NLCD Plot All Land Classes Data

December 8, 2020

1 Analyze All Land Classes

1.1 Initialize variables

[1]:
```
# Import modules
import os
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
import numpy as np
from matplotlib import gridspec

# User initialized variables
# Excel input locations
inputFolder = 'C:/Users/dwfir/OneDrive/4E Analytics/Projects/App Voices - Nat␣
 ↪Geo Grant/GIS/'
inputFileName = 'NLCD All Land Classes␣
 ↪20201130-ARM_postIntersection_With_mine_selection.xlsx'
yearsSheetName = 'Land Classes'
stackedSheetName = 'Stacked Land Classes'

# Land Class class range
landClassDict = {
    'Open Water': 11,
    'Developed, Open Space': 21,
    'Developed, Low Intensity': 22,
    'Developed, Medium Intensity': 23,
    'Developed, High Intensity': 24,
    'Barren': 31,
    'Forest Deciduous': 41,
    'Forest Evergreen': 42,
    'Forest Mixed': 43,
    'Shrub': 52,
    'Grasslands': 71,
    'Pasture/Hay': 81,
    'Cultivated Crops': 82,
    'Woody Wetlands': 95,
```

```
    'Emergent Herbaceous Wetlands': 95
}

categoryDict = {
    'Barren': 1,
    'Grassland': 2,
    'Farmland': 3,
    'Shrub': 4,
    'Forest': 5
}

areaColumn = {'Name':'sum', 'Units': 'Square km', 'Factor': 1e6}

# Lower limit defining the high fraction cut for a land class.
lowerLimit = 0.9
```

1.2 Read in data from spreadsheet into dataframes and create the High Fraction dataframe.

[2]:
```
# Read in the excel spreadsheets initialize dataframes
stacked_df = pd.read_excel(inputFolder + inputFileName,␣
 ↪sheet_name=stackedSheetName)
years_df = pd.read_excel(inputFolder + inputFileName, sheet_name=yearsSheetName

# Create a list of the feature counts by land class.
# Create a dataframe of the high fraction, between the lower limit and 1 for␣
 ↪each land class.
# The land class fractions will be stacked in a single column named fraction␣
 ↪other fields are retaine for each record.
countList = []
highFrac_df = pd.DataFrame(columns=['land class','fraction'])
for landClass in landClassDict.values():
    tmp_df = stacked_df[stacked_df[str(landClass)+'_fraction'] >=␣
 ↪lowerLimit][[str(landClass)+'_fraction','arm','NDVI','year']]
    tmp_df = tmp_df.rename(columns={str(landClass)+'_fraction': 'fraction'})
    countList = countList + [len(tmp_df)]
    if len(tmp_df) > 0:
        tmp_df['land class'] = landClass
        highFrac_df = highFrac_df.append(tmp_df)
```

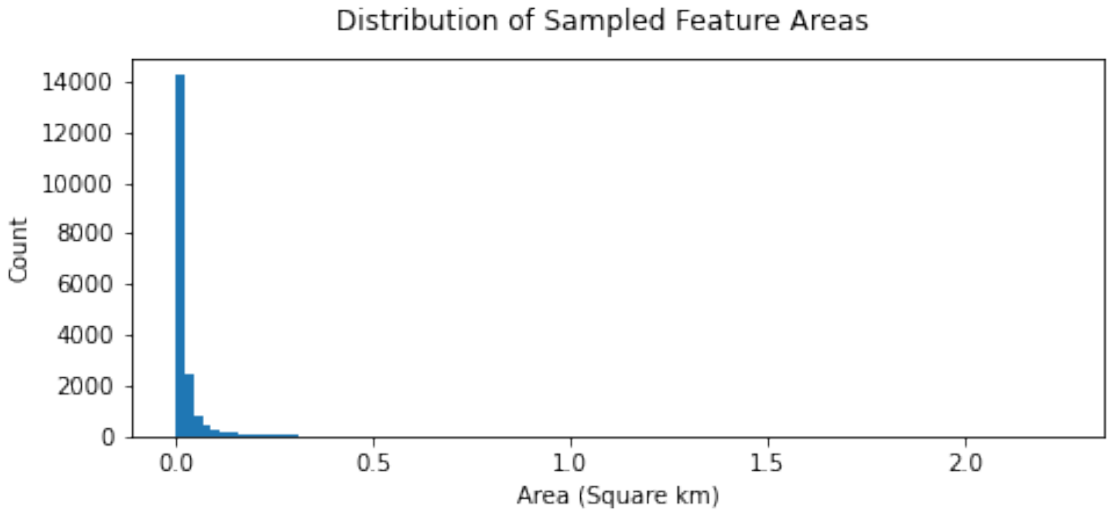
1.3 Counts for high fraction features by land class

[3]:
```
print(f'Land Class - Code - Count')
for item in landClassDict.keys():
    print(f'{item} - {str(landClassDict.get(item))} -␣
 ↪{countList[list(landClassDict.keys()).index(item)]}')
```

```
Land Class - Code - Count
Open Water - 11 - 7
Developed, Open Space - 21 - 943
Developed, Low Intensity - 22 - 334
Developed, Medium Intensity - 23 - 152
Developed, High Intensity - 24 - 56
Barren - 31 - 6666
Forest Deciduous - 41 - 34120
Forest Evergreen - 42 - 209
Forest Mixed - 43 - 1148
Shrub - 52 - 2057
Grasslands - 71 - 13056
Pasture/Hay - 81 - 565
Cultivated Crops - 82 - 0
Woody Wetlands - 95 - 0
Emergent Herbaceous Wetlands - 95 - 0
```

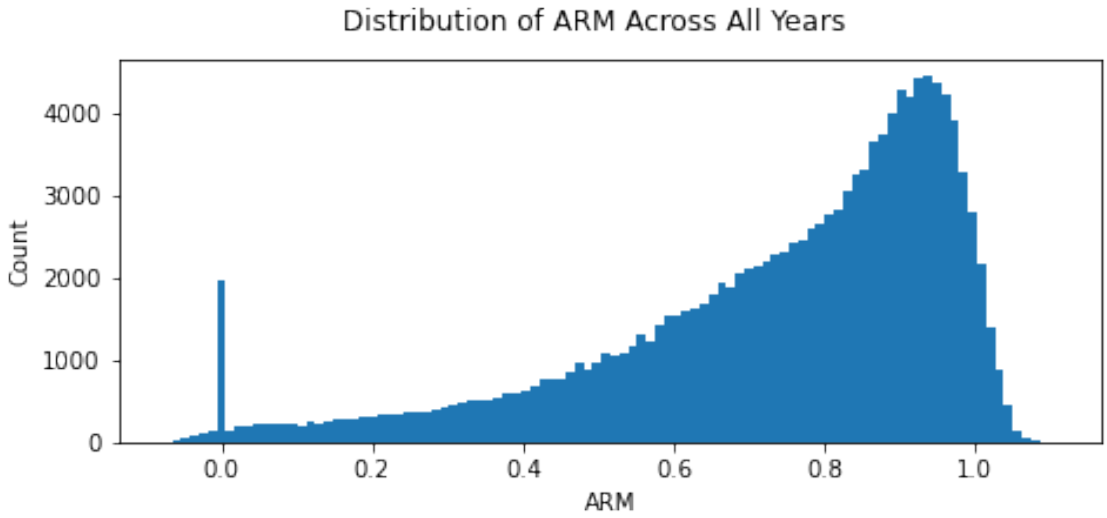
1.4 Distribution of areas across all features

[28]:
```
areaHist, areaAxes = plt.subplots(1, figsize=(7.5,3))
areaHist.suptitle(f'Distribution of Sampled Feature Areas')
areaAxes.hist(years_df[areaColumn.get('Name')]/areaColumn.get('Factor'),␣
 ↪bins=100)
units = areaColumn.get('Units')
areaAxes.set_xlabel(f'Area ({units})')
areaAxes.set_ylabel('Count')

plt.show()
```



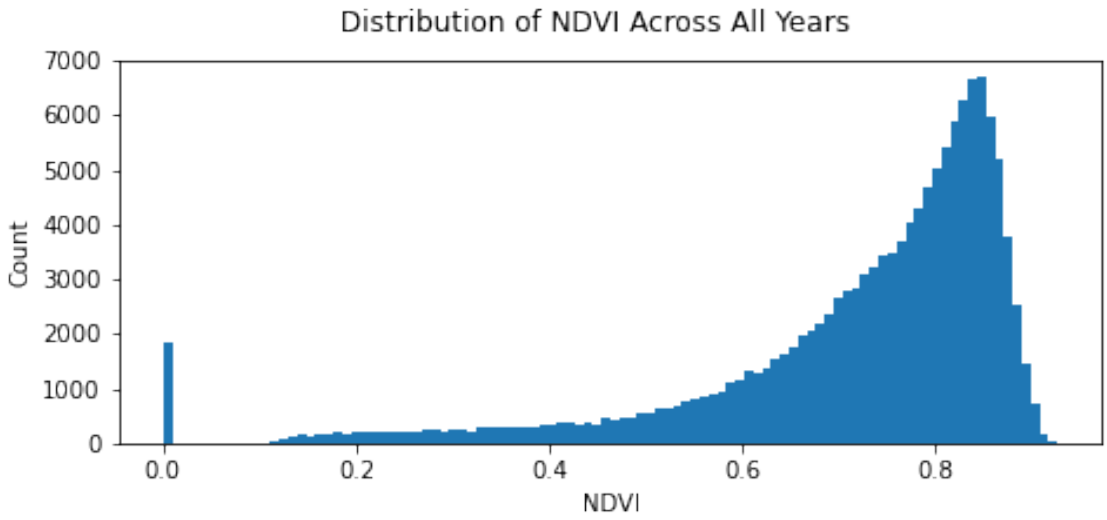
1.5 Distribution of ARM across all featues

[5]:
```
armHist, armAxes = plt.subplots(1, figsize=(7.5,3))
armHist.suptitle(f'Distribution of ARM Across All Years')
armAxes.hist(stacked_df['arm'], bins=100)
armAxes.set_xlabel('ARM')
armAxes.set_ylabel('Count')

plt.show()
```



1.6 Distribution of NDVI across all features

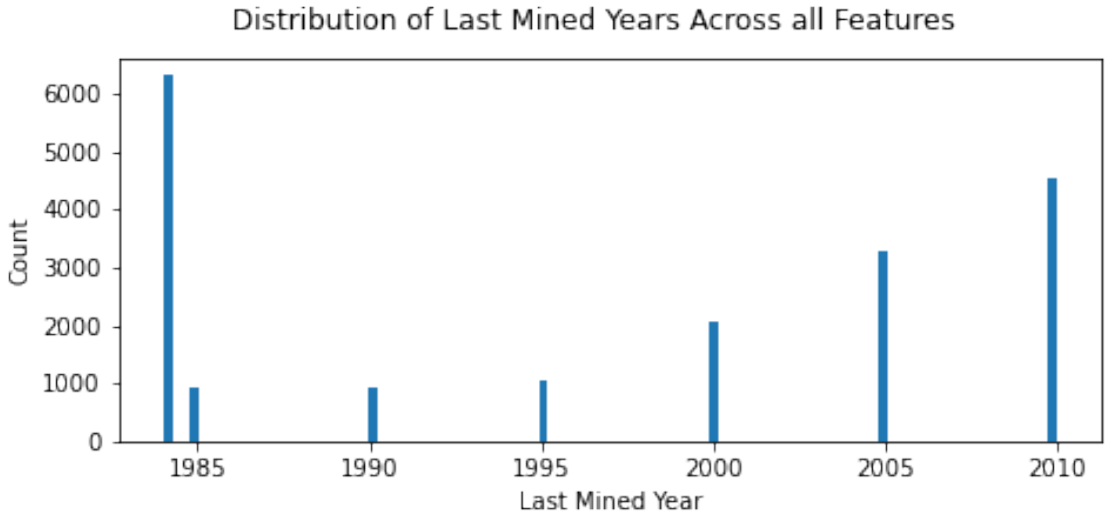
[29]:
```
ndviHist, ndviAxes = plt.subplots(1, figsize=(7.5,3))
ndviHist.suptitle(f'Distribution of NDVI Across All Years')
ndviAxes.hist(stacked_df['NDVI'], bins=100)
ndviAxes.set_xlabel('NDVI')
ndviAxes.set_ylabel('Count')

plt.show()
```



1.7 Distribution of Last Mined year across all features

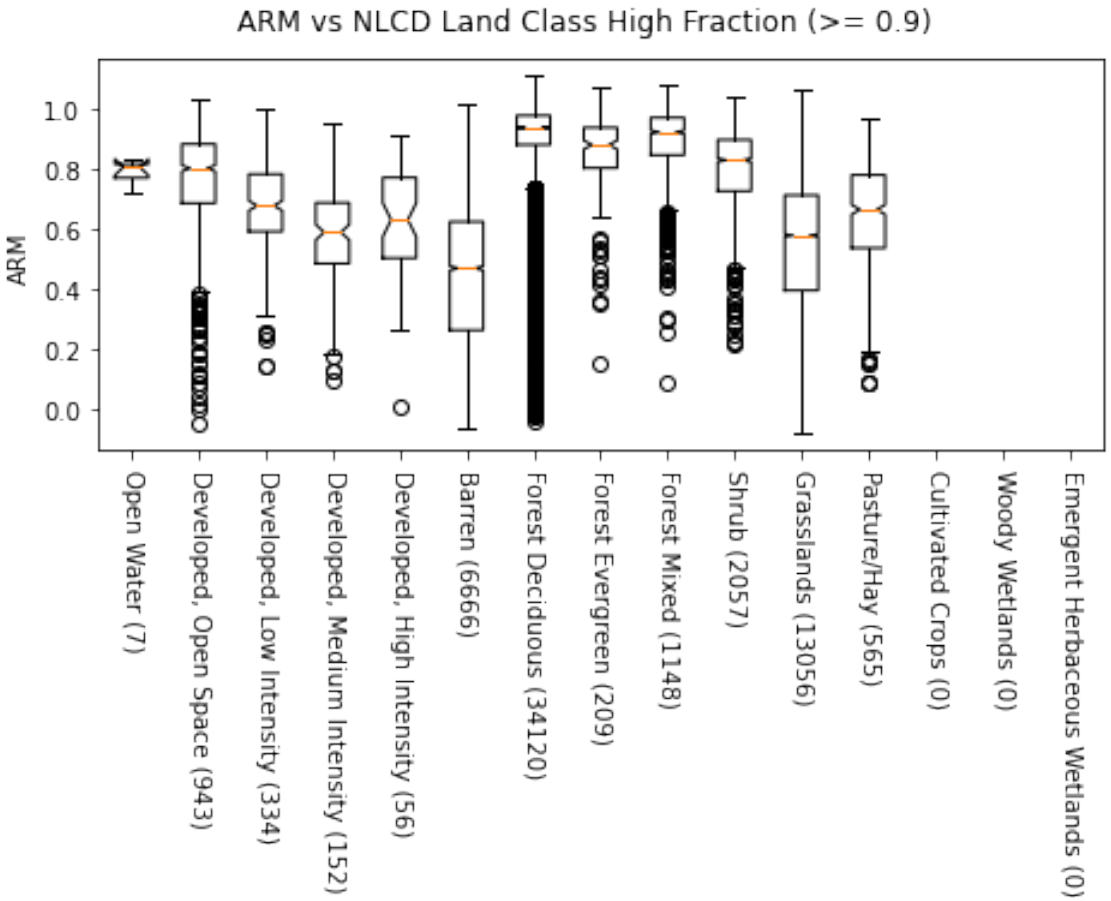
[7]:
```
lastMinedHist, lastMinedAxes = plt.subplots(1, figsize=(7.5,3))
lastMinedHist.suptitle(f'Distribution of Last Mined Years Across all Features')
lastMinedAxes.hist(years_df['lastMined'], bins=100)
lastMinedAxes.set_xlabel('Last Mined Year')
lastMinedAxes.set_ylabel('Count')
```

[7]:
```
Text(0, 0.5, 'Count')
```



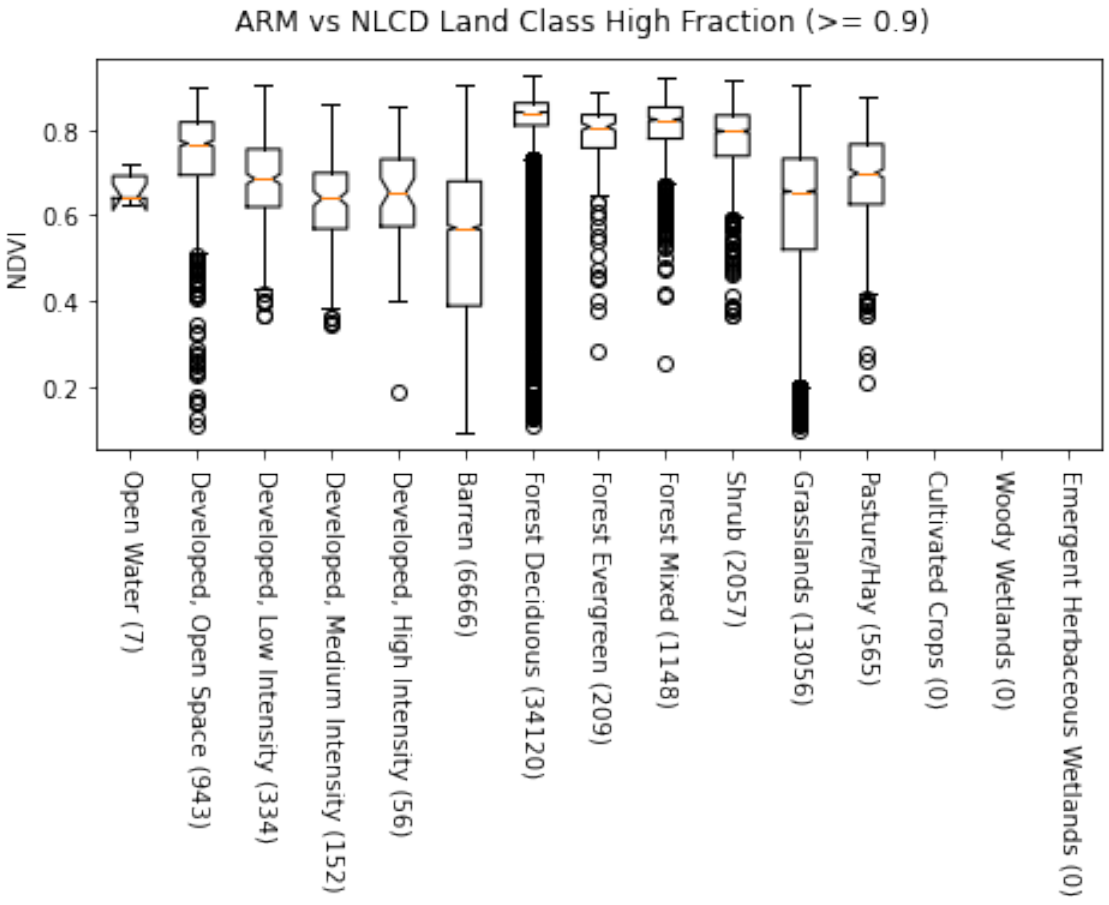
1.8 Box plots of ARM by land class across all years for high fraction features

[8]:
```
#Extract arm values into lists
landClassList = []
for landClass in landClassDict.values():
    landClassList.append(list(highFrac_df[highFrac_df['land␣
 ↪class']==landClass]['arm']))

highFracBox, boxAxes = plt.subplots(1,figsize=(7.5,3))
highFracBox.suptitle(f'ARM vs NLCD Land Class High Fraction (>= 0.9)')
boxAxes.boxplot(landClassList,notch=True)

xlabels = []
for item in landClassDict.keys():
    xlabels = xlabels + [f'{item} ({countList[list(landClassDict.keys()).
 ↪index(item)]})']
boxAxes.xaxis.set_major_formatter(ticker.FixedFormatter(xlabels))
boxAxes.set_ylabel('ARM')
plt.xticks(rotation=-90)
plt.show()
```



1.9 Box plots of NDVI by land class across all years for high fraction features

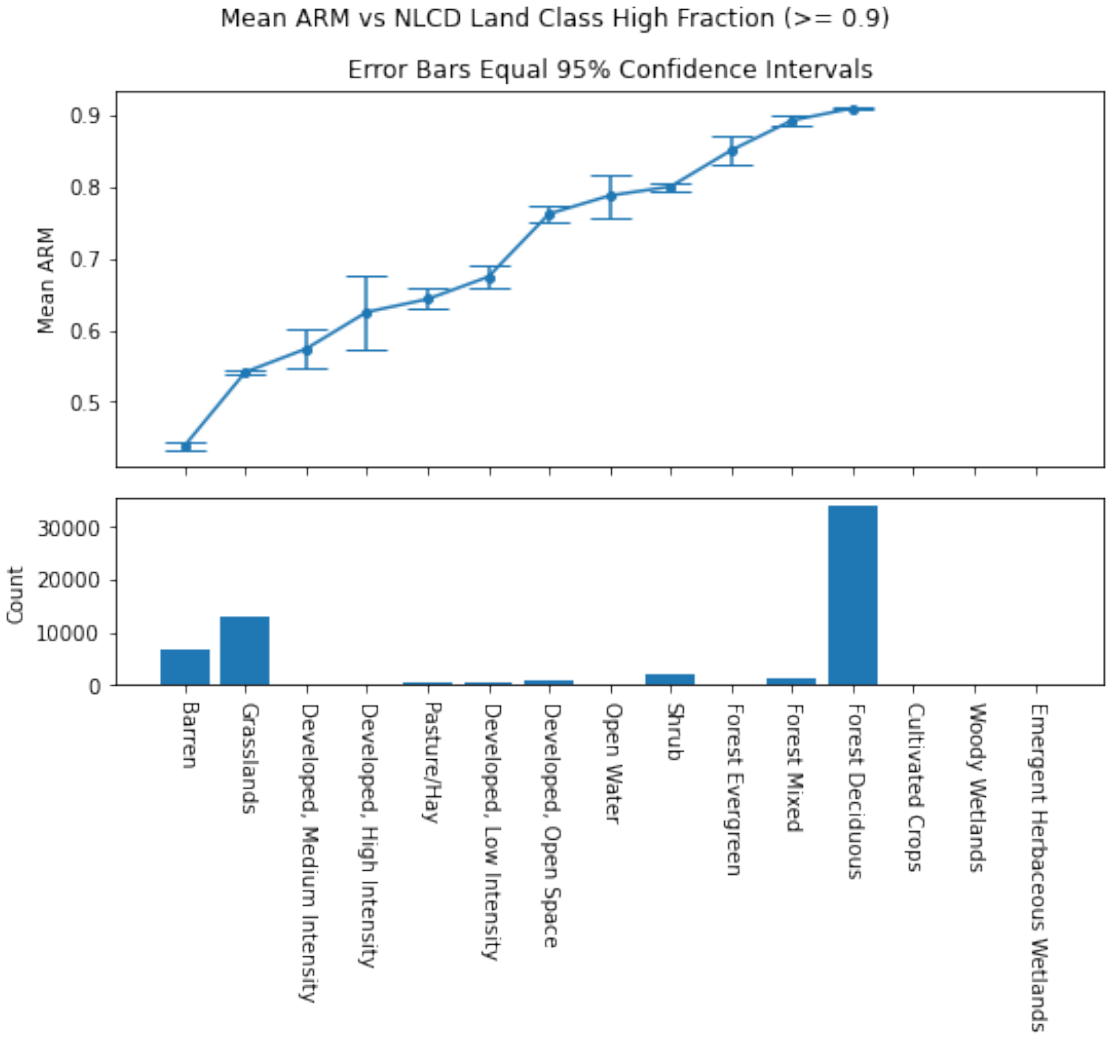
[9]:
```
#Extract arm values into lists
landClassList = []
for landClass in landClassDict.values():
    landClassList.append(list(highFrac_df[highFrac_df['land␣
 ↪class']==landClass]['NDVI']))

highFracBox, boxAxes = plt.subplots(1,figsize=(7.5,3))
highFracBox.suptitle(f'ARM vs NLCD Land Class High Fraction (>= 0.9)')
boxAxes.boxplot(landClassList,notch=True)

xlabels = []
for item in landClassDict.keys():
    xlabels = xlabels + [f'{item} ({countList[list(landClassDict.keys()).
 ↪index(item)]})']
boxAxes.xaxis.set_major_formatter(ticker.FixedFormatter(xlabels))
boxAxes.set_ylabel('NDVI')
plt.xticks(rotation=-90)
plt.show()
```



1.10 ARM means by land class across all years for high fraction features

[53]:
```
meanList = []
medianList = []
stdevList = []
ciList = []
for item in landClassDict.values():
    meanList = meanList + [highFrac_df[highFrac_df['land class']==item]['arm'].
 ↪mean()]
    medianList = medianList + [highFrac_df[highFrac_df['land␣
 ↪class']==item]['arm'].median()]
    if len(highFrac_df[highFrac_df['land class']==item]) != 0:
        stdev = highFrac_df[highFrac_df['land class']==item]['arm'].std()
        stdevList = stdevList + [stdev]
        ciList = ciList + [(stdev)*(1.96/np.sqrt(countList[list(landClassDict.
 ↪values()).index(item)]))]
    else:
        stdevList = stdevList + [np.nan]
```

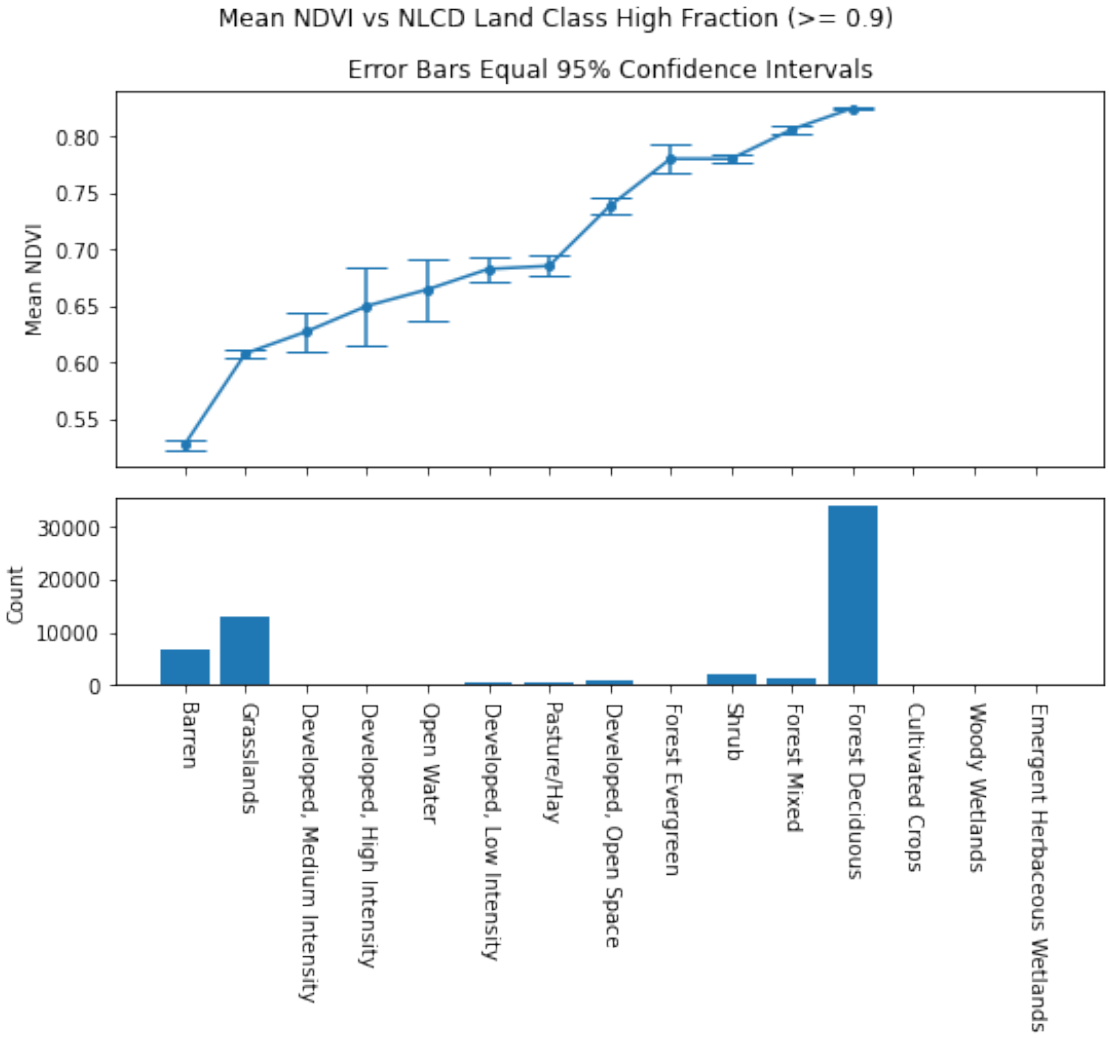
```
        ciList = ciList + [np.nan]

# Create dataframe from lists then sort
sorted_df = pd.DataFrame()
sorted_df['mean'] = meanList
sorted_df['median'] = medianList
sorted_df['stdev'] = stdevList
sorted_df['ci'] = ciList
sorted_df['labels'] = list(landClassDict.keys())
sorted_df['codes'] = list(landClassDict.values())
sorted_df['counts'] = countList
sorted_df = sorted_df.sort_values('mean',ascending=True)

xvalues = np.arange(1,len(meanList)+1)
meanFig, meanAxes = plt.subplots(2, gridspec_kw={
                            'height_ratios': [2, 1]},
                        figsize=(7.5,7),constrained_layout=True,sharex=True)
meanFig.suptitle(f'Mean ARM vs NLCD Land Class High Fraction (>= 0.9)')

meanAxes[0].set_title(f'Error Bars Equal 95% Confidence Intervals')
#meanAxes[0].plot(xvalues,sorted_df['median'],marker='o',markersize=7)
meanAxes[0].set_ylabel('Mean ARM')
meanAxes[0].
 ↪errorbar(xvalues,sorted_df['mean'],marker='o',markersize=4,yerr=sorted_df['ci'],capsize=10)
xlabels = list(sorted_df['labels'])
meanAxes[1].bar(xvalues,sorted_df['counts'])
meanAxes[1].xaxis.set_major_locator(ticker.FixedLocator(xvalues))
meanAxes[1].xaxis.set_major_formatter(ticker.FixedFormatter(xlabels))
meanAxes[1].set_ylabel('Count')
plt.xticks(rotation=-90)
plt.show()
```



1.11 NDVI means by land class across all years for high fraction features

[54]:
```
meanList = []
stdevList = []
ciList = []
for item in landClassDict.values():
    meanList = meanList + [highFrac_df[highFrac_df['land class']==item]['NDVI'].
 ↪mean()]
    if len(highFrac_df[highFrac_df['land class']==item]) != 0:
        stdev = highFrac_df[highFrac_df['land class']==item]['NDVI'].std()
        stdevList = stdevList + [stdev]
        ciList = ciList + [(stdev)*(1.96/np.sqrt(countList[list(landClassDict.
 ↪values()).index(item)]))]
    else:
```

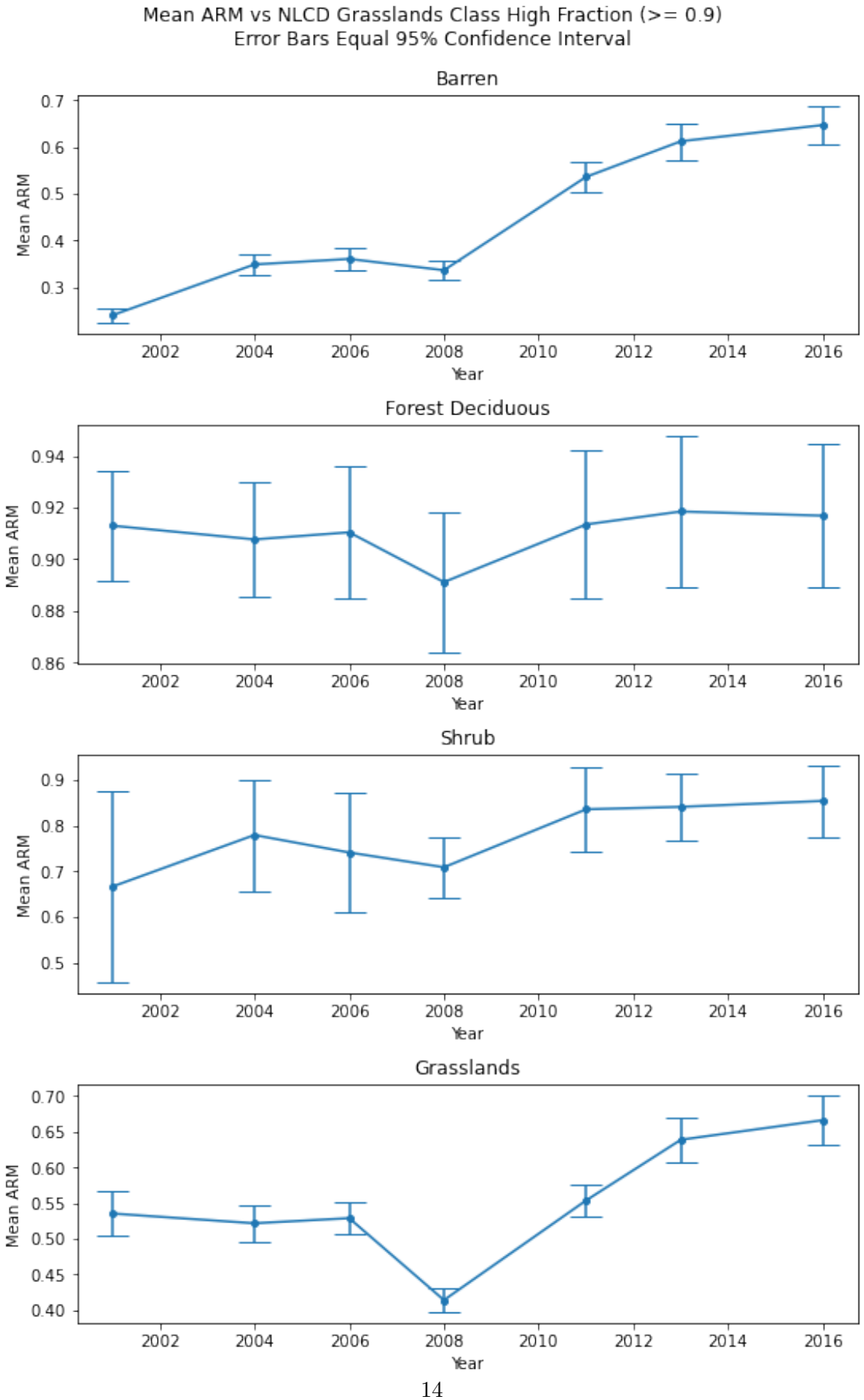
```
        stdevList = stdevList + [np.nan]
        ciList = ciList + [np.nan]

# Create dataframe from lists then sort
sorted_df = pd.DataFrame()
sorted_df['mean'] = meanList
sorted_df['stdev'] = stdevList
sorted_df['ci'] = ciList
sorted_df['labels'] = list(landClassDict.keys())
sorted_df['codes'] = list(landClassDict.values())
sorted_df['counts'] = countList
sorted_df = sorted_df.sort_values('mean',ascending=True)

xvalues = np.arange(1,len(meanList)+1)
meanFig, meanAxes = plt.subplots(2, gridspec_kw={
                            'height_ratios': [2, 1]},
                        figsize=(7.5,7),constrained_layout=True,sharex=True)
meanFig.suptitle(f'Mean NDVI vs NLCD Land Class High Fraction (>= 0.9)')

meanAxes[0].set_title(f'Error Bars Equal 95% Confidence Intervals')
meanAxes[0].set_ylabel('Mean NDVI')
meanAxes[0].
 ↪errorbar(xvalues,sorted_df['mean'],marker='o',markersize=4,yerr=sorted_df['ci'],capsize=10)
xlabels = list(sorted_df['labels'])
meanAxes[1].bar(xvalues,sorted_df['counts'])
meanAxes[1].xaxis.set_major_locator(ticker.FixedLocator(xvalues))
meanAxes[1].xaxis.set_major_formatter(ticker.FixedFormatter(xlabels))
meanAxes[1].set_ylabel('Count')
plt.xticks(rotation=-90)
plt.show()
```



1.12 ARM Means for select land classes by year for high fraction features

[56]:
```
#errorType = 'stdev'
errorType = 'ci'

landClassNumber = [31,41,52,71]

meanFig, meanAxes = plt.subplots(len(landClassNumber),figsize=(7.
↪5,len(landClassNumber*3)),constrained_layout=True)
if errorType == 'stdev':
    meanFig.suptitle(f'Mean ARM vs NLCD {landClass} Class High Fraction (>= 0.
↪9)\nError Bars Equal Standard Deviation')
else:
```

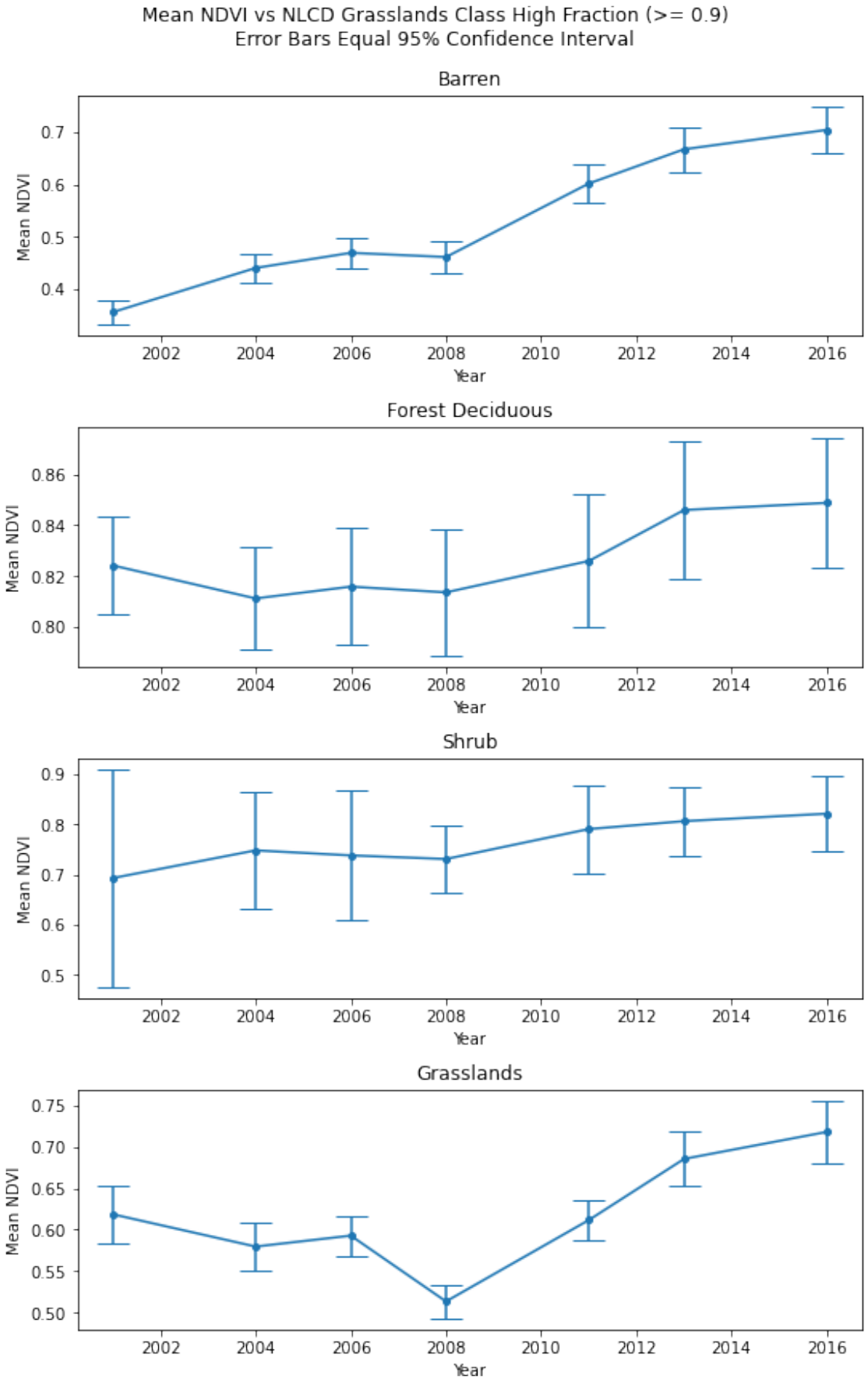
```
    meanFig.suptitle(f'Mean ARM vs NLCD {landClass} Class High Fraction (>= 0.
 ↪9)\nError Bars Equal 95% Confidence Interval')


axis = 0
for landClassItem in landClassNumber:
    landClass = list(landClassDict.keys())[list(landClassDict.values()).
 ↪index(landClassItem)]

    landClassYearMean = highFrac_df[highFrac_df['land class']==landClassItem].
 ↪groupby(['year']).mean().reset_index()
    landClassYearStdev = highFrac_df[highFrac_df['land class']==landClassItem].
 ↪groupby(['year']).std().reset_index()
    landClassYearCount = highFrac_df[highFrac_df['land class']==landClassItem].
 ↪groupby(['year']).count().reset_index()

    ciList = []
    for i in range(0,len(landClassYearCount)):
        ciList = ciList + [(landClassYearMean.iloc[i]['arm'])*(1.96/np.
 ↪sqrt(landClassYearCount.iloc[i]['arm']))]

    meanAxes[axis].set_title(f'{landClass}')

    if errorType == 'stdev':
        meanAxes[axis].
 ↪errorbar(landClassYearMean['year'],landClassYearMean['arm'],marker='o',markersize=4,yerr=la
    else:
        meanAxes[axis].
 ↪errorbar(landClassYearMean['year'],landClassYearMean['arm'],marker='o',markersize=4,yerr=ci

    meanAxes[axis].set_ylabel('Mean ARM')
    meanAxes[axis].set_xlabel('Year')

    axis = axis + 1

plt.show()
```


Mean ARM vs NLCD Grasslands Class High Fraction (>= 0.9)
Error Bars Equal 95% Confidence Interval
Barren
Forest Deciduous
Shrub
Grasslands
Mean ARM
Year
2002
2004
2006
2008
2010
2012
2014
2016

1.13 NDVI means for select land classes by year for high fraction features

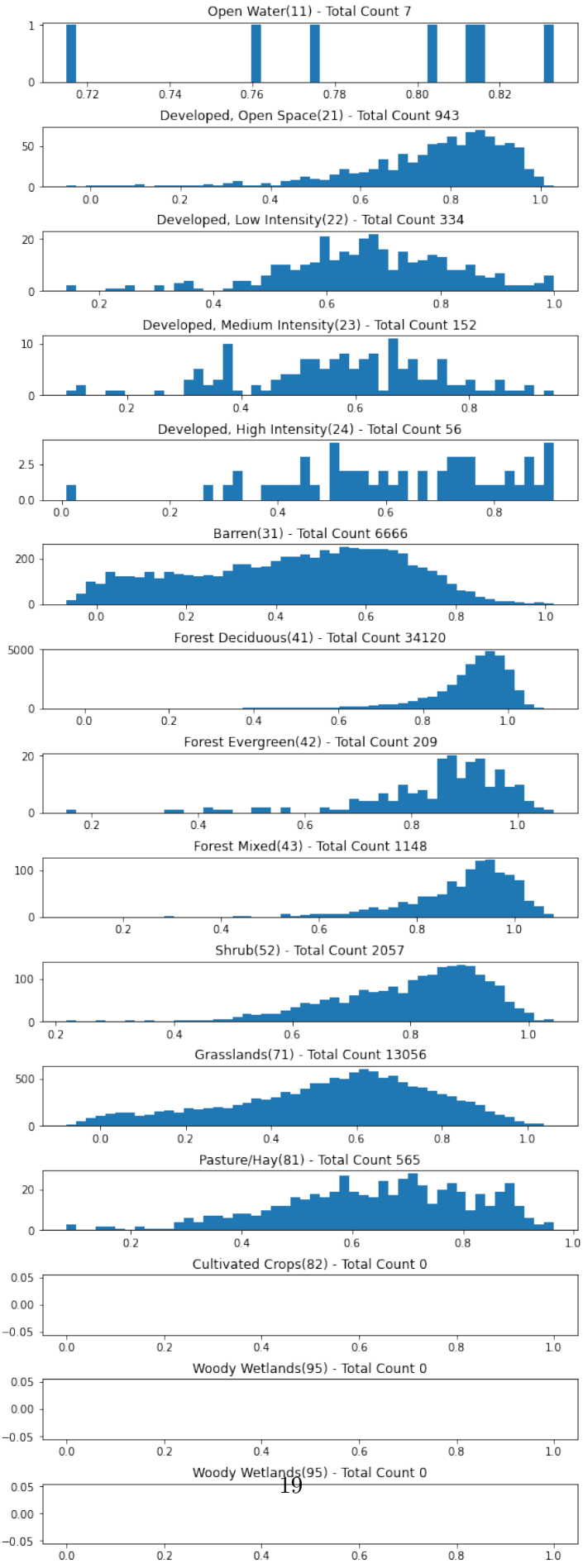
[57]:
```
#errorType = 'stdev'
errorType = 'ci'

landClassNumber = [31,41,52,71]

meanFig, meanAxes = plt.subplots(len(landClassNumber),figsize=(7.
 ↪5,len(landClassNumber*3)),constrained_layout=True)
if errorType == 'stdev':
    meanFig.suptitle(f'Mean NDVI vs NLCD {landClass} Class High Fraction (>= 0.
 ↪9)\nError Bars Equal Standard Deviation')
else:
    meanFig.suptitle(f'Mean NDVI vs NLCD {landClass} Class High Fraction (>= 0.
 ↪9)\nError Bars Equal 95% Confidence Interval')


axis = 0
for landClassItem in landClassNumber:
    landClass = list(landClassDict.keys())[list(landClassDict.values()).
 ↪index(landClassItem)]

    landClassYearMean = highFrac_df[highFrac_df['land class']==landClassItem].
 ↪groupby(['year']).mean().reset_index()
    landClassYearStdev = highFrac_df[highFrac_df['land class']==landClassItem].
 ↪groupby(['year']).std().reset_index()
    landClassYearCount = highFrac_df[highFrac_df['land class']==landClassItem].
 ↪groupby(['year']).count().reset_index()

    ciList = []
    for i in range(0,len(landClassYearCount)):
        ciList = ciList + [(landClassYearMean.iloc[i]['NDVI'])*(1.96/np.
 ↪sqrt(landClassYearCount.iloc[i]['NDVI']))]

    meanAxes[axis].set_title(f'{landClass}')

    if errorType == 'stdev':
        meanAxes[axis].
 ↪errorbar(landClassYearMean['year'],landClassYearMean['NDVI'],marker='o',markersize=4,yerr=l
    else:
        meanAxes[axis].
 ↪errorbar(landClassYearMean['year'],landClassYearMean['NDVI'],marker='o',markersize=4,yerr=c

    meanAxes[axis].set_ylabel('Mean NDVI')
```

```
    meanAxes[axis].set_xlabel('Year')

    axis = axis + 1

plt.show()
```


Mean NDVI vs NLCD Grasslands Class High Fraction (>= 0.9)
Error Bars Equal 95% Confidence Interval
Barren
Mean NDVI
0.7
0.6
0.5
0.4
2002
2004
2006
2008
2010
2012
2014
2016
Year
Forest Deciduous
Mean NDVI
0.86
0.84
0.82
0.80
2002
2004
2006
2008
2010
2012
2014
2016
Year
Shrub
Mean NDVI
0.9
0.8
0.7
0.6
0.5
2002
2004
2006
2008
2010
2012
2014
2016
Year
Grasslands
Mean NDVI
0.75
0.70
0.65
0.60
0.55
0.50
2002
2004
2006
2008
2010
2012
2014
2016
Year

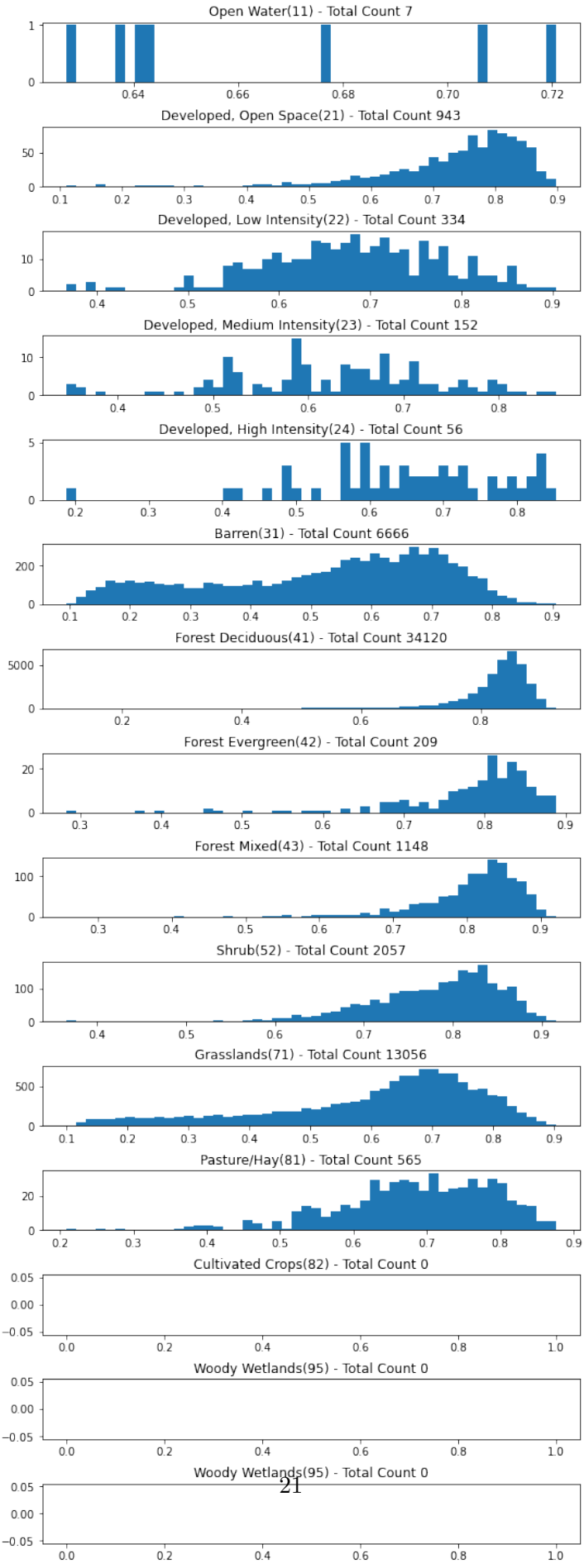
1.14 ARM distributions by land class across all years for high fraction features

[15]:
```
classHistFig, classHistAxes = plt.subplots(len(landClassDict),figsize=(7.5,20),␣
↪constrained_layout=True)
i = 0
for landClass in landClassDict.values():
    classHistAxes[i].hist(highFrac_df[highFrac_df['land␣
↪class']==landClass]['arm'],bins=50)
    descr = list(landClassDict.keys())[list(landClassDict.values()).
↪index(landClass)]
    landClassCount = len(highFrac_df[highFrac_df['land␣
↪class']==landClass]['arm'])
    classHistAxes[i].set_title(f'{descr}({str(landClass)}) - Total Count␣
↪{landClassCount}')
    i = i + 1
plt.show()
```


Open Water(11) - Total Count 7
Developed, Open Space(21) - Total Count 943
Developed, Low Intensity(22) - Total Count 334
Developed, Medium Intensity(23) - Total Count 152
Developed, High Intensity(24) - Total Count 56
Barren(31) - Total Count 6666
Forest Deciduous(41) - Total Count 34120
Forest Evergreen(42) - Total Count 209
Forest Mixed(43) - Total Count 1148
Shrub(52) - Total Count 2057
Grasslands(71) - Total Count 13056
Pasture/Hay(81) - Total Count 565
Cultivated Crops(82) - Total Count 0
Woody Wetlands(95) - Total Count 0
Woody Wetlands(95) - Total Count 0

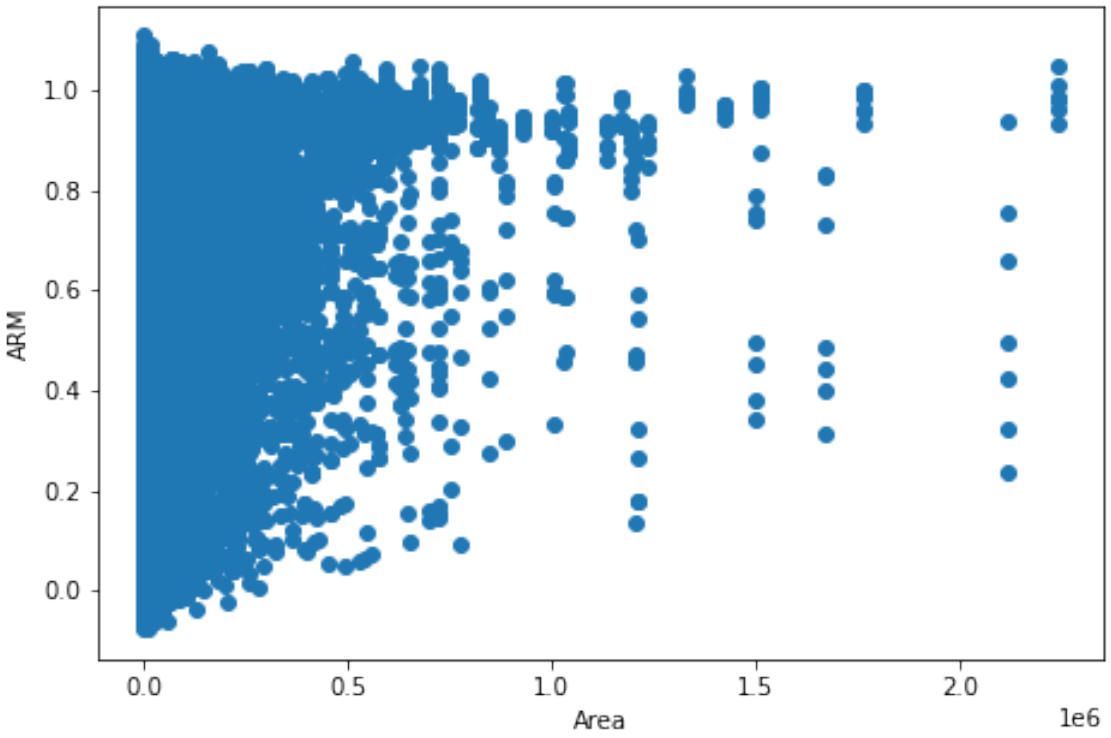
1.15 NDVI distributions by land class across all years for high fraction features

[16]:
```
classHistFig, classHistAxes = plt.subplots(len(landClassDict),figsize=(7.5,20),␣
↪constrained_layout=True)
i = 0
for landClass in landClassDict.values():
    classHistAxes[i].hist(highFrac_df[highFrac_df['land␣
↪class']==landClass]['NDVI'],bins=50)
    descr = list(landClassDict.keys())[list(landClassDict.values()).
↪index(landClass)]
    landClassCount = len(highFrac_df[highFrac_df['land␣
↪class']==landClass]['NDVI'])
    classHistAxes[i].set_title(f'{descr}({str(landClass)}) - Total Count␣
↪{landClassCount}')
    i = i + 1
plt.show()
```


Open Water(11) - Total Count 7
Developed, Open Space(21) - Total Count 943
Developed, Low Intensity(22) - Total Count 334
Developed, Medium Intensity(23) - Total Count 152
Developed, High Intensity(24) - Total Count 56
Barren(31) - Total Count 6666
Forest Deciduous(41) - Total Count 34120
Forest Evergreen(42) - Total Count 209
Forest Mixed(43) - Total Count 1148
Shrub(52) - Total Count 2057
Grasslands(71) - Total Count 13056
Pasture/Hay(81) - Total Count 565
Cultivated Crops(82) - Total Count 0
Woody Wetlands(95) - Total Count 0
Woody Wetlands(95) - Total Count 0

1.16 ARM by area over all features

[41]:
```
sizeFig, sizeAxes = plt.subplots(1,figsize=(7.5,5))
sizeAxes.scatter(stacked_df['sum'],stacked_df['arm'])
sizeAxes.set_xlabel('Area')
sizeAxes.set_ylabel('ARM')
plt.show()
```



[ ]:
```

```