

Out[81]:
```
array([12, 14, 16, 18, 20])
```

In [82]:
```
# select the items from array2 that are greater than 7 and less than 13
array2[(array2 > 7) & (array2 < 13)]
```

Out[82]:
```
array([ 8, 10, 12])
```

Arithmetic

In [83]:
```
# initialize arrayA as a one-dimensional numpy array
# containing the odd integers between 1 and 20 inclusive
arrayA = np.arange(1, 21, 2)
arrayA
```

Out[83]:
```
array([ 1,  3,  5,  7,  9, 11, 13, 15, 17, 19])
```

In [84]:
```
# initialize arrayB as a one-dimensional numpy array
# containing the integers 1 to 10 inclusive
arrayB = np.arange(1, 11)
arrayB
```

Out[84]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

In [85]:
```
# add each element of arrayA to each corresponding element of arrayB,
# creating a new numpy array
arrayA + arrayB
```

Out[85]:
```
array([ 2,  5,  8, 11, 14, 17, 20, 23, 26, 29])
```

In [86]:
```
# subtract each element of arrayB from each corresponding element from arrayA,
# creating a new numpy array
arrayA - arrayB
```

Out[86]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [87]:
```
# multiply each element of arrayA by each corresponding element of arrayB,
# creating a new numpy array
arrayA * arrayB
```

Out[87]:
```
array([  1,   6,  15,  28,  45,  66,  91, 120, 153, 190])
```

In [88]:
```
# divide each element of arrayA by each corresponding element of arrayB,
# creating a new numpy array
arrayA / arrayB
```

Out[88]:
```
array([1.        , 1.5       , 1.66666667, 1.75      , 1.8       ,
       1.83333333, 1.85714286, 1.875     , 1.88888889, 1.9       ])
```

In [89]:
```
# initialize arrayC as a multi-dimensional numpy array with 2 rows and 3 columns
arrayC = np.array([[1, 2, 3], [4, 5, 6]])
arrayC
```

Out[89]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [90]:
```
# initialize arrayD as another multi-dimensional numpy array with 2 rows and 3 columns
arrayD = np.array([[7, 8, 9], [10, 11, 12]])
arrayD
```

Out[90]:
```
array([[ 7,  8,  9],
       [10, 11, 12]])
```

In [91]:
```
# add each element of arrayC to each corresponding element of arrayD,
# creating a new numpy array
arrayC + arrayD
```

Out[91]:
```
array([[ 8, 10, 12],
       [14, 16, 18]])
```

In [92]:
```
# subtract each element of arrayD from each corresponding element from arrayC,
# creating a new numpy array
arrayC - arrayD
```

Out[92]:
```
array([[-6, -6, -6],
```

```
       [-6, -6, -6]])
```

In [93]:
```
# multiply each element of arrayC by each corresponding element of arrayD,
# creating a new numpy array
arrayC * arrayD
```

Out[93]:
```
array([[ 7, 16, 27],
       [40, 55, 72]])
```

In [94]:
```
# divide each element of arrayC by each corresponding element of arrayD,
# creating a new numpy array
arrayC / arrayD
```

Out[94]:
```
array([[0.14285714, 0.25      , 0.33333333],
       [0.4       , 0.45454545, 0.5       ]])
```

Scalar Numpy

In [96]:
```
# initialize arrayA as a one-dimensional numpy array
# containing the even integers between 2 and 20 inclusive
arrayA = np.arange(2, 21, 2)
arrayA
```

Out[96]:
```
array([ 2,  4,  6,  8, 10, 12, 14, 16, 18, 20])
```

In [97]:
```
# add 3 to each element of arrayA, creating a new numpy array
arrayA + 3
```

Out[97]:
```
array([ 5,  7,  9, 11, 13, 15, 17, 19, 21, 23])
```

In [98]:
```
# subtract 4 from each element of arrayA, creating a new numpy array
arrayA - 4
```

Out[98]:
```
array([-2,  0,  2,  4,  6,  8, 10, 12, 14, 16])
```

In [99]:
```
# multiply each element of arrayA by 5, creating a new numpy array
arrayA * 5
```

Out[99]:
```
array([ 10,  20,  30,  40,  50,  60,  70,  80,  90, 100])
```

In [100…
```
# divide each element of arrayA by 2, creating a new numpy array
arrayA / 2
```

Out[100…
```
array([ 1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10.])
```

In [101…
```
# initialize arrayB as a multi-dimensional numpy array with 2 rows and 3 columns
arrayB = np.array([[1, 2, 3], [4, 5, 6]])
arrayB
```

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

Out[101…
```
       [4, 5, 6]])
```

In [102…
```
# add 3 to each element of arrayB, creating a new numpy array
arrayB + 3
```

Out[102…
```
array([[4, 5, 6],
       [7, 8, 9]])
```

In [103…
```
# multiply each element of arrayB by 5, creating a new numpy array
arrayB * 5
```

Out[103…
```
array([[ 5, 10, 15],
       [20, 25, 30]])
```

In [104…
```
# divide each element of arrayB by 2, creating a new numpy array
arrayB / 2
```

Out[104…
```
array([[0.5, 1. , 1.5],
       [2. , 2.5, 3. ]])
```

statistical

In [105…
```
# initialize arrayA as a numpy array
# representing the scores of participants in a competition
scores = np.random.randint(50, 101, 200)
```

In [106…
```
# compute the median of scores
np.median(scores)
```

Out[106…
```
73.0
```

In [107…
```
# compute the mean of scores
np.mean(scores)
```

Out[107…
```
74.105
```

In [108…
```
# compute the variance of scores
np.var(scores)
```

Out[108…
```
217.233975
```

In [109…
```
# compute the standard deviation of scores
np.std(scores)
```

Out[109…
```
14.738859352066562
```

Other functions

In [110…
```
# initialize arrayA as a one-dimensional numpy array
# containing the integers 0 to 5 inclusive
arrayA = np.arange(6)
arrayA
```

Out[110…
```
array([0, 1, 2, 3, 4, 5])
```

In [111…
```
# compute the square of each item in arrayA,
# creating a new numpy array
np.square(arrayA)
```

Out[111…
```
array([ 0,  1,  4,  9, 16, 25], dtype=int32)
```

In [112…
```
# initialize arrayB as the following one-dimensional numpy array
arrayB = np.array([36, 49, 64, 81, 100, 121, 144, 169, 196, 225])
```

In [113…
```
# compute the square root of each item in arrayB,
# creating a new numpy array
np.sqrt(arrayB)
```

Out[113…
```
array([ 6.,  7.,  8.,  9., 10., 11., 12., 13., 14., 15.])
```

In [114…
```
# compute the exponential of each item in arrayA,
# creating a new numpy array
np.exp(arrayA)
```

Out[114…
```
array([  1.        ,   2.71828183,   7.3890561 ,  20.08553692,
        54.59815003, 148.4131591 ])
```

In [115…
```
# initialize arrayC as a one-dimensional numpy array
# containing the integers 1 to 11 inclusive
arrayC = np.arange(1, 11)
arrayC
```

Out[115…
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

In [116…
```
# compute the natural logarithm of each item in arrayC,
# creating a new numpy array
np.log(arrayC)
```

Out[116…
```
array([0.        , 0.69314718, 1.09861229, 1.38629436, 1.60943791,
       1.79175947, 1.94591015, 2.07944154, 2.19722458, 2.30258509])
```

In [117…
```
# initialize arrayD as a one-dimensional numpy array
# representing some angles in radians
arrayD = np.array([0, np.pi/6, np.pi/4, np.pi/3, np.pi/2, 2*np.pi/3, 3*np.pi/4, 5*np.pi
```

In [118…
```
# compute the sine of each item in arrayD,
# creating a new numpy array
np.sin(arrayD)
```

Out[118…
```
array([0.        , 0.5       , 0.70710678, 0.8660254 , 1.        ,
       0.8660254 , 0.70710678, 0.5       , 1.        ])
```

In [119…
```
# initialize arrayE as a one-dimensional numpy array containing the integers -10 to 10
arrayE = np.arange(-10, 11)
arrayE
```

Out[119…
```
array([-10,  -9,  -8,  -7,  -6,  -5,  -4,  -3,  -2,  -1,   0,   1,   2,
         3,   4,   5,   6,   7,   8,   9,  10])
```

In [120…
```
# compute the absolute value of each item in arrayE,
# creating a new numpy array
np.abs(arrayE)
```

Out[120…
```
array([10,  9,  8,  7,  6,  5,  4,  3,  2,  1,  0,  1,  2,  3,  4,  5,  6,
        7,  8,  9, 10])
```

In [121…
```
# initialize arrayF as the following one-dimensional numpy array
arrayF = np.arange(21)
arrayF
```

Out[121…
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20])
```

In [122…
```
# compute the sum of all the items in arrayF
np.sum(arrayF)
```

Out[122…
```
210
```

In [123…
```
# initialize arrayG as the following multi-dimensional numpy array
arrayG = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
arrayG
```

Out[123…
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

In [124…
```
# compute the sum of all the items in arrayG
np.sum(arrayG)
```

Out[124…
```
45
```

In [125…
```
# compute the sum of the items in each column of arrayG,
# creating a new numpy array containing the sum for each column
np.sum(arrayG, axis=0)
```

Out[125…
```
array([12, 15, 18])
```

In [126…
```
# compute the sum of the items in each row of arrayG,
# creating a new numpy array containing the sum for each row
np.sum(arrayG, axis=1)
```

Out[126… array([6, 15, 24])

Chapter 4

Linear algebra

In [127…
```
# import relevant libraries and modules
import numpy as np
from scipy import linalg
```

In [128…
```
# create a numpy array representing a 2 by 2 matrix & save in variable
matrix2by2 = np.array([[1, 2],
                       [3, 4]])
matrix2by2
```

Out[128…
```
array([[1, 2],
       [3, 4]])
```

In [129…
```
# compute determinant of matrix2by2
linalg.det(matrix2by2)
```

Out[129… -2.0

In [130…
```
# compute inverse of matrix2by2 & save in variable
inv_matrix2by2 = linalg.inv(matrix2by2)
inv_matrix2by2
```

Out[130…
```
array([[-2. ,  1. ],
       [ 1.5, -0.5]])
```

statistical

In [131…
```
# import relevant libraries and modules
import numpy as np
from scipy import stats
```

In [132…
```
# declare B to be a binomial discrete random variable with parameters 10 and 0.4
B = stats.binom(10, 0.4)
```

In [133…
```
# compute value of its probability mass function at 2
B.pmf(2)
```

Out[133… 0.12093235199999991

In [134…
```
# compute value of its cumulative density function at 3
B.cdf(3)
```

Out[134… 0.3822806015999999

In [135…
```
# declare P to be a poission discrete random variable with parameter 2
P = stats.poisson(2)
```

In [136…
```
# compute value of its probability mass function at 2
P.pmf(2)
```

Out[136… 0.2706705664732254

In [137…
```
# compute value of its cumulative density function at 4
P.cdf(4)
```

Out[137… 0.9473469826562889

In [138…
```
# declare G to be a geometric discrete random variable with parameter 0.25
G = stats.geom(0.25)
```

In [139…
```
# compute value of its probability mass function at 3
G.pmf(3)
```

Out[139… 0.140625

In [140…
```
# compute value of its cumulative density function at 4
G.cdf(4)
```

Out[140… 0.68359375

In [141…
```
# declare N to be a normal continuous random variable with parameters 0 and 1
N = stats.norm(0, 1)
```

In [142…
```
# compute its probability density at 0.1
N.pdf(0.1)
```

Out[142… 0.3969525474770118

In [143…
```
# compute its cumulative density at -0.2
N.cdf(-0.2)
```

Out[143… 0.42074029056089696

In [144…
```
# declare E to be an exponential continuous random variable with parameter 4
E = stats.expon(4)
```

In [145…
```
# declare X to be a beta continuous random variable with parameters 1 and 3
```

```
X = stats.beta(1, 3)
```

In [146…
```
# compute its probability density at 0.6
X.pdf(0.6)
```

Out[146…
```
0.4799999999999999
```

In [147…
```
# compute its cumulative density at 0.5
X.cdf(0.5)
```

Out[147…
```
0.875
```

In [148…
```
# create a variable named scores containing a numpy array
# that represents the scores of participants in a competition
scores = np.random.randint(50, 101, 300)
```

In [149…
```
# compute the 50th percentile of scores
stats.scoreatpercentile(scores, 50)
```

Out[149…
```
73.0
```

In [150…
```
# compute the 90th percentile of scores
stats.scoreatpercentile(scores, 90)
```

Out[150…
```
95.10000000000002
```

In [151…
```
# create a variable named round1_scores containing a numpy array
# that represents the scores of participants from school A in competition
schoolA_scores = np.random.normal(70, 15, size=100)
```

In [152…
```
# create a variable named round2_scores containing a numpy array
# that represents the scores of participants from school B in competition
schoolB_scores = np.random.normal(80, 15, size=10)
```

In [153…
```
# conduct the T-test for the means of schoolA_scores and schoolB_scores
stats.ttest_ind(schoolA_scores, schoolB_scores)
```

Out[153…
```
Ttest_indResult(statistic=-2.6210423934797293, pvalue=0.010029730310732704)
```

Chapter 5

In [154…
```
# create a pandas series containing 8 random integers from -10 to 10 inclusive
pd.Series(np.random.randint(-10, 11, size=8))
```

Out[154…
```
0    -3
1     4
```

```
2    10
3     8
4    -1
5    -9
6    -8
7     2
dtype: int32
```

In [155…
```
# create a pandas series containing 8 random integers from -10 to 10 inclusive
# with index being a through h
pd.Series(np.random.randint(-10, 11, size=8),
          index=['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'])
```

Out[155…
```
a    -1
b    -5
c     0
d     9
e     3
f     3
g     7
h   -10
dtype: int32
```

In [156…
```
# create a pandas series from a Python dictionary
pd.Series({'Mon': True, 'Tue': False, 'Wed': True, 'Thu': False,
           'Fri': True, 'Sat': False, 'Sun': True})
```

Out[156…
```
Mon     True
Tue    False
Wed     True
Thu    False
Fri     True
Sat    False
Sun     True
dtype: bool
```

In [157…
```
# create a pandas series from a scalar value
# to represent the maximum number of points students can earn
# in each of 10 exams for a particular course
pd.Series(150, index=np.arange(1, 11))
```

Out[157…
```
1     150
2     150
3     150
4     150
5     150
6     150
7     150
8     150
9     150
10    150
dtype: int64
```

In [158…
```
# read grades.csv into a pandas dataframe & save the dataframe in a variable
grades = pd.read_csv('grades.csv')
```

In [159…

```
# display grades
grades
```

Out[159…

	exam	student_id	grade
0	1	1	86.0
1	1	2	65.0
2	1	3	70.0
3	1	4	98.0
4	1	5	89.0
5	1	6	NaN
6	1	7	75.0
7	1	8	56.0
8	1	9	90.0
9	1	10	81.0
10	2	1	79.0
11	2	2	60.0
12	2	3	78.0
13	2	4	75.0
14	2	5	NaN
15	2	6	80.0
16	2	7	87.0
17	2	8	82.0
18	2	9	95.0
19	2	10	96.0
20	3	1	78.0
21	3	2	80.0
22	3	3	87.0
23	3	4	NaN
24	3	5	89.0
25	3	6	90.0
26	3	7	100.0
27	3	8	72.0
28	3	9	73.0
29	3	10	75.0
30	4	1	NaN
31	4	2	80.0

	exam	student_id	grade
32	4	3	81.0
33	4	4	82.0
34	4	5	83.0
35	4	6	84.0
36	4	7	85.0
37	4	8	86.0
38	4	9	87.0
39	4	10	88.0
40	5	1	90.0
41	5	2	NaN
42	5	3	91.0
43	5	4	92.0
44	5	5	93.0
45	5	6	94.0
46	5	7	95.0
47	5	8	96.0
48	5	9	97.0
49	5	10	98.0

In [160…

```
# display first few rows of grades
grades.head()
```

Out[160…

	exam	student_id	grade
0	1	1	86.0
1	1	2	65.0
2	1	3	70.0
3	1	4	98.0
4	1	5	89.0

In [161…

```
# create a Python dictionary of Series & save in a variable named points
points = {'player1': pd.Series([15, 10, 20, 25],
                               index=['game1', 'game2', 'game3', 'game4']),
          'player2': pd.Series([10, 15, 23, 27],
                               index=['game1', 'game2', 'game3', 'game4'])}

# create a pandas dataframe from points
pd.DataFrame(points)
```

Out[161...

	player1	player2
game1	15	10
game2	10	15
game3	20	23
game4	25	27

In [162...

```
# create a Python dictionary of lists & save in a variable named sales
sales = {'foodTruck1': [216,275,203,210,315,402,380],
         'foodTruck2': [374,90,95,115,130,150,140]}

# create a pandas dataframe from sales with index being day1 through day7
pd.DataFrame(sales, index=['day1', 'day2', 'day3', 'day4', 'day5', 'day6', 'day7'])
```

Out[162...

	foodTruck1	foodTruck2
day1	216	374
day2	275	90
day3	203	95
day4	210	115
day5	315	130
day6	402	150
day7	380	140

In [163...

```
# create a multi-dimensional numpy array & save in a variable named passengers
passengers = np.array([[20, 40, 60, 80], [15, 30, 45, 60], [10, 20, 30, 40]])

# create a pandas DataFrame from passengers
# with index being plane1 through plane3
# and columns being infants, children, adults, seniors
pd.DataFrame(passengers,
             index=['plane1', 'plane2', 'plane3'],
             columns=['infants', 'children', 'adults', 'seniors'])
```

Out[163...

	infants	children	adults	seniors
plane1	20	40	60	80
plane2	15	30	45	60
plane3	10	20	30	40

Select DataFrame

In [164...

```
# read grades.csv into a pandas dataframe & save the dataframe in a variable
grades = pd.read_csv('grades.csv')
```

In [165…
```
# display first few rows of grades
grades.head()
```

Out[165…

	exam	student_id	grade
0	1	1	86.0
1	1	2	65.0
2	1	3	70.0
3	1	4	98.0
4	1	5	89.0

In [166…
```
# select exam column and grade column from grades dataframe
# result will be a pandas dataframe containing just those two columns
grades.loc[:, ['exam', 'grade']]
```

Out[166…

	exam	grade
0	1	86.0
1	1	65.0
2	1	70.0
3	1	98.0
4	1	89.0
5	1	NaN
6	1	75.0
7	1	56.0
8	1	90.0
9	1	81.0
10	2	79.0
11	2	60.0
12	2	78.0
13	2	75.0
14	2	NaN
15	2	80.0
16	2	87.0
17	2	82.0
18	2	95.0
19	2	96.0
20	3	78.0
21	3	80.0

	exam	grade
22	3	87.0
23	3	NaN
24	3	89.0
25	3	90.0
26	3	100.0
27	3	72.0
28	3	73.0
29	3	75.0
30	4	NaN
31	4	80.0
32	4	81.0
33	4	82.0
34	4	83.0
35	4	84.0
36	4	85.0
37	4	86.0
38	4	87.0
39	4	88.0
40	5	90.0
41	5	NaN
42	5	91.0
43	5	92.0
44	5	93.0
45	5	94.0
46	5	95.0
47	5	96.0
48	5	97.0
49	5	98.0

In [167…

```
# select grade column from grades dataframe
# result will be a pandas dataframe containing just that column
grades.loc[:, ['grade']]
```

Out[167…

	grade
0	86.0

	grade
1	65.0
2	70.0
3	98.0
4	89.0
5	NaN
6	75.0
7	56.0
8	90.0
9	81.0
10	79.0
11	60.0
12	78.0
13	75.0
14	NaN
15	80.0
16	87.0
17	82.0
18	95.0
19	96.0
20	78.0
21	80.0
22	87.0
23	NaN
24	89.0
25	90.0
26	100.0
27	72.0
28	73.0
29	75.0
30	NaN
31	80.0
32	81.0
33	82.0
34	83.0

	grade
35	84.0
36	85.0
37	86.0
38	87.0
39	88.0
40	90.0
41	NaN
42	91.0
43	92.0
44	93.0
45	94.0
46	95.0
47	96.0
48	97.0
49	98.0

In [168…
```
# select row 0 from grades dataframe such that result is a pandas series
grades.iloc[0]
```

Out[168…
```
exam          1.0
student_id    1.0
grade        86.0
Name: 0, dtype: float64
```

In [169…
```
# select row 0 from grades dataframe such that result is a pandas dataframe
grades.iloc[[0], :]
```

Out[169…

	exam	student_id	grade
0	1	1	86.0

In [170…
```
# select row 0 and row 4 from grades dataframe
# result will be a pandas dataframe
grades.iloc[[0, 10], :]
```

Out[170…

	exam	student_id	grade
0	1	1	86.0
10	2	1	79.0

In [171…
```
# select item at row 4 column 2 from grades dataframe
```

```
grades.iloc[4, 2]
```

Out[171…

```
89.0
```

In [172…

```
# select column 0 and column 2 from grades dataframe
# result will be a pandas dataframe
grades.iloc[:, [0, 2]]
```

Out[172…

	exam	grade
0	1	86.0
1	1	65.0
2	1	70.0
3	1	98.0
4	1	89.0
5	1	NaN
6	1	75.0
7	1	56.0
8	1	90.0
9	1	81.0
10	2	79.0
11	2	60.0
12	2	78.0
13	2	75.0
14	2	NaN
15	2	80.0
16	2	87.0
17	2	82.0
18	2	95.0
19	2	96.0
20	3	78.0
21	3	80.0
22	3	87.0
23	3	NaN
24	3	89.0
25	3	90.0
26	3	100.0
27	3	72.0

	exam	grade
28	3	73.0
29	3	75.0
30	4	NaN
31	4	80.0
32	4	81.0
33	4	82.0
34	4	83.0
35	4	84.0
36	4	85.0
37	4	86.0
38	4	87.0
39	4	88.0
40	5	90.0
41	5	NaN
42	5	91.0
43	5	92.0
44	5	93.0
45	5	94.0
46	5	95.0
47	5	96.0
48	5	97.0
49	5	98.0

In [173…

```
# select row 0 and row 2 from grades dataframe
# result will be a pandas dataframe
grades.iloc[[0, 2], :]
```

Out[173…

	exam	student_id	grade
0	1	1	86.0
2	1	3	70.0

In [174…

```
# select rows 35 and 45 and columns 0 and 2 from grades dataframe
# result will be a pandas dataframe
grades.iloc[[35, 45], [0, 2]]
```

Out[174…

	exam	grade

	exam	grade
35	4	84.0
45	5	94.0

In [175…

```
# select every row from grades dataframe for which entry in grade column is atleast 70.
grades[grades['grade'] >= 70.0]
```

Out[175…

	exam	student_id	grade
0	1	1	86.0
2	1	3	70.0
3	1	4	98.0
4	1	5	89.0
6	1	7	75.0
8	1	9	90.0
9	1	10	81.0
10	2	1	79.0
12	2	3	78.0
13	2	4	75.0
15	2	6	80.0
16	2	7	87.0
17	2	8	82.0
18	2	9	95.0
19	2	10	96.0
20	3	1	78.0
21	3	2	80.0
22	3	3	87.0
24	3	5	89.0
25	3	6	90.0
26	3	7	100.0
27	3	8	72.0
28	3	9	73.0
29	3	10	75.0
31	4	2	80.0
32	4	3	81.0
33	4	4	82.0

	exam	student_id	grade
34	4	5	83.0
35	4	6	84.0
36	4	7	85.0
37	4	8	86.0
38	4	9	87.0
39	4	10	88.0
40	5	1	90.0
42	5	3	91.0
43	5	4	92.0
44	5	5	93.0
45	5	6	94.0
46	5	7	95.0
47	5	8	96.0
48	5	9	97.0
49	5	10	98.0

In [176…

```
# select all data representing students' grades on exam no. 5 that were atleast 70.0%
# in other words, select every row from grades dataframe for which
# entry in exam column is 5 and entry in grade column is atleast 70.0
grades[(grades['exam'] == 5) & (grades['grade'] >= 70.0)]
```

Out[176…

	exam	student_id	grade
40	5	1	90.0
42	5	3	91.0
43	5	4	92.0
44	5	5	93.0
45	5	6	94.0
46	5	7	95.0
47	5	8	96.0
48	5	9	97.0
49	5	10	98.0

Modifying pandas objects

In [177…

```
# read grades.csv into a pandas dataframe & save the dataframe in a variable
grades = pd.read_csv('grades.csv')
```

In [178…

```
# display grades
grades
```

Out[178…

	exam	student_id	grade
0	1	1	86.0
1	1	2	65.0
2	1	3	70.0
3	1	4	98.0
4	1	5	89.0
5	1	6	NaN
6	1	7	75.0
7	1	8	56.0
8	1	9	90.0
9	1	10	81.0
10	2	1	79.0
11	2	2	60.0
12	2	3	78.0
13	2	4	75.0
14	2	5	NaN
15	2	6	80.0
16	2	7	87.0
17	2	8	82.0
18	2	9	95.0
19	2	10	96.0
20	3	1	78.0
21	3	2	80.0
22	3	3	87.0
23	3	4	NaN
24	3	5	89.0
25	3	6	90.0
26	3	7	100.0
27	3	8	72.0
28	3	9	73.0
29	3	10	75.0
30	4	1	NaN

	exam	student_id	grade
31	4	2	80.0
32	4	3	81.0
33	4	4	82.0
34	4	5	83.0
35	4	6	84.0
36	4	7	85.0
37	4	8	86.0
38	4	9	87.0
39	4	10	88.0
40	5	1	90.0
41	5	2	NaN
42	5	3	91.0
43	5	4	92.0
44	5	5	93.0
45	5	6	94.0
46	5	7	95.0
47	5	8	96.0
48	5	9	97.0
49	5	10	98.0

In [179…

```
# fill missing values in grade column with zeros
grades['grade'] = grades['grade'].fillna(0)
```

In [180…

```
# display grades
grades
```

Out[180…

	exam	student_id	grade
0	1	1	86.0
1	1	2	65.0
2	1	3	70.0
3	1	4	98.0
4	1	5	89.0
5	1	6	0.0
6	1	7	75.0
7	1	8	56.0

	exam	student_id	grade
8	1	9	90.0
9	1	10	81.0
10	2	1	79.0
11	2	2	60.0
12	2	3	78.0
13	2	4	75.0
14	2	5	0.0
15	2	6	80.0
16	2	7	87.0
17	2	8	82.0
18	2	9	95.0
19	2	10	96.0
20	3	1	78.0
21	3	2	80.0
22	3	3	87.0
23	3	4	0.0
24	3	5	89.0
25	3	6	90.0
26	3	7	100.0
27	3	8	72.0
28	3	9	73.0
29	3	10	75.0
30	4	1	0.0
31	4	2	80.0
32	4	3	81.0
33	4	4	82.0
34	4	5	83.0
35	4	6	84.0
36	4	7	85.0
37	4	8	86.0
38	4	9	87.0
39	4	10	88.0
40	5	1	90.0

	exam	student_id	grade
41	5	2	0.0
42	5	3	91.0
43	5	4	92.0
44	5	5	93.0
45	5	6	94.0
46	5	7	95.0
47	5	8	96.0
48	5	9	97.0
49	5	10	98.0

In [181…

```
# drop student_id column from grades
grades = grades.drop(columns=['student_id'])
```

In [182…

```
# display grades
grades
```

Out[182…

	exam	grade
0	1	86.0
1	1	65.0
2	1	70.0
3	1	98.0
4	1	89.0
5	1	0.0
6	1	75.0
7	1	56.0
8	1	90.0
9	1	81.0
10	2	79.0
11	2	60.0
12	2	78.0
13	2	75.0
14	2	0.0
15	2	80.0
16	2	87.0
17	2	82.0

	exam	grade
18	2	95.0
19	2	96.0
20	3	78.0
21	3	80.0
22	3	87.0
23	3	0.0
24	3	89.0
25	3	90.0
26	3	100.0
27	3	72.0
28	3	73.0
29	3	75.0
30	4	0.0
31	4	80.0
32	4	81.0
33	4	82.0
34	4	83.0
35	4	84.0
36	4	85.0
37	4	86.0
38	4	87.0
39	4	88.0
40	5	90.0
41	5	0.0
42	5	91.0
43	5	92.0
44	5	93.0
45	5	94.0
46	5	95.0
47	5	96.0
48	5	97.0
49	5	98.0

In [183…

```
# rename exam column --- change that column's label from 'exam' to 'exam #'
```

```
grades = grades.rename(columns={'exam': 'exam #'})
```

In [184…

```
# display grades
grades
```

Out[184…

	exam #	grade
0	1	86.0
1	1	65.0
2	1	70.0
3	1	98.0
4	1	89.0
5	1	0.0
6	1	75.0
7	1	56.0
8	1	90.0
9	1	81.0
10	2	79.0
11	2	60.0
12	2	78.0
13	2	75.0
14	2	0.0
15	2	80.0
16	2	87.0
17	2	82.0
18	2	95.0
19	2	96.0
20	3	78.0
21	3	80.0
22	3	87.0
23	3	0.0
24	3	89.0
25	3	90.0
26	3	100.0
27	3	72.0
28	3	73.0
29	3	75.0

	exam #	grade
30	4	0.0
31	4	80.0
32	4	81.0
33	4	82.0
34	4	83.0
35	4	84.0
36	4	85.0
37	4	86.0
38	4	87.0
39	4	88.0
40	5	90.0
41	5	0.0
42	5	91.0
43	5	92.0
44	5	93.0
45	5	94.0
46	5	95.0
47	5	96.0
48	5	97.0
49	5	98.0

Combining Data

In [185…

```
# create a pandas dataframe containing grades on exam #1
# for students with student id numbers 1 through 5
exam1_grades = pd.DataFrame({'SID': [1, 2, 3, 4, 5],
                             'exam1': [86.0, 65.0, 70.0, 98.0, 89.0]})

# create a pandas dataframe containing grades on exam #2
# for students with student id numbers 1 through 5
exam2_grades = pd.DataFrame({'SID': [1, 2, 3, 4, 5],
                             'exam2': [80.0, 87.0, 82.0, 95.0, 96.0]})
```

In [186…

```
# create a pandas dataframe containing grades on exam #1 and exam #2
# for students with id numbers 1 through 7
sid_1_to_7 = pd.DataFrame({'SID': [1, 2, 3, 4, 5, 6, 7],
              'exam1': [86.0, 65.0, 70.0, 98.0, 89.0, 75.0, 56.0],
              'exam2': [80.0, 87.0, 82.0, 95.0, 96.0, 78.0, 80.0]})

# create a pandas dataframe containing grades on exam #1 and exam #2
```

```
# for students with id numbers 8 through 10
sid_8_to_10 = pd.DataFrame({'SID': [8, 9, 10],
                            'exam1': [90.0, 81.0, 0.0],
                            'exam2': [87.0, 82.0, 95.0]})
```

In [187…

```
# create a pandas dataframe containing grades on exam #1 and exam #2
# for students 1 through 5
exams1and2 = pd.DataFrame({'exam1': [86.0, 65.0, 70.0, 98.0, 89.0],
              'exam2': [80.0, 87.0, 82.0, 95.0, 96.0]},
              index=['student1', 'student2', 'student3', 'student4', 'student5'])

# create a pandas dataframe containing grades on exam #3 for students 1 through 5
exam3 = pd.DataFrame({'exam3': [78.0, 80.0, 87.0, 89.0, 89.0]},
              index=['student1', 'student2', 'student3', 'student4', 'student5'])
```

In [188…

```
# display exam1_grades
exam1_grades
```

Out[188…

	SID	exam1
0	1	86.0
1	2	65.0
2	3	70.0
3	4	98.0
4	5	89.0

In [189…

```
# display exam2_grades
exam2_grades
```

Out[189…

	SID	exam2
0	1	80.0
1	2	87.0
2	3	82.0
3	4	95.0
4	5	96.0

In [190…

```
# merge exam1_grades and exam2_grades on 'SID'
# result will be a new pandas dataframe
pd.merge(exam1_grades, exam2_grades, on='SID')
```

Out[190…

	SID	exam1	exam2
0	1	86.0	80.0
1	2	65.0	87.0

	SID	exam1	exam2
2	3	70.0	82.0
3	4	98.0	95.0
4	5	89.0	96.0

In [191…
```
# display sid_1_to_7
sid_1_to_7
```

Out[191…

	SID	exam1	exam2
0	1	86.0	80.0
1	2	65.0	87.0
2	3	70.0	82.0
3	4	98.0	95.0
4	5	89.0	96.0
5	6	75.0	78.0
6	7	56.0	80.0

In [192…
```
# display sid_8_to_10
sid_8_to_10
```

Out[192…

	SID	exam1	exam2
0	8	90.0	87.0
1	9	81.0	82.0
2	10	0.0	95.0

In [193…
```
# concatenate sid_1_to_7 and sid_8_to_10 along the rows (along axis 0)
# result will be a new pandas dataframe
pd.concat([sid_1_to_7, sid_8_to_10], axis=0)
```

Out[193…

	SID	exam1	exam2
0	1	86.0	80.0
1	2	65.0	87.0
2	3	70.0	82.0
3	4	98.0	95.0
4	5	89.0	96.0
5	6	75.0	78.0
6	7	56.0	80.0

	SID	exam1	exam2
0	8	90.0	87.0
1	9	81.0	82.0
2	10	0.0	95.0

In [194…
```
# display exams1and2
exams1and2
```

Out[194…

	exam1	exam2
student1	86.0	80.0
student2	65.0	87.0
student3	70.0	82.0
student4	98.0	95.0
student5	89.0	96.0

In [195…
```
# display exam3
exam3
```

Out[195…

	exam3
student1	78.0
student2	80.0
student3	87.0
student4	89.0
student5	89.0

In [196…
```
# concatenate exams1and2 and exam3 along the columns (along axis 1)
# result will be a new pandas dataframe
pd.concat([exams1and2, exam3], axis=1)
```

Out[196…

	exam1	exam2	exam3
student1	86.0	80.0	78.0
student2	65.0	87.0	80.0
student3	70.0	82.0	87.0
student4	98.0	95.0	89.0
student5	89.0	96.0	89.0

Grouping Data Pandas

In [197…
```
# read grades.csv into a pandas dataframe & save the dataframe in a variable
grades = pd.read_csv('grades.csv')
```

In [198…
```
# display grades
grades
```

Out[198…

	exam	student_id	grade
0	1	1	86.0
1	1	2	65.0
2	1	3	70.0
3	1	4	98.0
4	1	5	89.0
5	1	6	NaN
6	1	7	75.0
7	1	8	56.0
8	1	9	90.0
9	1	10	81.0
10	2	1	79.0
11	2	2	60.0
12	2	3	78.0
13	2	4	75.0
14	2	5	NaN
15	2	6	80.0
16	2	7	87.0
17	2	8	82.0
18	2	9	95.0
19	2	10	96.0
20	3	1	78.0
21	3	2	80.0
22	3	3	87.0
23	3	4	NaN
24	3	5	89.0
25	3	6	90.0
26	3	7	100.0
27	3	8	72.0
28	3	9	73.0

	exam	student_id	grade
29	3	10	75.0
30	4	1	NaN
31	4	2	80.0
32	4	3	81.0
33	4	4	82.0
34	4	5	83.0
35	4	6	84.0
36	4	7	85.0
37	4	8	86.0
38	4	9	87.0
39	4	10	88.0
40	5	1	90.0
41	5	2	NaN
42	5	3	91.0
43	5	4	92.0
44	5	5	93.0
45	5	6	94.0
46	5	7	95.0
47	5	8	96.0
48	5	9	97.0
49	5	10	98.0

In [199…

```
# drop exam column from grades, group by student_id,
# and compute mean grade for each student_id
# result will be a new pandas dataframe
grades.drop(columns='exam').groupby('student_id').mean()
```

Out[199…

	grade
student_id	
1	83.25
2	71.25
3	81.40
4	86.75
5	88.50
6	87.00

student_id	grade
7	88.40
8	78.40
9	88.40
10	87.60

In [200…
```
# drop student_id column from grades, group by exam,
# and compute mean grade for each exam
# result will be a new pandas dataframe
grades.drop(columns='student_id').groupby('exam').mean()
```

Out[200…

exam	grade
1	78.888889
2	81.333333
3	82.666667
4	84.000000
5	94.000000

Chapter 6

Line Plots

In [205…
```
# import relevant libraries and modules
import pandas as pd
import matplotlib.pyplot as plt
```

In [206…
```
# create a list of days
days = [1,2,3,4,5,6,7]

# create a list of temperatures (in fahrenheit)
temps = [72,73,77,81,79,72,68]

# create a pandas dataframe
# containing the temperatues (in fahrenheit) for 7 consecutive days in Santa Barbara, C
temp_sb = pd.DataFrame(data={'day':days,'temperature':temps})
```

In [207…
```
# create a list of days
days = [1,2,3,4,5,6,7]

# create a list of temperatures (in fahrenheit)
temps = [93,91,91,91,91,88,90]
```

```
# create a pandas dataframe
# containing the temperatues (in fahrenheit) for 7 consecutive days in Memphis, TN
temp_mem = pd.DataFrame(data={'day':days,'temperature':temps})
```

In [208…

```
# display temp_sb
temp_sb
```

Out[208…

	day	temperature
0	1	72
1	2	73
2	3	77
3	4	81
4	5	79
5	6	72
6	7	68

In [209…

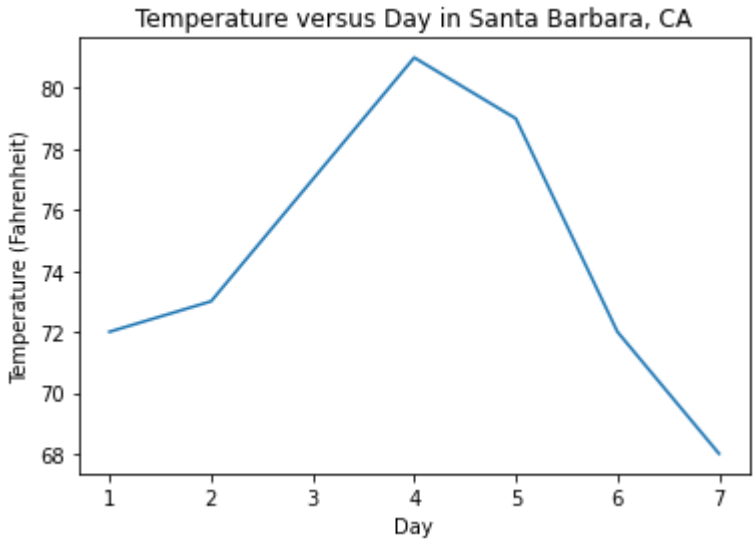
```
# display temp_mem
temp_mem
```

Out[209…

	day	temperature
0	1	93
1	2	91
2	3	91
3	4	91
4	5	91
5	6	88
6	7	90

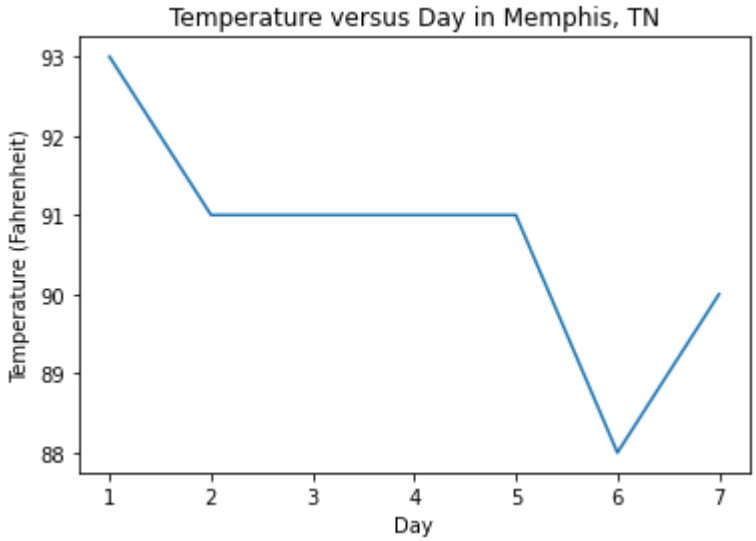
In [210…

```
# create a line plot of Santa Barbara data:
# plot temperature versus day in Santa Barbara
plt.plot(temp_sb['day'], temp_sb['temperature'])
plt.xlabel('Day')
plt.ylabel('Temperature (Fahrenheit)')
plt.title('Temperature versus Day in Santa Barbara, CA')
plt.show()
```



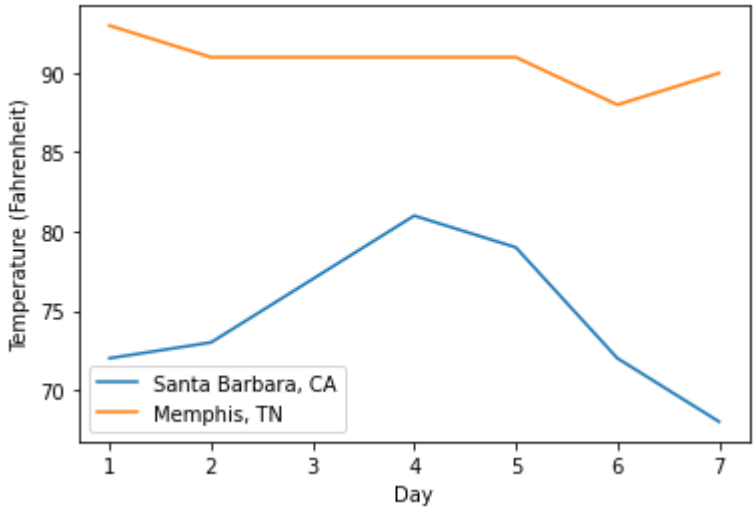
In [211…

```
# create a line plot of Memphis data:
# plot temperature versus day in Memphis
plt.plot(temp_mem['day'], temp_mem['temperature'])
plt.xlabel('Day')
plt.ylabel('Temperature (Fahrenheit)')
plt.title('Temperature versus Day in Memphis, TN')
plt.show()
```



In [212…

```
# plot santa barbara data and memphis data on the same line plot with a legend
plt.plot(temp_sb['day'], temp_sb['temperature'], label='Santa Barbara, CA')
plt.plot(temp_mem['day'], temp_mem['temperature'], label='Memphis, TN')
plt.xlabel('Day')
plt.ylabel('Temperature (Fahrenheit)')
plt.legend()
plt.show()
```



Scatter Plot

In [213…
```
# import relevant libraries and modules
import pandas as pd
import matplotlib.pyplot as plt
```

In [214…
```
# create a list of temperatures (in fahrenheit)
temp = [78, 89, 73, 75, 90, 99, 101,100,50,68,81, 85, 86, 70]

# create a list of sales (in $)
sale = [216,275,203,210,315,402,380,374,90,95,115,130,150,140]

# create a pandas dataframe containing the the sales
# (in dollars) of an ice cream truck for 14 days
# and the (temperature) in fahrenheit for each day
sales = pd.DataFrame(data={'temp':temp,'ice cream truck sale':sale})
```

In [215…
```
# display sales
sales
```

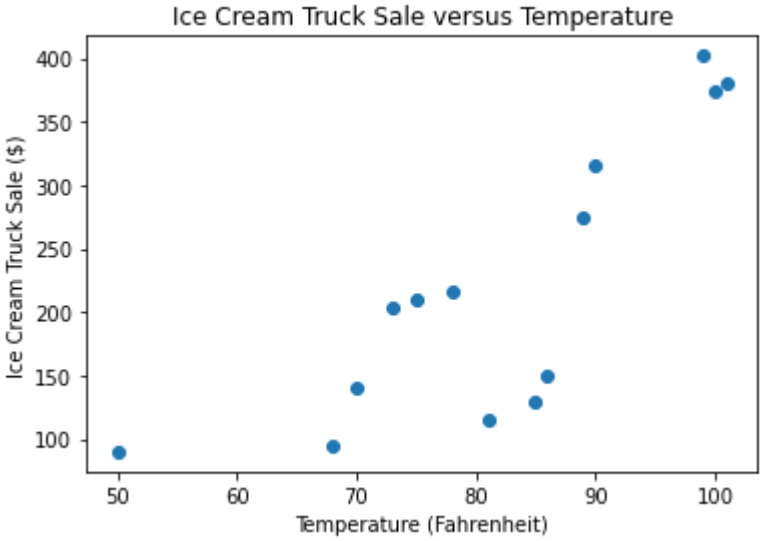
Out[215…

	temp	ice cream truck sale
0	78	216
1	89	275
2	73	203
3	75	210
4	90	315
5	99	402
6	101	380
7	100	374
8	50	90

	temp	ice cream truck sale
9	68	95
10	81	115
11	85	130
12	86	150
13	70	140

In [204…

```
# create a scatter plot --- plot ice cream truck sale versus temp
plt.scatter(sales['temp'], sales['ice cream truck sale'])
plt.xlabel('Temperature (Fahrenheit)')
plt.ylabel('Ice Cream Truck Sale ($)')
plt.title('Ice Cream Truck Sale versus Temperature')
plt.show()
```



Bar Plots

In [216…

```
# create a pandas dataframe containing the means for five exams taken by a set of stude
# save in a variable named exam_means
exam_means = pd.DataFrame(data={'exam':[1,2,3,4,5],
                                'mean':[73.2,71.5,79.0,62.0,84.6]})
```

In [217…

```
# display exam_means
exam_means
```

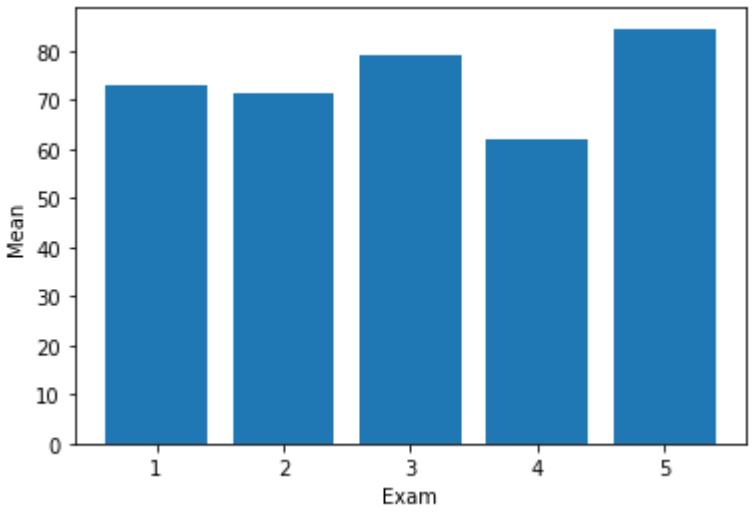
Out[217…

	exam	mean
0	1	73.2
1	2	71.5
2	3	79.0
3	4	62.0

	exam	mean
4	5	84.6

In [218…

```
# create a bar plot --- plot mean versus exam
plt.bar(exam_means['exam'], exam_means['mean'])
plt.xlabel('Exam')
plt.ylabel('Mean')
plt.show()
```



Pie Chart

In [219…

```
# create a list of categories
categ = ['Homework', 'Labs', 'Quizzes', 'Midterm', 'Final']

# create a list of weights (in %)
weights = [15, 15, 15, 22, 33]

# create a pandas dataframe containing the weighted components
# used to give students their grades in a particular course
breakdown = pd.DataFrame(data={'category':categ,'weight':weights})
```

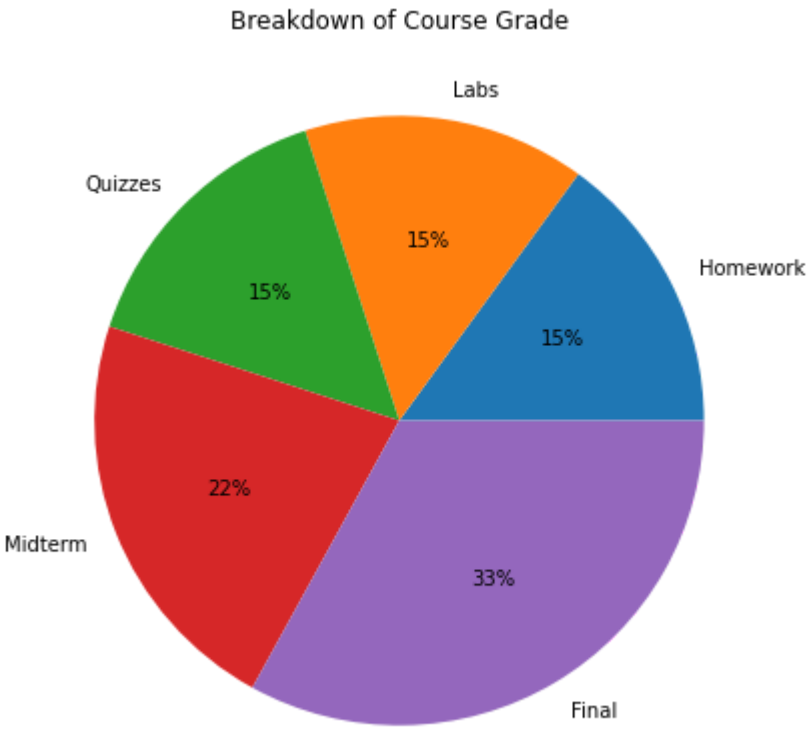
In [220…

```
# display breakdown
breakdown
```

Out[220…

	category	weight
0	Homework	15
1	Labs	15
2	Quizzes	15
3	Midterm	22
4	Final	33

In [221…
```
# create a pie chart --- illustrate the breakdown of students' course
# grades in a particular course
plt.figure(figsize=(7,7))
plt.pie(breakdown['weight'], labels=breakdown['category'],
        autopct='%1.0f%%')
plt.title('Breakdown of Course Grade')
plt.show()
```



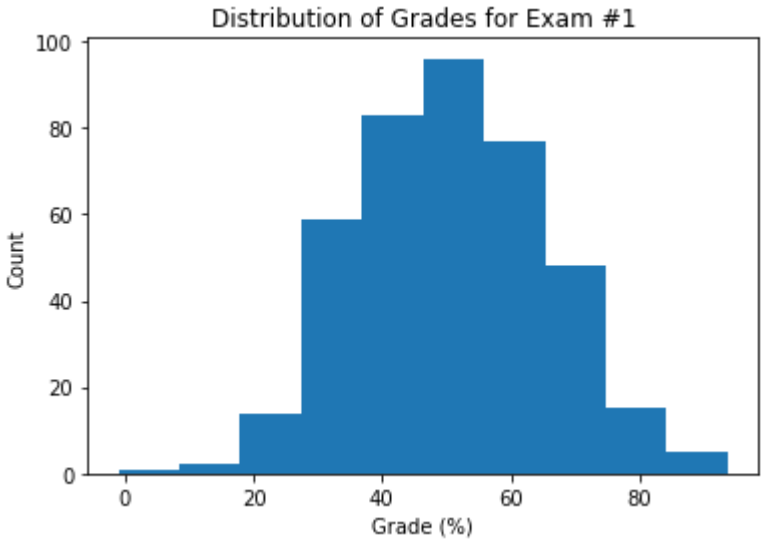
Histogram

In [222…
```
# create a pandas series containing students' grades on Exam #1 in a particular course
exam1_grades = pd.Series(np.random.normal(50, 15, 400))
```

In [223…
```
# display first few values in exam1_grades
exam1_grades.head()
```

Out[223…
```
0    56.979267
1    48.140717
2    46.521488
3    30.051874
4    53.095763
dtype: float64
```

In [224…
```
# create a histogram --- visualize exam1_grades
plt.hist(exam1_grades)
plt.xlabel('Grade (%)')
plt.ylabel('Count')
plt.title('Distribution of Grades for Exam #1')
plt.show()
```



Subplots

In [225…
```
# create a pandas series containing students' grades on Exam #1 in a particular course
exam1_grades = pd.Series(np.random.normal(50, 15, 400))
```

In [226…
```
# create a pandas series containing students' grades on Exam #2 in the same course
exam2_grades = pd.Series(np.random.normal(60, 10, 400))
```

In [227…
```
# display first few values in exam1_grades
exam1_grades.head()
```

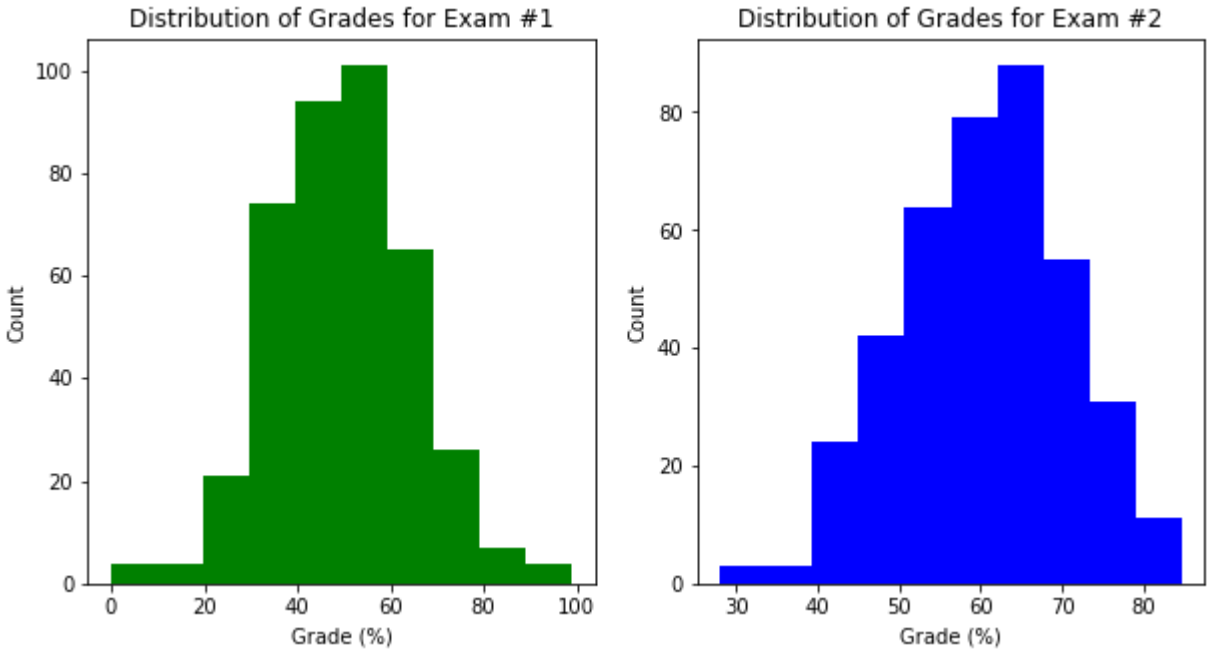
Out[227…
```
0    42.144654
1    58.725325
2    49.573887
3    53.166365
4    53.719261
dtype: float64
```

In [228…
```
# display first few values in exam2_grades
exam2_grades.head()
```

Out[228…
```
0    55.975289
1    51.774943
2    81.480904
3    62.079183
4    46.547136
dtype: float64
```

In [229…
```
# create 2 horizontally stacked subplots:
# display histogram for exam1_grades and histogram for exam2_grades side by side

figure, axes = plt.subplots(1, 2, figsize=(10,5))

axes[0].hist(exam1_grades, color='green')
axes[0].set_title('Distribution of Grades for Exam #1')
axes[1].hist(exam2_grades, color='blue')
```

```
axes[1].set_title('Distribution of Grades for Exam #2')

for a in axes.flat:
    a.set(xlabel='Grade (%)', ylabel='Count')

plt.show()
```



Chapter 7

BoxPlots

In [233…

```
# import relevant libraries
import pandas as pd
import seaborn as sns
# read grades.csv into a pandas dataframe & save the dataframe in a variable
grades = pd.read_csv('grades.csv')
```

In [234…

```
# display first few rows of grades
grades.head()
```

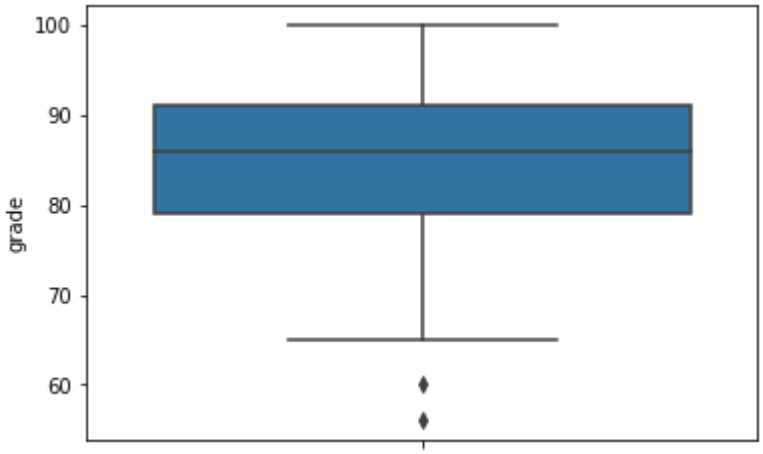
Out[234…

	exam	student_id	grade
0	1	1	86.0
1	1	2	65.0
2	1	3	70.0
3	1	4	98.0
4	1	5	89.0

In [235…

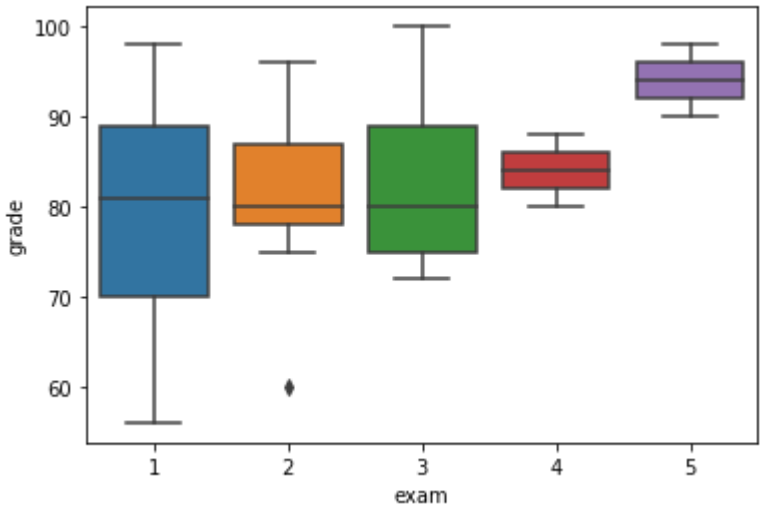
```
# create a vertical box plot of the data in the grade column of the grades dataset
```

```
sns.boxplot(y='grade', data=grades);
```



In [236…

```
# create a vertical boxplot of the data in the grade column of the grades dataset
# grouped by exam
sns.boxplot(x='exam', y='grade', data=grades);
```



Kernel Density

In [237…

```
# create a pandas series representing the scores of 400 students
# on a particular exam
scores = pd.Series(np.random.normal(70, 15, size=400))
```

In [238…

```
# initialize mean as a list containing two means,
# corresponding to the bivariate distribution
mean = [60, 70]

# initialize covariance_matrix as the covariance matrix
# of the bivariate distribution
covariance_matrix = [(1, .5), (.5, 1)]

# initialize data as a two-dimensional numpy array containing
# random samples drawn from
# the specified bivariate normal distribution
```

```
data = np.random.multivariate_normal(mean,
                                     covariance_matrix, size=400)

# initialize midterms as a pandas dataframe
# containing data as column values
# and midterm1, midterm2 as column labels
midterms = pd.DataFrame(data, columns=["midterm1", "midterm2"])
```

In [239…

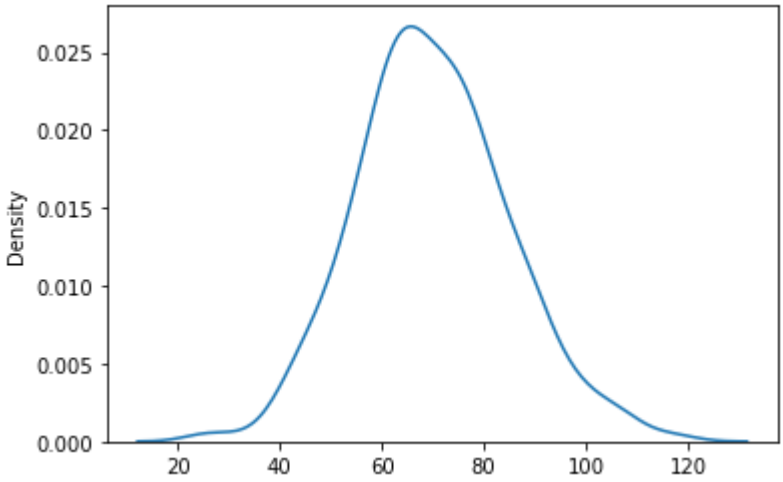
```
# display first few values in scores
scores.head()
```

Out[239…

```
0    66.287644
1    66.828713
2    50.950913
3    62.745157
4    64.495271
dtype: float64
```

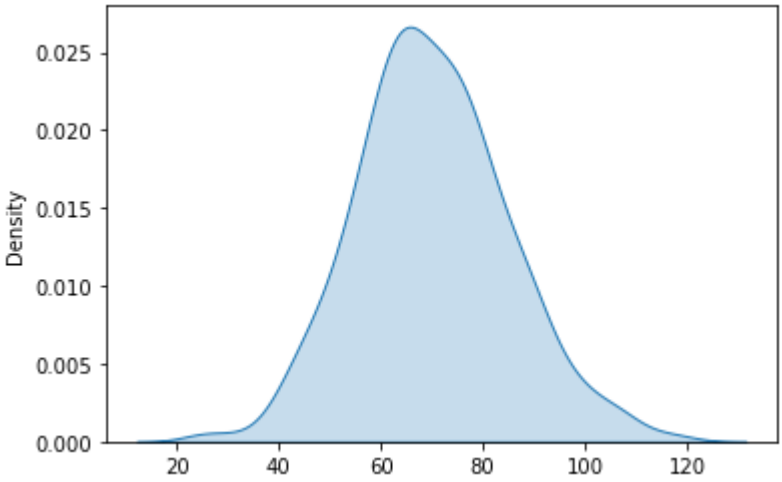
In [240…

```
# create a kernel density estimate plot
# to visualize the univariate distribution of scores
sns.kdeplot(scores);
```



In [241…

```
# create a shaded kernel density estimate plot
# to visualize the univariate distribution of scores
sns.kdeplot(scores, shade=True);
```



In [242…

```
# display first few rows of midterms
midterms.head()
```

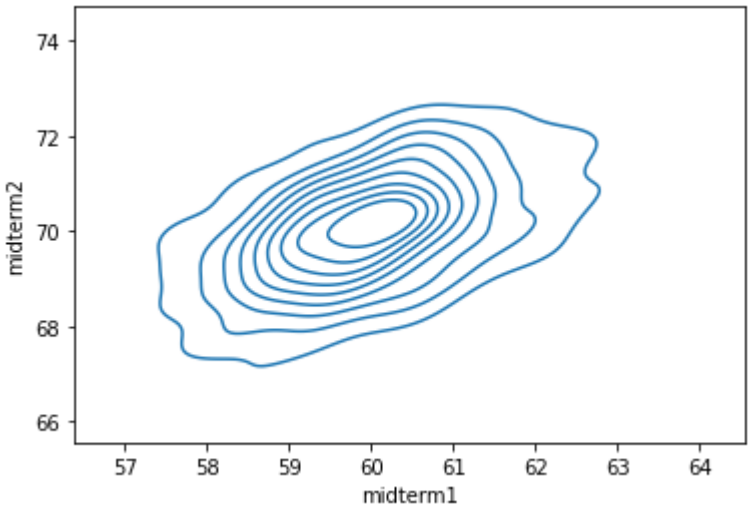
Out[242…

	midterm1	midterm2
0	59.850150	69.704398
1	60.497660	71.662953
2	59.089453	69.945242
3	58.844031	68.423547
4	59.241816	70.689407

In [243…

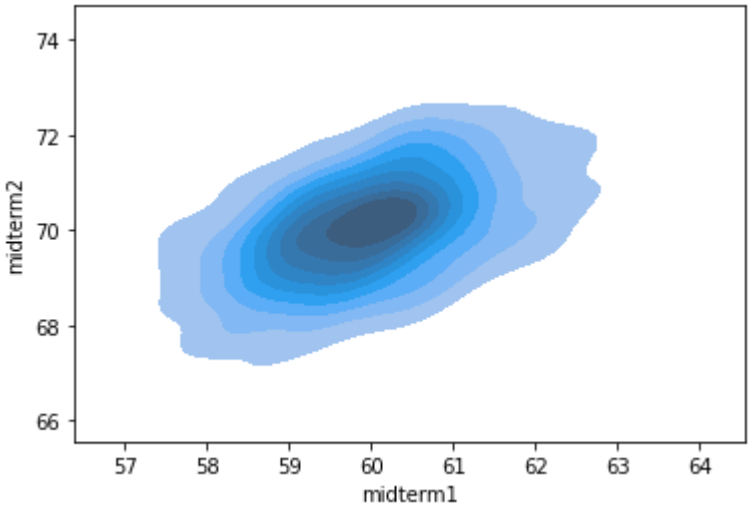
```
# create a two-dimensional kernel density estimate plot
# to visualize the bivariate distribution of midterm1 and midterm2
sns.kdeplot(midterms['midterm1'], midterms['midterm2']);
```

```
S:\Anaconda\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the followi
ng variable as a keyword arg: y. From version 0.12, the only valid positional argument w
ill be `data`, and passing other arguments without an explicit keyword will result in an
error or misinterpretation.
  warnings.warn(
```



In [244…
```
# create a shaded two-dimensional kernel density estimate plot
# to visualize the bivariate distribution of midterm1 and midterm2
sns.kdeplot(midterms['midterm1'], midterms['midterm2'], shade=True);
```

```
S:\Anaconda\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the followi
ng variable as a keyword arg: y. From version 0.12, the only valid positional argument w
ill be `data`, and passing other arguments without an explicit keyword will result in an
error or misinterpretation.
  warnings.warn(
```



Violin Plots

In [245…
```
# create a numpy array representing the scores of 400 students
# on midterm1
m1 = np.random.normal(70, 10, size=400)

# create a numpy array representing the students' scores
# on midterm2
m2 = np.random.normal(80, 15, size=400)

# create a numpy array representing the students' scores
# on the final exam
final = np.random.normal(75, 20, size=400)

# create a list of the exam types
exams = ['midterm1']*400 + ['midterm2']*400 + ['final']*400

# create a numpy array containing all the scores
scores = np.append(m1, np.array([m2, final]))

# create a pandas dataframe representing midterm1, midterm2, and final exam scores
# for the class of 400 students
exam_scores = pd.DataFrame({'exam': exams, 'score': scores})
```

In [246…
```
# display first few rows of exam_scores
exam_scores.head()
```

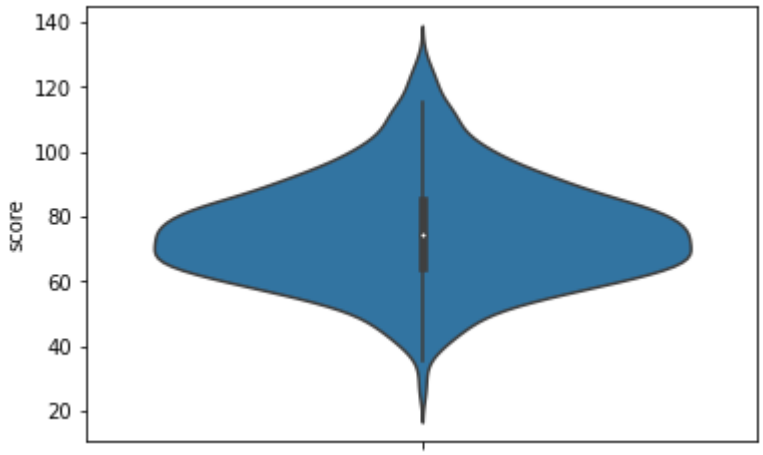
Out[246…

	exam	score

	exam	score
0	midterm1	63.484465
1	midterm1	65.329124
2	midterm1	71.971705
3	midterm1	62.939158
4	midterm1	77.619794

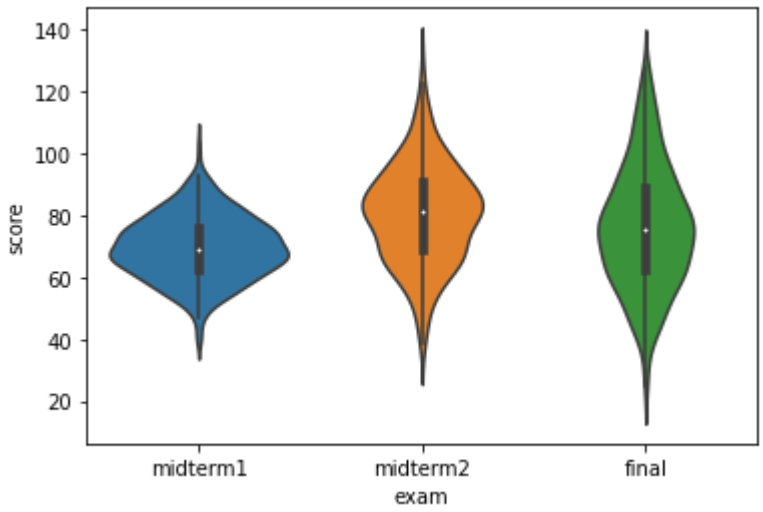
In [247…

```
# draw a vertical violin plot of the scores
sns.violinplot(y='score', data=exam_scores);
```



In [248…

```
# draw a vertical violin plot of the scores grouped by exam type
sns.violinplot(x='exam', y='score', data=exam_scores);
```



Heatmaps

In [249…

```
# create a variable that contains a numpy array representing the sales of a small busin
data = np.random.randint(100, 301, size=(12,12))

# create a variable that contains the list of months
```

```
months = ['January', 'February', 'March', 'April', 'May', 'June', 'July',
          'August', 'September', 'October', 'November', 'December']

# create a variable that contains a list of the years from 2008 to 2019 inclusive
years = [2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019]

# create a variable named sales that contains a dataframe
# the dataframe consists of the sales of a small business across 12 months
# there is an entry for each month of each year between 2008 and 2019 inclusive
sales = pd.DataFrame(data, index=months, columns=years)
```

In [250…

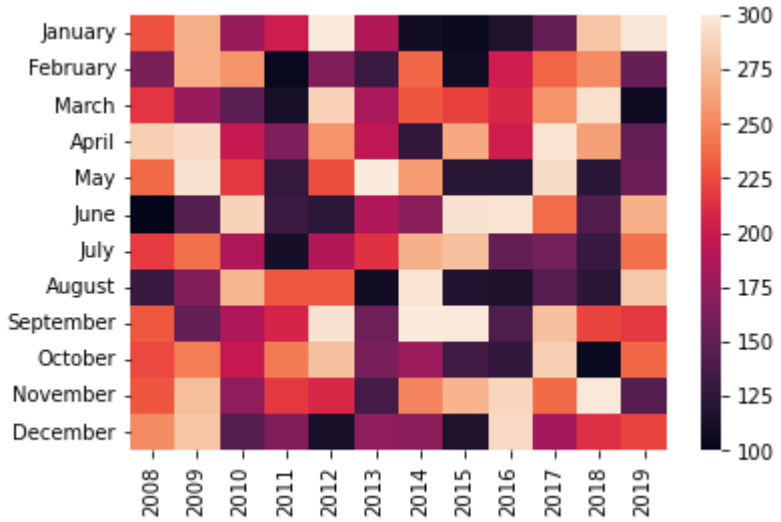
```
# display first few rows of sales
sales.head()
```

Out[250…

	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019
January	228	269	176	201	300	188	109	105	116	151	280	297
February	163	268	256	104	165	131	236	109	202	235	252	153
March	215	177	147	113	286	183	229	220	209	256	294	108
April	285	291	198	164	256	194	126	266	203	296	261	150
May	237	295	217	128	226	300	259	121	121	292	122	155

In [251…

```
# create a heatmap of sales
sns.heatmap(sales);
```



In [ ]: