Data Management and Analysis, Sample Midterm Questions:

(These questions do not indicate the total length / difficulty level of the exam; instead, it shows the types of questions that you'll see… and the reference is not going to be the exact reference used in the exam)

1. Answer the questions about list comprehensions below:

   a. Consider the following variable definitions:

   ```python
   vowels = 'aeiou'
   s = "queueing"
   ```

   Write one line of code to count the number of vowels in \textit{s}. Both variables listed above must be used. You must use a list comprehension somewhere in your solution (you can use other built-in functions / methods in conjunction with your list comprehension to help):

   ________________________________________________________________

   b. Use a list comprehension to find the largest label in the following Series, \textit{s} (again, you can use any other built-in functions / methods in conjunction with your list comprehension to help):

   ```python
   from pandas import Series
   s = Series(
       data=['foo', 'bar', 'baz', 'qux', 'quxx', 'corge'],
       index=['ant', 'bat', 'cat', 'good doge', 'eel', 'fly']
   )
   ```

   ________________________________________________________________

2. Given the following DataFrame, write code to modify it based on the specifications below:

   ```python
   df = DataFrame(
       [['Ca', 'San Diego'],
        ['OR', 'Portland'],
        ['CA', 'Oakland'],
        ['ca', 'Pasadena'],
        ['wA', 'Olympia']
   ],
   columns=['state', 'city']
   )
   ```

   a. Change \textit{df} so that state abbreviations are normalized to uppercase.

   ________________________________________________________________

   b. Modify \textit{df} so that the order of columns is swapped (change to city first… followed by state)

   ________________________________________________________________
Write the output of code below in the space adjacent to the code (error or no output are possible; not all lines have to be filled):

```python
numbers = [1, 2, 3]
def foo():
    numbers = [True, False]
print(numbers)

def outer(x):
    print('stan')
    def inner():
        x()
        print('mable')
        return inner

def f():
    print('dipper')

def add_o(v):
    # type(v) == str checks if v is a string
    s = f'.oO{v}Oo.' if type(v) == str else v
    return s

def fancify_it(plain):
    def fancy(*args):
        modified = [add_o(a) for a in args]
        plain(*modified)
        return fancy

@fancify_it
def wat(a, b):
    print(a)
    print(b)

wat('hi', 1)
```
Imagine that a single pixel in a digital image can be represented by a tuple consisting of 3 integers, each from 0 - 255. The integers can represent red, green, and blue (based on position in the tuple). Imagine further that an image can be composed of a 2-dimensional array of pixels (with each pixel having length 3).

Using this idea of a digital image, create a class, Image, such that it can be initialized by passing in a grid of pixels (again, each pixel has integers for red, green and blue). The instance of this class can be indexed into to retrieve or write values. Furthermore, it can support a method called dim that returns the width and height as a tuple (width first, height next). Here's how the class works:

```python
img = Image([[(255, 255, 255), (0, 0, 0), (100, 200, 0)], [(0, 100, 0), (0, 77, 77), (123, 123, 123)]])
print(img[1, 0])  # prints out (0, 0, 0)
img[2, 1] = (1, 1, 1) # (sets pixel to tuple)
print(img[2, 1])  # prints out (1, 1, 1)
print(img.dim())  # prints out (3, 2)
```

Create a class that would support the behavior above (the bare minimum implementation, without error handling or concern regarding efficiency is adequate).

Hint: In the code, `img[2, 1]`, the what type is 2, 1?
# Given the following data:
```
exam_data = {
    'name':      ['Anastasia', 'Dima', 'Katherine', 'James', 'Emily',
                 'Michael', 'Matthew', 'Laura', 'Kevin', 'Jonas'],
    'score':     [12.5, 9, 16.5, np.nan, 9, 20, 14.5, np.nan, 8, 19],
    'attempts':  [1, 3, 2, 3, 2, 3, 1, 1, 2, 1],
    'qualify':   ['yes', 'no', 'yes', 'no', 'no', 'yes', 'yes', 'no', 'no', 'yes']
}
labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
```

a) Create a dataframe from the dictionary data above, and with index as the variable, labels.

b) Get the first 3 rows of the dataframe

c) Select the ‘name’ and ‘score’ columns from the dataframe

d) Select rows where number of attempts in the examination is greater than 2

e) Select rows where the score is missing i.e. is NaN

f) Select rows where number of attempts in the examination is < 3 and score > 15

g) Change the score in row ‘d’ to 11.5

h) Calculate the mean score amongst all the students

i) Add a new row with index ‘k’ to the dataframe with the following values for each column:
   name: “Sam”, score: 15.5, attempts: 1, qualify: “yes”, label: “k”
   Then, delete the new row that you just created!

j) Sort the DataFrame first by “name” in descending order

k) Modify the values in “qualify” column such that ‘yes’ becomes True and ‘no’ becomes False
6 Consider the data shown in the below table:

<table>
<thead>
<tr>
<th>street_number</th>
<th>street_name</th>
<th>num_bedrooms</th>
<th>owner_occupied</th>
</tr>
</thead>
<tbody>
<tr>
<td>104</td>
<td>PUTNAM</td>
<td>15</td>
<td>Yes</td>
</tr>
<tr>
<td>197</td>
<td>LEXINGTON</td>
<td>3</td>
<td>No</td>
</tr>
<tr>
<td>NaN</td>
<td>LEXINGTON</td>
<td>n/a</td>
<td>No</td>
</tr>
<tr>
<td>201</td>
<td>BERKELEY</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>203</td>
<td>BERKELEY</td>
<td>3</td>
<td>Yes</td>
</tr>
<tr>
<td>207</td>
<td>BERKELEY</td>
<td>NaN</td>
<td>Yes</td>
</tr>
<tr>
<td>NaN</td>
<td>LEXINGTON</td>
<td>2</td>
<td>NaN</td>
</tr>
<tr>
<td>213</td>
<td>TREMONT</td>
<td>-5</td>
<td>Yes</td>
</tr>
<tr>
<td>217</td>
<td>TREMONT</td>
<td>na</td>
<td>Yes</td>
</tr>
</tbody>
</table>

- List 4 issues with the data above
- Describe your approach to "fixing" each issue
- Write out any assumptions you've made when coming up with these issues and their corresponding fixes
- When writing code, refer to the data above as df (think of this variable as a DataFrame containing the data above that already exists prior to writing you code)
7 Using the same date in 6, assume that the data is stored in a table called property
   a. find the average number of rooms for each street in the data set; sort by average number of rooms in decreasing order of properties
   b. only retrieve the street number and number of bedrooms, but restrict results to not owner occupied with more than two bedrooms
## Pandas DataFrame methods (we did not go over most of these, and you won’t need to use the majority of these)

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs()</td>
<td>Return a Series/DataFrame with absolute numeric value of each element.</td>
</tr>
<tr>
<td>add(</td>
<td>Addision of dataframe and other, element-wise (binary operator add).</td>
</tr>
<tr>
<td>all(</td>
<td>Return whether all elements are True, potentially over an axis.</td>
</tr>
<tr>
<td>any()</td>
<td>Return whether any element is True over requested axis.</td>
</tr>
<tr>
<td>append</td>
<td>Append rows of other to the end of this frame, returning a new object.</td>
</tr>
<tr>
<td>apply</td>
<td>Apply a function along an axis of the DataFrame.</td>
</tr>
<tr>
<td>applymap</td>
<td>Apply a function to a Dataframe elementwise.</td>
</tr>
<tr>
<td>clip()</td>
<td>Trim values at input threshold(s).</td>
</tr>
<tr>
<td>clip_lower</td>
<td>Return copy of the input with values below a threshold truncated.</td>
</tr>
<tr>
<td>clip_upper</td>
<td>Return copy of input with values above given value(s) truncated.</td>
</tr>
<tr>
<td>combine()</td>
<td>Add two Dataframe objects and do not propagate NaN values, so if for a (column, time) one frame is missing a value, it will default to the other frame’s value (which might be NaN as well)</td>
</tr>
<tr>
<td>combine_first</td>
<td>Combine two DataFrame objects and default to non-null values in frame calling the method.</td>
</tr>
<tr>
<td>count()</td>
<td>Count non-NA cells for each column or row.</td>
</tr>
<tr>
<td>cummax()</td>
<td>Return cumulative maximum over a DataFrame or Series axis.</td>
</tr>
<tr>
<td>cummin()</td>
<td>Return cumulative minimum over a DataFrame or Series axis.</td>
</tr>
<tr>
<td>cumprod()</td>
<td>Return cumulative product over a DataFrame or Series axis.</td>
</tr>
<tr>
<td>cumsum()</td>
<td>Return cumulative sum over a DataFrame or Series axis.</td>
</tr>
<tr>
<td>describe()</td>
<td>Generates descriptive statistics that summarize the central tendency, dispersion and shape of a dataset’s distribution, excluding NaN values.</td>
</tr>
<tr>
<td>drop()</td>
<td>Drop specified labels from rows or columns.</td>
</tr>
<tr>
<td>drop_duplicates()</td>
<td>Return DataFrame with duplicate rows removed, optionally only considering certain columns</td>
</tr>
<tr>
<td>dropna()</td>
<td>Remove missing values.</td>
</tr>
<tr>
<td>duplicated()</td>
<td>Return boolean Series denoting duplicate rows, optionally only considering certain columns</td>
</tr>
<tr>
<td>equals()</td>
<td>Determines if two NDFrame objects contain the same elements.</td>
</tr>
<tr>
<td>fillna()</td>
<td>Fill NA/NaN values using the specified method.</td>
</tr>
<tr>
<td>filter()</td>
<td>Subset rows or columns of dataframe according to labels in the specified index.</td>
</tr>
<tr>
<td>ge()</td>
<td>Wrapper for flexible comparison methods ge</td>
</tr>
</tbody>
</table>

### Examples

```python
import pandas as pd

# Create a DataFrame
df = pd.DataFrame({'A': [1, 2, 3, 4], 'B': [5, 6, 7, 8], 'C': [9, 10, 11, 12]})

# Accessing methods
df.abs()  # Absolute values
df.cumsum()  # Cumulative sum
df.describe()  # Descriptive statistics
```
True and otherwise are from other.

where(cond[, other, inplace, axis, level, …])

truncate(before, after, axis, copy)

transpose(*args, **kwargs)

transform(tune, *args, **kwargs) Call function producing a like-indexed NDFrame and return a NDFrame with the transformed values

transpose(*args, **kwargs) Transpose index and columns.

truncate(before, after, axis, copy) Truncate a Series or DataFrame before and after some index value.

return an object of same shape as self and whose corresponding entries are from self where cond is True and otherwise are from other.
Solutions

1. a. \( \text{len([letter for letter in s if letter in vowels])} \)
   \( \text{sum([1 for letter in s if letter in vowels])} \)
   \( \text{sum([letter in vowels for letter in s])} \) # tricky solution!
   The last one is a bit strange in that sum's behavior coerces True to 1 and False to 0 (not the expected solution, but added for completeness).

   b. \( \text{max([len(label) for label in s.index])} \)

2. a. \( \text{df['state'] = df['state'].str.upper()} \)
   \( \text{df['state']} = \text{df['state'].apply(lambda state: state.upper())} \)

   b. \( \text{df = df.reindex(columns=['city', 'state'])} \)

   df = df[['city', 'state']]

3. a. [1, 2, 3]
   b. stan
dipper
   mable
c. .OOhOo.

4. class Image:
   def __init__(self, pixels):
       self.pixels = pixels
   def __getitem__(self, t):
       x, y = t
       return self.pixels[y][x]
   def __setitem__(self, t, v):
       x, y = t
       self.pixels[y][x] = v
   def dim(self):
       return len(self.pixels[0]), len(self.pixels)

5. a. \( \text{df = pd.DataFrame(exam_data, index=labels)} \)
   b. \( \text{df.head(3) \# OR df.iloc[:3]} \)
   c. \( \text{df[['name','score']]} \)
   d. \( \text{df[df['attempts']>2]} \)
   e. \( \text{df['score'].isnull() \}} \)
   f. \( \text{df[df['attempts'] < 3][df['score'] > 15]} \)
      # OR \( \text{df[(df['attempts'] < 3) \& (df['score'] > 15)]} \) \( \ldots \) (did not cover, but for completeness)
   g. \( \text{df.loc['d', 'score']} = 11.5 \)
   h. \( \text{df['score'].mean()} \)
   i. \( \text{df.loc['k']} = ['Sam', 15.5, 1, 'yes'] \)
      df = df.drop('k')
   j. \( \text{df.sort_values(by=['name'], ascending=[False])} \)
   k. \( \text{df['qualify']} = \text{df['qualify'].map({'yes': True, 'no': False})} \)

6. Varying solutions

7. a. \( \text{SELECT street_name, AVG(num_bedrooms) FROM property GROUP BY street_name ORDER BY AVG(num_bedrooms) DESC;} \)
   b. \( \text{SELECT street_number, num_bedrooms FROM property WHERE num_bedrooms > 2 \& owner_occupied <> 'Yes';} \)