CSE 163 **Name:** _____
Spring 2019
Exam 2 - Section Practice
6/03/2019 **Student Number:** _____
Time Limit: 50 Minutes

Do not open the exam before the exam begins and close the booklet when time is called. Starting early or working after time is called will lead to a -10 deduction. You may write your name and Net ID on the front of the exam before the exam starts.

This exam contains 9 pages (including this cover page) and 4 questions. Some questions have sub-parts.

You are allowed to have one sheet of paper (both sides) with you as your cheat sheet. All other materials besides writing utensils should be put away before the exam starts. This includes all electronic devices like phones, calculators, and smart watches.

This exam is not, in general, graded on style and you do not need to include comments or imports for your code. Specific questions may specify restrictions about the style of your code that you must follow to receive full credit.

You may not abbreviate any code, such as "ditto" marks or "..." marks. You may write code to the side and indicate where it should be inserted. These markings must be unambiguous and any ambiguity when grading may result in a deduction if your code is not readable. All code and answers should remain within the provided boxes if possible.

You are allowed to ask for scratch paper after the exam starts to use as additional space when writing answers, but you must indicate on the original page for the problem that part of the answer is on scratch paper. Scratch paper must be stapled to the **END** of your exam after you finish the test. Failure to do so may result in your work on scratch paper not being accepted while grading.

Initials: _____

Initial above to indicate you have read and agreed to the rules above.
Failure to initial may result in your exam not being accepted for credit.

Practice Exam Notes

Like before, this practice exam focuses more on covering the breadth of topics you might see on the exam rather than writing a practice that would take students exactly 50 minutes to complete. Some of the response pages may be shorter than they would appear on the actual exam to save space; this means on the real exam, where we will leave plenty of space to write answers, will likely look much longer even though the amount of content is about the same.

1. For this problem, you will be writing a simplified `DataFrame` class to explore one way to implement the pandas functionality. You may not use `pandas` to solve this problem. In the space provided, write a class named `DataFrame` that has the following methods.

   - An initializer that takes the data the `DataFrame` will store in the list of dictionaries format. You may assume each `dict` row has the same set of column names.
   - Write method named `get` that takes two optional parameters for a row index and a column name. You may assume if values are passed for the row or column, that they are valid entries in the `DataFrame`; you do not need to handle slices or list of rows/cols, just single values for each. This method behaves differently depending on which parameters are passed:
     - If neither the row or column are specified, returns `None`
     - If the row is specified but not the column, returns the `dict` row at the given index.
     - If the column is specified but not the row, returns a list of all the values for that column.
     - If both are specified, returns the single value for the given row an column.

   The class should have private fields, but you don't need to worry about returning or storing references to data the client might have a reference to.

Solution:

```
class DataFrame:
    def __init__(self, data):
        self._data = data

    def get(self, row=None, col=None):
        if row is None and col is None:
            return None
        elif row is None:
            result = []
            for d in self._data:
                result.append(d[col])
            return result
        else:
            d = self._data[row]
            if col is None:
                return d
            else:
                return d[col]
```

2. This question has a few short answer questions on miscellaneous topics from the quarter.

   (a) Assume we have the following hash function defined in a `Card` class that represents a card in a deck of cards.

```
def __hash__(self):
    return 0
```

   In the space below, describe whether or not this hash function is functionally correct. If it is not correct, explain what causes it to break a hash table that uses it. If it is correct, explain whether or not this is a good hash function. Make sure you clearly answer the first question in your response.

Solution: This hash function works functionally, but is not a good hash function because it sends all of the objects to the same index in the hash table. There are no external errors in using this function, but it would be slow since we don't get the benefits of the values being spread out.

   (b) Below we have defined a function that may or may not have a bug. In the space on the next page, write "No bug" if there is no bug in the program. If there is a bug, write a short program that constructs a small set of inputs and uses the `assert_equals(expected, received)` we have been using from `cse163_utils`.

```
def contains(data, col, val):
    """
    Takes a list of dictionaries that represents
    tabular data, a column name, and a value and
    returns True if the given value appears anywhere
    in the given column and false otherwise.

    Assumes col is a valid column in the dataset.
    """
    for row in data:
        if row[col] == val:
            return True
        else:
            return False
```

Solution:

```
data = [{'a': 1}, {'a': 2}]
result = contains(data, 'a', 2)
cse163_utils.assert_equals(True, result)
```

(c) Consider the task of using a neural network that takes pixel images of a hand-written lowercase letter (a-z) and predicts which letter was written. The input images are 10 pixels by 10 pixels and there are 26 possible letters.

In the space below, indicate the number of input neurons and output neurons the simple neural network described in class that solves this task.

Input Neurons:

Solution: 100

Output Neurons:

Solution: 26

3. For this problem, assume we have the following datasets. The first is a regular `DataFrame` stored in a variable named `df`, while the geo-spatial dataset stored in a `GeoDataFrame` named `gdf`:

df

city	country	emissions
Seattle	U.S.A	10
Portland	U.S.A	30
Tokyo	Japan	20
Pyrus	Genovia	40

gdf

name	geometry
Japan	Polygon1
South Korea	Polygon2
U.S.A	Polygon3

(a) What is the resulting table when executing the following join?

```
df.merge(gdf, left_on='country', right_on='name',
         how='right')
```

Make sure to include all of the column names.

Solution:

city	country	emissions	name	geometry
Seattle	U.S.A	10	U.S.A	Polygon3
Portland	U.S.A	30	U.S.A	Polygon3
Tokyo	Japan	20	Japan	Polygon1
NaN	NaN	NaN	South Korea	Polygon2

(b) In the space below, write a function called `missing_matches` that takes two parameters, one for each of the datasets described above. The function should return a new `DataFrame` that has all the rows from the datasets that do not have a corresponding row in the other when merging on country and name. For example, if we were to call the function with the datasets described above:

```
missing_matches(df, gdf)
```

It would return a `DataFrame` with the following values:

city	country	emissions	name	geometry
Pyrus	Genovia	40	NaN	NaN
NaN	NaN	NaN	South Korea	Polygon2

Your code should work on any dataset with this column layout. This means you should not assume any specific rows are in the data when writing your solution.

Solution:

```
def missing_matches(df, gdf):
    res = df.merge(gdf, left_on='country', right_on='name',
                   how='outer')
    return res[res['country'].isnull() | res['name'].isnull()]
```

(c) In the space below, write a function that plots a map of the world where each country is colored by the total emissions of cities in that country. Countries that do not have any emissions data should not be plotted. Your plot should include a legend and you should save the plot to a file called `'world_emissions.png'`.

You do not need to write a function, you may assume the variables `df` and `gdf` defined at the beginning of the problem exist, and we have run the standard code to import `matplotlib` as `plt`. You should not assume the data is limited to the rows shown above but you may assume the datasets have the columns described. You may assume operations that combine a `DataFrame` and a `GeoDataFrame` returns a `GeoDataFrame`.

Solution:

```
merged = gdf.merge(df, left_on='name', right_on='country',
                   how='inner')
grouped = merged.dissolve(by='country', aggfunc='sum')
grouped.plot(column='emissions', legend=True)
plt.savefig('world_emissions.png')

# We would aso accept df.merge(gdf, ...) since we said
# you can assume merging operations return GeoDataFrames
```

4. The following problems relate to image processing.

   (a) Assume we have the following three `numpy` arrays defined. We show the name and the shape above each array

`a (4, 3)`

0	0	1
0	1	0
0	1	1
1	0	0

`b (4,)`

1	2	3	4

`c (4, 1)`

1
2
3
4

   In the spaces below, write out the results of the following arithmetic operations as well as writing down the shape of the result. If the result is an error, indicate so and write a sentence or two explaining why it is an error.

   i. `a + a`

Solution: Shape: `(4, 3)`

0	0	2
0	2	0
0	2	2
2	0	0

   ii. `a + b`

Solution: This causes an error because the shapes are not compatible when broadcasting. After padding the shape with a dimension with value 1 and stretching that dimension to have value 4, there will be a mismatch in the second dimension (`a` has 3, `b` has 4) which is an error since neither are 1.

   iii. `a + c`

Solution: Shape: `(4, 3)`

1	1	2
2	3	2
3	4	4
5	4	4

   iv. `c + b`

Solution: Shape: `(4, 4)`

2	3	4	5
3	4	5	6
4	5	6	7
5	6	7	8

(b) Write a function called `stripes` that takes a color image (a `(n, m, 3)` `numpy` array) and returns a new color image that is the result removing a certain color from certain ranges of pixels. The result will be horizontal stripes of differing colors on top of the image.

To do this, we will break the image into 3 pieces by height. The top most section of the image (corresponding to pixels near height 0), should have all red pixels set to 0. The middle third should have all green pixels set to zero. The remaining third should have the blue pixels set to 0. If the image has a height that is not evenly divisible by 3, the leftover pixels can be included in any of the sections that have had their color changed as described above.

Your method should not modify the input array. For full credit, your solution should have no loops that loop over the image.

Solution:

```
def stripes(img):
    height = img.shape[0]
    stripe_height = round(height / 3)

    result = img.copy()
    result[:stripe_height, :, 0] = 0
    result[stripe_height:2*stripe_height, :, 1] = 0
    result[2*stripe_height:, :, 2] = 0

    return result
```

(c) Write a function called `compress` that takes a gray-scale image (`numpy` array with shape `(n, m)`) that compresses 2x2 patches of the image by taking the average of the pixel values. This method should behave differently than the standard convolution code we have seen before in the sense that the patches should not overlap. For example, if we had the following image named `img`:

1	2	3	1
2	1	2	2
2	2	1	1
2	2	1	7

If we call `compress(img)`, it would return a `numpy` array with the following values

1.5	2
2	2.5

Notice that the resulting size is `(2, 2)` because we don't let the patches overlap (resulting in two positions across the width of this example image).

You do not need to worry about casting the values to integers. You may assume the given image is square and has an even width/height. Your method should not modify the given array, but instead should return a new one. For full credit, your solution should only have two loops.

Solution:

```
def compress(img):
    height, width = img.shape
    result = np.zeros((height // 2, width // 2))

    for i in range(result.shape[0]):
        for j in range(result.shape[1]):
            curr = img[2*i:2*i+2, 2*j:2*j+2]
            result[i, j] = np.mean(curr)
    return result
```

- **Built-in Python functions**
  - `print(*strings)`
  - `range(end)` / `range(start, end, step?)`
  - `abs(v)`
  - `min(v1, v2)` / `max(v1, v2)`
  - `sum(v1, v2)`
  - `open(fname)`
  - `zip(l1, l2)`
  - Types: `int(v)`,`float(v)`,`str(v)`,`bool(v)`
- **String methods**
  - `upper()`, `lower()`
  - `find(s)`
  - `strip()`
  - `split()`
- **List methods**
  - Construct: `list()` or `[]`
  - `append(val)`
  - `extend(lst)`
  - `insert(idx, val)`
  - `remove(val)`
  - `pop(idx)`
  - `index(val)`
  - `reverse()`
  - `sort(key?)`
- **Set methods**
  - Construct: `set()`
  - `add(val)`
  - `remove(val)`
- **Dictionary methods**
  - Construct: `dict()` or `{}`
  - `keys()`
  - `values()`
  - `items()`
- **Special object methods**
  - `__init__`
  - `__repr__`
  - `__eq__`
  - `__hash__`
- **Pandas methods**
  - `mean()`
  - `min()` / `max()`
  - `idxmin()` / `idxmax()`
  - `count()`
  - `unique()`
  - `groupby(col)`
  - `apply(fun)`
  - `isnull()` / `notnull()`
  - `dropna()` / `fillna(v)`
  - `sort_values(col)` / `sort_index()`
  - `nlargest(n, col)`
  - `merge(df, left_on, right_on, how)`
- **Pandas fields**
  - `index`
  - `loc[row, col]`
- **Geopandas obnject methods**
  - `plot(column?, legend?, ax?, color?, vmin?, vmax?)`
  - `dissolve(by, aggfunc)`
  - Any of the pandas functions above
- **Geopandas module methods**
  - `geopandas.sjoin(left, right, op, how)`
- **matplotlib model classes**
  - `plt.subplots(nrows, ncols)`
  - `plt.show()`
  - `plt.savefig(f_name)`
- **numpy module methods**
  - `np.array(vals?)`
  - `np.arange(end)` / `np.arange(start, end, step?)`
  - `np.ones(shape)` / `np.zeros(shape)`
  - `np.dot(a1, a2)`
  - `np.sum(a)` / `np.min(a)` / `np.max(a)` / `np.mean(a)`
- **numpy array object methods**
  - `reshape(shape)`
  - `sum()` / `min()` / `max()` / `mean()`
  - `copy()`
- **numpy module methods**
  - `shape`