```
In [1]: NAME = "Dustin Seltz"
```

Purpose:

This file aims to answer Question 4 through simple queries.

This file contains some queries Oleksandra requested. Given that you have completed a certain level from a sequence (ex: "N1" from JLPT, "grade 6" from school, or level "50" from WaniKani) what are the next kanji you should learn (the most frequent in the following level/s)?

Input:

cleaned_link.csv This file contains information for the 2136 Jōyō kanji. This program uses the difficulty levels and basic Jisho frequency information.

Output:

Tells the user which kanjis to learn next. Inline output, no csvs.

```
In [2]: import pandas as pd
        import matplotlib.pyplot as plt
        from numpy import isnan
```

```
In [3]: filename = "../Question1/cleaned_link.csv"
        df = pd.read_csv(filename)
```

In [4]:
```
#We will need to translate between strings representing levels and easy-to-compare
numbers

intIsJustItself = lambda x: x

#Low N# means higher level. Scale of N5 to N1.
JLPT_levels = 5
levelValues = {"N"+str(i): (JLPT_levels+1)-i for i in range(1, JLPT_levels+1)}
levelValues["none"] = 0 #'none' is support for the queries at the end of this file
def translateJLPT(levelStr):
    try:
        return levelValues[levelStr]
    except:
        return float('nan')
print(df["grade"].unique())
gradeLevels = {
                'none': 0, #'none' is support for the queries at the end of this file
                'grade 1': 1,
                'grade 2': 2,
                'grade 3': 3,
                'grade 4': 4,
                'grade 5': 5,
                'grade 6': 6,
                'junior high': 7,
                }
def translateGradeLevel(levelStr):
    try:
        return gradeLevels[levelStr]
    except:
        return float('nan')

possibleGenkiValues = df["Genki_Lesson"].unique()
possibleGenkiValues.sort()
print(possibleGenkiValues)

genki_levels = 0
for value in df["Genki_Lesson"].unique():
    if(not isnan(value)):
        genki_levels += 1
print(genki_levels, "possible valid values.")
#It looks like there is no lesson before lesson 3. A consequence of our source?
#http://genki.japantimes.co.jp/self/genki-kanji-list-linked-to-wwkanji
#We need level numbers to be 1..genki_levels, inclusive
def translateGenki(x):
    try:
        return x-2
    except:
        return float('nan')
```

```
['junior high' 'grade 4' 'grade 3' 'grade 5' 'grade 6' 'grade 1' 'grade 2'
 nan]
[ 3.  4.  5.  6.  7.  8.  9. 10. 11. 12. 13. 14. 15. 16. 17. 18. 19. 20.
 21. 22. 23. nan]
21 possible valid values.
```

In [5]:
```
#Sequence is what you're learning, like jlpt or Genki_Lessons or wanikani_Level
#Translator translates that sequence's levels like "N1" into number ranks.
#numberToGet is how many kanji from the next level to return
def getNextInSquence(sequence, level, translator, numberToGet):
    #Frequency is the frequency from Jisho
    col = zip(df["kanji"], df[sequence], df["frequency"])

    colAboveLevel = []
    previouslyCompletedLevel = translator(level)
    #The column is now created by index, and the index is from Joyo.
    for (kanji, levelInSequence, freq) in col:
        numericLevelInSequence = translator(levelInSequence)
        #If you wanted to retrieve data from the current as well as any higher leve
l, just make this >=
        if(numericLevelInSequence > previouslyCompletedLevel):
            colAboveLevel.append((kanji, levelInSequence, numericLevelInSequence, f
req))
    #Now we sort by the numeric level (Can't sort by string level, N1 is harder tha
n N2),
    #    so that we give them the next hardest kanji instead of any harder kanji.
    colAboveLevel.sort(key=lambda tup: tup[2])

    if(len(colAboveLevel) == 0 ):
        return []

    for index in range(len(colAboveLevel)):
        #min(len(colAboveLevel), bounds[nextBound]),
        (_,_, numericLevelInSequenceCurrent, freqCurrent) = colAboveLevel[index]
        #Frequency is a rank so lower is more frequent
        indexLeft = index
        #Debug, see how many swaps this kanji has done
        num = 0
        while indexLeft > 0:
            indexLeft -= 1
            (_,_, numericLevelInSequenceLeft, freqLeft) = colAboveLevel[indexLeft]
            #We're sorted by level so if our levels are unequal then we are done (o
nly sorting within levels)
            #I consider NaN frequencies to be high frequency rank (infrequent),
            #    since it'd make sense for a more obscure kanji to not have a frequ
ency rating
            if((freqLeft > freqCurrent or isnan(freqLeft)) and numericLevelInSequen
ceCurrent == numericLevelInSequenceLeft):
                #Swap
                #print("Swapping", colAboveLevel[indexLeft], colAboveLevel[index],
"b/c", freqLeft, ">",freqCurrent,"num=",num)
                num += 1
                tmp = colAboveLevel[index]
                colAboveLevel[index] = colAboveLevel[indexLeft]
                colAboveLevel[indexLeft] = tmp
                index -= 1
                #print(colAboveLevel[:5])
            else:
                #print("not swapping ",colAboveLevel[indexLeft], colAboveLevel[inde
x])
                break
    #print("returning ",numberToGet,"of", len(colAboveLevel))
    return colAboveLevel[:numberToGet]
```

In [6]:
```
def getMoreWani(level, numberToGet):
    #Actual col name in the dataframe
    sequence = "wanikani_level"
    #Values in the dataframe may not be ints, so the translator is required.
    #But for WaniKani, it is just ints
    translator = intIsJustItself
    return getNextInSquence(sequence, level, translator, numberToGet)
def getMoreGrade(level, numberToGet):
    #Actual col name in the dataframe
    sequence = "grade"
    #Values in the dataframe may not be ints, so the translator is required.
    translator = translateGradeLevel
    return getNextInSquence(sequence, level, translator, numberToGet)
def getMoreJLPT(level, numberToGet):
    #Actual col name in the dataframe
    sequence = "jlpt"
    #Values in the dataframe may not be ints, so the translator is required.
    translator = translateJLPT
    return getNextInSquence(sequence, level, translator, numberToGet)
def getMoreGenki(level, numberToGet):
    #Actual col name in the dataframe
    sequence = "Genki_Lesson"
    #Values in the dataframe may not be ints, so the translator is required.
    translator = translateGenki
    return getNextInSquence(sequence, level, translator, numberToGet)

#Takes the name of the sequence you're learning, the level you last completed, and
how many kanji you want to get.
#Returns a tuple: (kanji, level string, level number)
def getMore(sequenceName, lastCompletedLevel, numberToGet):
    if(numberToGet <= 0):
        return []
    sequenceName = sequenceName.lower()
    if(sequenceName == "wanikani"):
        return getMoreWani(lastCompletedLevel, numberToGet)
    if(sequenceName == "grade" or sequenceName == "grade level"):
        return getMoreGrade(lastCompletedLevel, numberToGet)
    if(sequenceName == "jlpt"):
        return getMoreJLPT(lastCompletedLevel, numberToGet)
    if(sequenceName == "genki"):
        return getMoreGenki(lastCompletedLevel, numberToGet)
    raise ValueError('No sequence found with name '+sequenceName)
```

For all of these queries you are saying the level you last completed, and getting stuff from the next level.

The results are currently sorted by both level and frequency within each level (if the next level contains 40 kanjis and you request 50, you'll get 40 from the next level and 10 most frequent from the level above that).

To retrieve kanji starting at the first level, for the previously completed level pass in 0 for numeric levels, or the string "none" for string-based levels.

Giving the final level or invalid input, the result should be empty.

Results are a list, where each element is the tuple (kanji, level, numericLevel, frequency)

In [7]:
```
mostRecentlyCompletedLevel = 50
numberToGet = 5
print(getMore("WaniKani", mostRecentlyCompletedLevel, numberToGet))
#Should give results from the first level since this is from before the first level
print(getMore("WaniKani", 0, numberToGet))
#Should give nothing
print(getMore("WaniKani", 60, numberToGet))
```

```
[('塚', 51.0, 51.0, 869.0), ('邸', 51.0, 51.0, 905.0), ('郡', 51.0, 51.0, 965.0),
('浦', 51.0, 51.0, 977.0), ('釈', 51.0, 51.0, 1097.0)]
[('一', 1.0, 1.0, 2.0), ('人', 1.0, 1.0, 5.0), ('大', 1.0, 1.0, 7.0), ('十', 1.0,
1.0, 8.0), ('二', 1.0, 1.0, 9.0)]
[]
```

In [8]:
```
mostRecentlyCompletedLevel = "grade 6"
print(getMore("grade", mostRecentlyCompletedLevel, numberToGet))
#Additional tests.
print(getMore("grade", "none", numberToGet))
print(getMore("grade", "junior high", numberToGet))
```

```
[('歳', 'junior high', 7, 269.0), ('企', 'junior high', 7, 278.0), ('藤', 'junior
high', 7, 291.0), ('沢', 'junior high', 7, 296.0), ('与', 'junior high', 7, 308.
0)]
[('日', 'grade 1', 1, 1.0), ('一', 'grade 1', 1, 2.0), ('人', 'grade 1', 1, 5.0),
('年', 'grade 1', 1, 6.0), ('大', 'grade 1', 1, 7.0)]
[]
```

In [9]:
```
mostRecentlyCompletedLevel = "N3"
print(getMore("jlpt", mostRecentlyCompletedLevel, numberToGet))
#Additional tests.
print(getMore("jlpt", "none", numberToGet))
print(getMore("jlpt", "N1", numberToGet))
```

```
[('党', 'N2', 4, 39.0), ('協', 'N2', 4, 121.0), ('総', 'N2', 4, 129.0), ('区', 'N2
', 4, 137.0), ('領', 'N2', 4, 138.0)]
[('日', 'N5', 1, 1.0), ('一', 'N5', 1, 2.0), ('国', 'N5', 1, 3.0), ('人', 'N5', 1,
5.0), ('年', 'N5', 1, 6.0)]
[]
```

In [10]:
```
mostRecentlyCompletedLevel = 10
print(getMore("genki", mostRecentlyCompletedLevel, numberToGet))
#Additional tests.
#The level information we have starts at 3, so 2 or lower should be invalid and giv
e the fist stuff (3).
print(getMore("genki", 1, numberToGet))
print(getMore("genki", 24, numberToGet))
```

```
[('市', 11.0, 9.0, 42.0), ('手', 11.0, 9.0, 60.0), ('明', 11.0, 9.0, 67.0), ('強',
11.0, 9.0, 112.0), ('院', 11.0, 9.0, 150.0)]
[('一', 3.0, 1.0, 2.0), ('十', 3.0, 1.0, 8.0), ('二', 3.0, 1.0, 9.0), ('三', 3.0,
1.0, 14.0), ('時', 3.0, 1.0, 16.0)]
[]
```

In [11]:
```
#Tests for invalid input

#Test with a nonexistant sequence, raises exception
#print(getMore("badvalue", 1, numberToGet))
#Test with a nonexistant numeric level. Just gives first level if too small, empty
[] if too large
#print(getMore("genki", -1, numberToGet))
#Test with a nonexistant string level. Just gives []
#print(getMore("jlpt", "level name", numberToGet))
#Test with invalid numberToGet, gives []
#print(getMore("genki", 1, -5))
```

In [12]:
```
#This question also was originally written to ask for vocab.
#Questions 1 & 2 contain information about readings and words with a given kanji,
#    so whatever information is desired can be retrieved from there after using the
se queries.
```