Assignment 2

- Due: Tuesday 2/25/2020 by 9 PM
- Topics: pandas and numpy

Instructions

- Download your notebook as a PDF
- Turn in both the notebook and PDF
- Show the output of your code working on the sample data, like below.
- If techniques are specified, use them. Otherwise, feel free to solve the problem how you want.
- Try to keep your code clean, concise and avoid loops if necessary.
- You have to show your work (code).
- You will be graded:
  - The functionality of your code (Does it do what it was meant to)
  - Showing the output on the sample data provided
  - How concise it is (did you use a for loop when you could have used a comprehension for instance). Simply put, try and write clean, concise and readable code (don't use 10 lines for what could have been done in 4).
  - Even if the answer isn't perfect, make an honest attempt as partial credit will be given.

Question 1

- sklearn has the iris dataset built in
- this is what we will be working with
- put the iris data into a dataframe where we have 5 columns (the 4 features and the target)
- but the target shouldn't be numerical, it should be the actual label (so put setosa not 0 in the target column)

In [1]:
```
from sklearn.datasets import load_iris
import pandas as pd
import numpy as np

iris = load_iris()
```

In [2]:
```
target_dict = dict(zip(set(iris['target']), iris['target_names']))
df1 = pd.DataFrame(iris['data'], columns = iris['feature_names'])
df2 = pd.DataFrame({'target': np.vectorize(target_dict.get)(iris['targe
t'])})
iris_dataframe = pd.concat([df1,df2], axis = 1, sort = False)
```

In [3]:
```
iris_dataframe.head()
```

Out[3]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

Question 2

- using the data array from the iris variable (should be a numpy array of size 150x4), for each row and column, find the sum, min and max
- print out the results, but for the rows, show only the first 5 elements of the array and display the length (otherwise it will be large arrays since each row has a value)

In [4]:
```
# Rows
sum_rows = np.sum(iris['data'], axis = 1)
print("Sum:", sum_rows[:5], "length:", len(sum_rows))
min_rows = np.min(iris['data'], axis = 1)
print("Min:", min_rows[:5], "length:", len(min_rows))
max_rows = np.max(iris['data'], axis = 1)
print("Max:", max_rows[:5], "length:", len(max_rows))
```

```
Sum: [10.2  9.5  9.4  9.4 10.2] length: 150
Min: [0.2 0.2 0.2 0.2 0.2] length: 150
Max: [5.1 4.9 4.7 4.6 5. ] length: 150
```

In [5]:
```
# Columns
sum_cols = np.sum(iris['data'], axis = 0)
print("Sum:", sum_cols, "length:", len(sum_cols))
min_cols = np.min(iris['data'], axis = 0)
print("Min:", min_cols, "length:", len(min_cols))
max_cols = np.max(iris['data'], axis = 0)
print("Max:", max_cols, "length:", len(max_cols))
```

```
Sum: [876.5 458.6 563.7 179.9] length: 4
Min: [4.3 2.  1.  0.1] length: 4
Max: [7.9 4.4 6.9 2.5] length: 4
```

Question 3

- write a function to return the multiplication of two matrices
- use some sort of error handling to make sure the dimensions are compatible
- use the function to multiply the iris data (don't use the label column) by it's transpose
- what are the dimensions of the result
- put the sum of each row of the resulting matrix into a vector, printing out the length and first 5 elements.
- put the sum of each column of the resulting matrix in a vector and print it out.

In [6]:
```
def matrix_multiplication (mat1, mat2):
    assert len(mat1[0]) == len(mat2), "Please make sure the dimensions a
re compatible."

    result = np.array([[sum(a*b for a,b in zip(mat1_row,mat2_col)) for m
at2_col in zip(*mat2)] for mat1_row in mat1])

    return result
```

In [7]:
```
answer = matrix_multiplication(iris['data'], np.transpose(iris['data']))
print(answer.shape)
print("First five elements of sum of each row:", np.sum(answer, axis = 1
)[:5], "Length:", len(np.sum(answer, axis = 1)))
print("First five elements of sum of each column:", np.sum(answer, axis
= 0)[:5], "Length:", len(np.sum(answer, axis = 0)))
```

```
(150, 150)
First five elements of sum of each row: [6900.41 6495.81 6355.86 6335.0
9 6858.62] Length: 150
First five elements of sum of each column: [6900.41 6495.81 6355.86 633
5.09 6858.62] Length: 150
```

Question 4

- create a subset of the iris dataframe that doesn't have the label column
- for each row, find the index of the min and max feature value
  - if we have a row with values [10,2,3,4] we want to return 1 for min and 0 for max, as these are the indicies that align to the min and max values
- the result should be two arrays of length 150 (one for each row) with the indices of the min and max value

In [8]:
```
iris_X = iris_dataframe.iloc[:,0:4]
min_indice = np.argmin(np.array(iris_X), axis=1)
max_indice = np.argmax(np.array(iris_X), axis=1)
```

In [9]:
```
print(min_indice)
print(max_indice)
```

```
[3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0]
```

Question 5

- Use the above two arrays to make a dataframe with 2 columns, the first being the feature of minimum value and the second the feature name of the maximum value
- note, don't have the cell values be the index value of the min/max value, have it be the feature name
  - a row should be [min_val = sepal width (cm), max_val = petal length (cm)]
- show the distributions for max and min features (how many times is each feature the max and min value for a row)

In [10]:
```
answer = pd.DataFrame({"min_val": [iris['feature_names'][i] for i in min
_indice], "max_val": [iris['feature_names'][i] for i in max_indice]})
```

In [11]:
```
print(answer['min_val'].value_counts())
print(answer['max_val'].value_counts())
```

```
petal width (cm)    150
Name: min_val, dtype: int64
sepal length (cm)    150
Name: max_val, dtype: int64
```

Question 6

- Describe a situation the functionality from question 4 and 5 could be of use?

If we have a sales table where each row is a month in the year and each column is the total monthly sales of one item, we can check the sales distribution of differnet items in each month by using such functionalities.

Question 7

- sort the feature values for each row and replace the indices with the feature names
  - so each row will have 4 columns, the first column being the feature name that is the highest value, the second column being the feature name that is the second highest value, etc.
- note, watch out how argsort in numpy works. you will need to reverse the order somehow. the sorted(reverse = True) functionality might be of help.
- make sure to replace the index values with the feature name, as we did above
- put the resulting 2-d array into a pandas dataframe and print the first 5 rows.
- hint, look at the apply_along_axis() method for numpy arrays

In [14]:
```
answer = pd.DataFrame(iris_X.columns[np.argsort(-np.array(iris_X), axis=
1)])
print(answer.head(5))
```

```
                    0                 1                   2
3
0  sepal length (cm)  sepal width (cm)  petal length (cm)  petal width
(cm)
1  sepal length (cm)  sepal width (cm)  petal length (cm)  petal width
(cm)
2  sepal length (cm)  sepal width (cm)  petal length (cm)  petal width
(cm)
3  sepal length (cm)  sepal width (cm)  petal length (cm)  petal width
(cm)
4  sepal length (cm)  sepal width (cm)  petal length (cm)  petal width
(cm)

/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:1: Depreca
tionWarning: Support for multi-dimensional indexing (e.g. `index[:, Non
e]`) on an Index is deprecated and will be removed in a future version.
Convert to a numpy array before indexing instead.
  """Entry point for launching an IPython kernel.
```

Question 8

- apply z-score normalization to each column in the iris data (note you do not need the target/label column)
- column wise meaning, treat each column as an array, and find the standard deviations and means of that column
- note, you can use zscore from scipy.stats
- results should be a pandas dataframe, printing out 5 rows and running the describe() method on the dataframe

In [15]:
```
from scipy import stats
answer = pd.DataFrame(stats.zscore(iris['data'], axis = 0), columns = ir
is['feature_names'])
print(answer.head(5))
```

```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width
(cm)
0          -0.900681          1.019004          -1.340227         -1.31
5444
1          -1.143017         -0.131979          -1.340227         -1.31
5444
2          -1.385353          0.328414          -1.397064         -1.31
5444
3          -1.506521          0.098217          -1.283389         -1.31
5444
4          -1.021849          1.249201          -1.340227         -1.31
5444
```

In [16]:
```
print(answer.describe())
```

```
       sepal length (cm)  sepal width (cm)  petal length (cm)  \
count       1.500000e+02      1.500000e+02       1.500000e+02
mean       -1.468455e-15     -1.823726e-15      -1.610564e-15
std         1.003350e+00      1.003350e+00       1.003350e+00
min        -1.870024e+00     -2.433947e+00      -1.567576e+00
25%        -9.006812e-01     -5.923730e-01      -1.226552e+00
50%        -5.250608e-02     -1.319795e-01       3.364776e-01
75%         6.745011e-01      5.586108e-01       7.627583e-01
max         2.492019e+00      3.090775e+00       1.785832e+00

       petal width (cm)
count      1.500000e+02
mean      -9.473903e-16
std        1.003350e+00
min       -1.447076e+00
25%       -1.183812e+00
50%        1.325097e-01
75%        7.906707e-01
max        1.712096e+00
```

Question 9

- make a function that takes in a 2d numpy array X, a 1d numpy array Y and the size the training data, test data and valiation datasets.
- return 6 items
  - train_x, train_y, test_x, test_y, val_x, val_y
- do not use any modules. this can be solved using bracket notation to subset. Note // will take care of decimals in doing division. you could also use int() to convert the float to a whole number
- make the params for the training, test and val size be decimals, that repsent percentages of the data. For instance .8, .1, .1 means an 80% training, 10% test and 10% validation split
- use an assert to make sure the numbers add to 1
- print out the dimensions of all 6 elements, using iris as a test case with an 80-10-10 split

In [17]:
```
def train_test_split (df_X, df_y, training_size, test_size, val_size):
    assert training_size + test_size + val_size == 1, "The sizes should
 be decimals and add up to 1."

    length = len(df_X)
    index = [i for i in range(length)]
    import random
    index_train = random.sample(index, int(length*training_size))
    index_test = random.sample([i for i in index if i not in index_train
], int(length*test_size))
    index_val = [i for i in index if i not in index_train + index_test]

    train_x = df_X[index_train]
    train_y = df_y[index_train]
    test_x = df_X[index_test]
    test_y = df_y[index_test]
    val_x = df_X[index_val]
    val_y = df_y[index_val]

    return(train_x, train_y, test_x, test_y, val_x, val_y)
```

In [18]:
```
train_x, train_y, test_x, test_y, val_x, val_y = train_test_split(iris[
'data'], iris['target'], .8, .1, .1)
```

In [19]:
```
print(train_x.shape)
print(train_y.shape)
print(test_x.shape)
print(test_y.shape)
print(val_x.shape)
print(val_y.shape)
```

```
(120, 4)
(120,)
(15, 4)
(15,)
(15, 4)
(15,)
```

Question 10

- using pandas, find the sum of each feature by species type (label)
- do the same, but find the min, max and median as well

In [20]:
```
# sum
iris_dataframe.groupby('target').sum()
```

Out[20]:

target	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
setosa	250.3	171.4	73.1	12.3
versicolor	296.8	138.5	213.0	66.3
virginica	329.4	148.7	277.6	101.3

In [21]:
```
# min
iris_dataframe.groupby('target').min()
```

Out[21]:

target	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
setosa	4.3	2.3	1.0	0.1
versicolor	4.9	2.0	3.0	1.0
virginica	4.9	2.2	4.5	1.4

In [22]:
```
# max
iris_dataframe.groupby('target').max()
```

Out[22]:

target	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
setosa	5.8	4.4	1.9	0.6
versicolor	7.0	3.4	5.1	1.8
virginica	7.9	3.8	6.9	2.5

In [23]:
```
# median
iris_dataframe.groupby('target').median()
```

Out[23]:

target	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
setosa	5.0	3.4	1.50	0.2
versicolor	5.9	2.8	4.35	1.3
virginica	6.5	3.0	5.55	2.0

Question 11

- mean center each column of the iris dataframe (excluding the label column of course)
- this means, for each column, the mean should be zero. to accomplish this we can subtract the column mean from each element of the column
- note, broadcasting, which if we have say a 150 row and 4 column dataframe, can take a row vector of size 4 and apply it to each element
- thus, we can answer this questions doing something like df - df_col_means
- run the describe() method at the end to show the data has been mean center

In [24]:
```
answer = iris_X - np.array(iris_X.mean(axis=0))
```

In [25]:
```
answer.describe()
```

Out[25]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
count	1.500000e+02	1.500000e+02	1.500000e+02	1.500000e+02
mean	-1.184238e-15	-7.815970e-16	-2.747432e-15	-7.105427e-16
std	8.280661e-01	4.358663e-01	1.765298e+00	7.622377e-01
min	-1.543333e+00	-1.057333e+00	-2.758000e+00	-1.099333e+00
25%	-7.433333e-01	-2.573333e-01	-2.158000e+00	-8.993333e-01
50%	-4.333333e-02	-5.733333e-02	5.920000e-01	1.006667e-01
75%	5.566667e-01	2.426667e-01	1.342000e+00	6.006667e-01
max	2.056667e+00	1.342667e+00	3.142000e+00	1.300667e+00

Quesiton 12

- Explain what the axis mean in regards to a pandas dataframe and numpy array
  A dataframe should have two axes. Axis 1 represents rows and axis0 represent columns. A multidimensional numpy array has a number of axes equal to the number of dimensions. Axis0 is the first dimension, axis1 is the second dimension, etc.
- How would you groupby two columns in pandas?
  df.groupby([col1_name, col2_name])
- What functions are used to read in csvs and excel files in pandas
  pd.read_csv pd.read_excel

Question 13

- using Pandas, subset the dataset for only species setosa and petal length > 1.3
- sum the feature columns and display the results for each (excluding the label column)
- do the same, but change the and to an or, and repeat the same calculation

In [26]:
```
answer1 = iris_dataframe[(iris_dataframe['target'] == 'setosa') & (iris_
dataframe['petal length (cm)'] > 1.3)]
answer1.drop('target', axis = 1).sum(axis=0)
```

Out[26]:
```
sepal length (cm)    196.7
sepal width (cm)     135.0
petal length (cm)     59.5
petal width (cm)       9.8
dtype: float64
```

In [27]:
```
answer2 = iris_dataframe[(iris_dataframe['target'] == 'setosa') | (iris_
dataframe['petal length (cm)'] > 1.3)]
answer2.drop('target', axis = 1).sum(axis=0)
```

Out[27]:
```
sepal length (cm)    876.5
sepal width (cm)     458.6
petal length (cm)    563.7
petal width (cm)     179.9
dtype: float64
```

Question 14

- write a lambda that subtracts 2 from all elements of a pandas dataframe
- display the top 5 rows
- use the same lambda, but applying it to the sepal length column only
- so the output should be the top 5 rows of a dataframe where all cells have had 2 subtracted and then another dataframe where only sepal length has 2 subtracted from it

In [28]:
```
answer = iris_X.copy()
answer.apply(lambda x: x-2).head(5)
```

Out[28]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	3.1	1.5	-0.6	-1.8
1	2.9	1.0	-0.6	-1.8
2	2.7	1.2	-0.7	-1.8
3	2.6	1.1	-0.5	-1.8
4	3.0	1.6	-0.6	-1.8

In [29]:
```
answer['sepal length (cm)'] = answer['sepal length (cm)'].apply(lambda x
: x-2)
answer.head(5)
```

Out[29]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	3.1	3.5	1.4	0.2
1	2.9	3.0	1.4	0.2
2	2.7	3.2	1.3	0.2
3	2.6	3.1	1.5	0.2
4	3.0	3.6	1.4	0.2

Question 15

- normalize each column of pandas dataframe to 0-1 scale.
- hint
  - 0-1 is done by using (X - xmin)/(xmax - xmin)
  - the approach should be similar to when we meean centered the dataframe

In [30]:
```
answer = (iris_X - iris_X.min(axis=0))/(iris_X.max(axis=0) - iris_X.min(
axis=0))
answer.head(5)
```

Out[30]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	0.222222	0.625000	0.067797	0.041667
1	0.166667	0.416667	0.067797	0.041667
2	0.111111	0.500000	0.050847	0.041667
3	0.083333	0.458333	0.084746	0.041667
4	0.194444	0.666667	0.067797	0.041667

Question 16

- assume the below 2 matrices, the first is observations and the second is cluster centers
- using cdist from scipy, create a matrix where the rows represent our 3 observations and the columns represent our 2 clusters and the cells values are the euclidean distances between a given observation and cluster
- when would this be of use?

In [31]:
```
observations = np.array([
    [1,2,3],
    [4,3,1],
    [2,3,4]
])

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

In [32]:
```
from scipy.spatial.distance import cdist
cdist(observations, cluster_centers, 'euclidean')
```

Out[32]:
```
array([[2.44948974, 1.41421356],
       [2.        , 3.46410162],
       [3.        , 2.23606798]])
```

This can be used in k-mean clustering, where we iteratively determine which cluster a point belongs to by calculating its euclidean distance.

In [ ]:
```
```