Jericho McLeod

CSI-777 Assignment 2

For provided glass dataset, plot histograms for numeric attributes. For provided iris data, create discretized intervals with consistent distributions.

First, we need to import appropriate libaries and the datasets.

In [1]:
```
import csv
import matplotlib.pyplot as plt
import numpy as np
```

In [2]:
```
glass = 'Glass&IrisDATA/glass.csv'
iris = 'Glass&IrisDATA/iris.csv'

def import_func(filename):
    data = []
    with open(filename) as csvfile:
        csv_r = csv.reader(csvfile,delimiter = ',')
        for i in csv_r:
            data.append(i)
    return(data)

glass_data = import_func(glass)
iris_data = import_func(iris)
```

Now we need to examine the glass dataset to build a basic understanding of its contents.

```
In [3]: for i in range(len(glass_data)):
            if i < 11:
                print(glass_data[i])
```

```
['refractive_index', 'Sodium', 'Magnesium', 'Aluminum', 'Silicon', 'Pot
assium', 'Calcium', 'Barium', 'Iron', 'Type']
['1.51793', '12.79', '3.5', '1.12', '73.03', '0.64', '8.77', '0', '0',
"'build wind float'"]
['1.51643', '12.16', '3.52', '1.35', '72.89', '0.57', '8.53', '0', '0',
"'vehic wind float'"]
['1.51793', '13.21', '3.48', '1.41', '72.64', '0.59', '8.43', '0', '0',
"'build wind float'"]
['1.51299', '14.4', '1.74', '1.54', '74.55', '0', '7.59', '0', '0', 'ta
bleware']
['1.53393', '12.3', '0', '1', '70.16', '0.12', '16.19', '0', '0.24',
"'build wind non-float'"]
['1.51655', '12.75', '2.85', '1.44', '73.27', '0.57', '8.79', '0.11',
'0.22', "'build wind non-float'"]
['1.51779', '13.64', '3.65', '0.65', '73', '0.06', '8.93', '0', '0',
"'vehic wind float'"]
['1.51837', '13.14', '2.84', '1.28', '72.85', '0.55', '9.07', '0', '0',
"'build wind float'"]
['1.51545', '14.14', '0', '2.68', '73.39', '0.08', '9.07', '0.61', '0.0
5', 'headlamps']
['1.51789', '13.19', '3.9', '1.3', '72.33', '0.55', '8.44', '0', '0.2
8', "'build wind non-float'"]
```

The data is structured with headers in the first row and instances on subsequent rows. Since we will be dealing with attributes independently, converting it to a dictionary will simplify tasks.

We will also check input lengths here.

```
In [4]: headers = glass_data[0] # extract the attribute names
        glass_data_dict = {}
        glass_hist_dict = {}
        for i in headers:
            glass_data_dict[i] = []
            glass_hist_dict[i] = {}
        for i in range(len(glass_data)):
            if i > 0: # to skip the attribute names in the loop
                for j in range(len(glass_data[i])):
                    try: # try adding data as floats, on exception as as strings
                        glass_data_dict[headers[j]].append(float(glass_data[i][j
        ]))
                    except:
                        glass_data_dict[headers[j]].append(glass_data[i][j])

        for k,v in glass_data_dict.items():
            print(k,'contains',len(v),'values')
```

```
refractive_index contains 214 values
Sodium contains 214 values
Magnesium contains 214 values
Aluminum contains 214 values
Silicon contains 214 values
Potassium contains 214 values
Calcium contains 214 values
Barium contains 214 values
Iron contains 214 values
Type contains 214 values
```

Since it is easy to spot-check the first instance, we will use it as the determining 'type' to identify which attributes contain numeric vs. textual data.

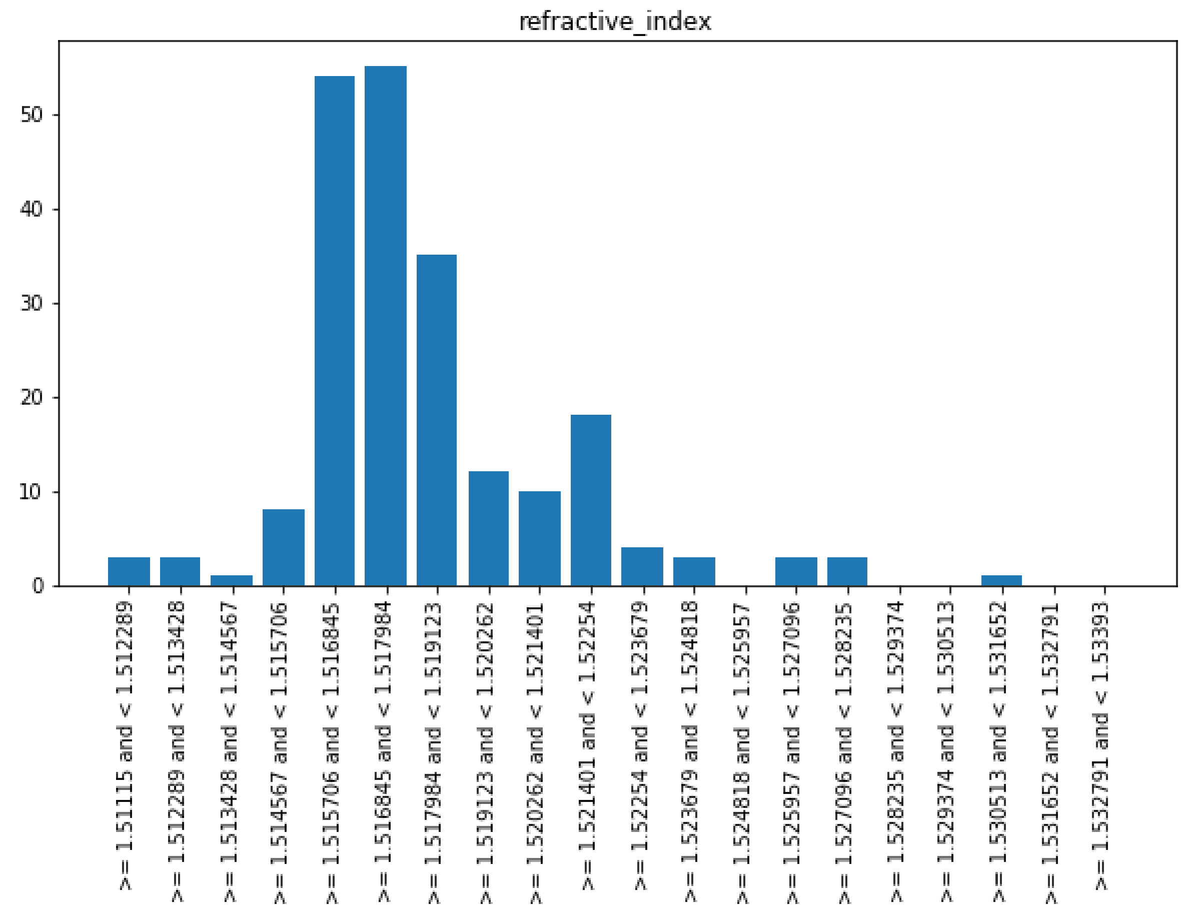
To get histogram counts for an attribute, we need to know the range across which it is distributed, then divide that by some amount and count the instances which fall into the defined subset. Below, 'n_categories' specifies the histogram width.

```
In [5]: n_categories = 20

        for i in headers:
            glass_hist_dict[i] = {}

        for k,v in glass_data_dict.items():
            if isinstance(v[0],float) or isinstance(v[0],int):
                min_v   = min(v)
                max_v   = max(v)
                range_v = max_v-min_v
                print(k,min_v,max_v)
                step = range_v/n_categories
                for i in range(n_categories):
                    lower = min_v + step*i
                    upper = lower + step
                    name = '>= '+str(lower)+' and < '+str(upper)
                    glass_hist_dict[k][name] = 0
                    for j in v:
                        if j >= lower and j < upper:
                            glass_hist_dict[k][name] +=1
```

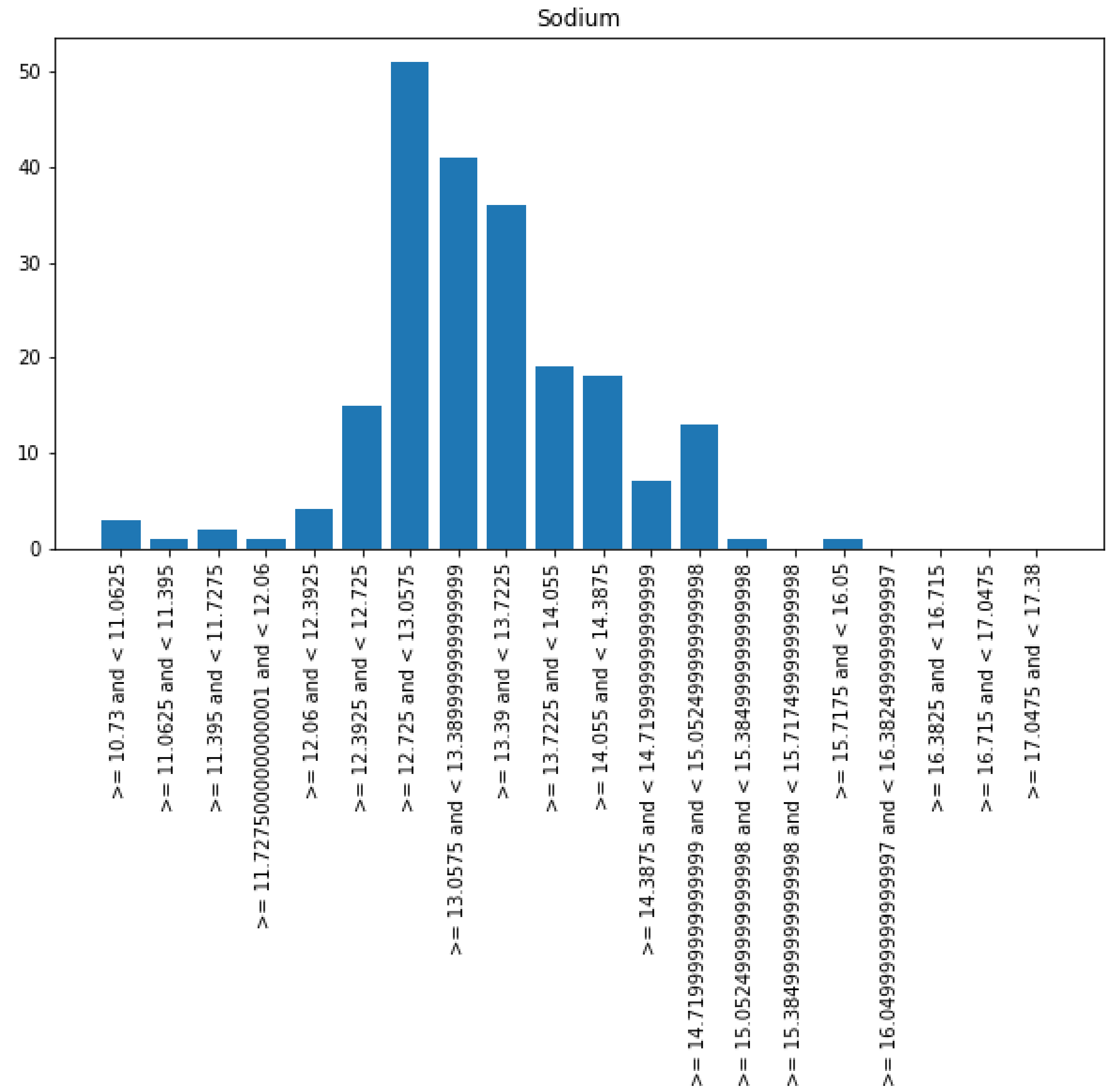
```
refractive_index 1.51115 1.53393
Sodium 10.73 17.38
Magnesium 0.0 4.49
Aluminum 0.29 3.5
Silicon 69.81 75.41
Potassium 0.0 6.21
Calcium 5.43 16.19
Barium 0.0 3.15
Iron 0.0 0.51
```

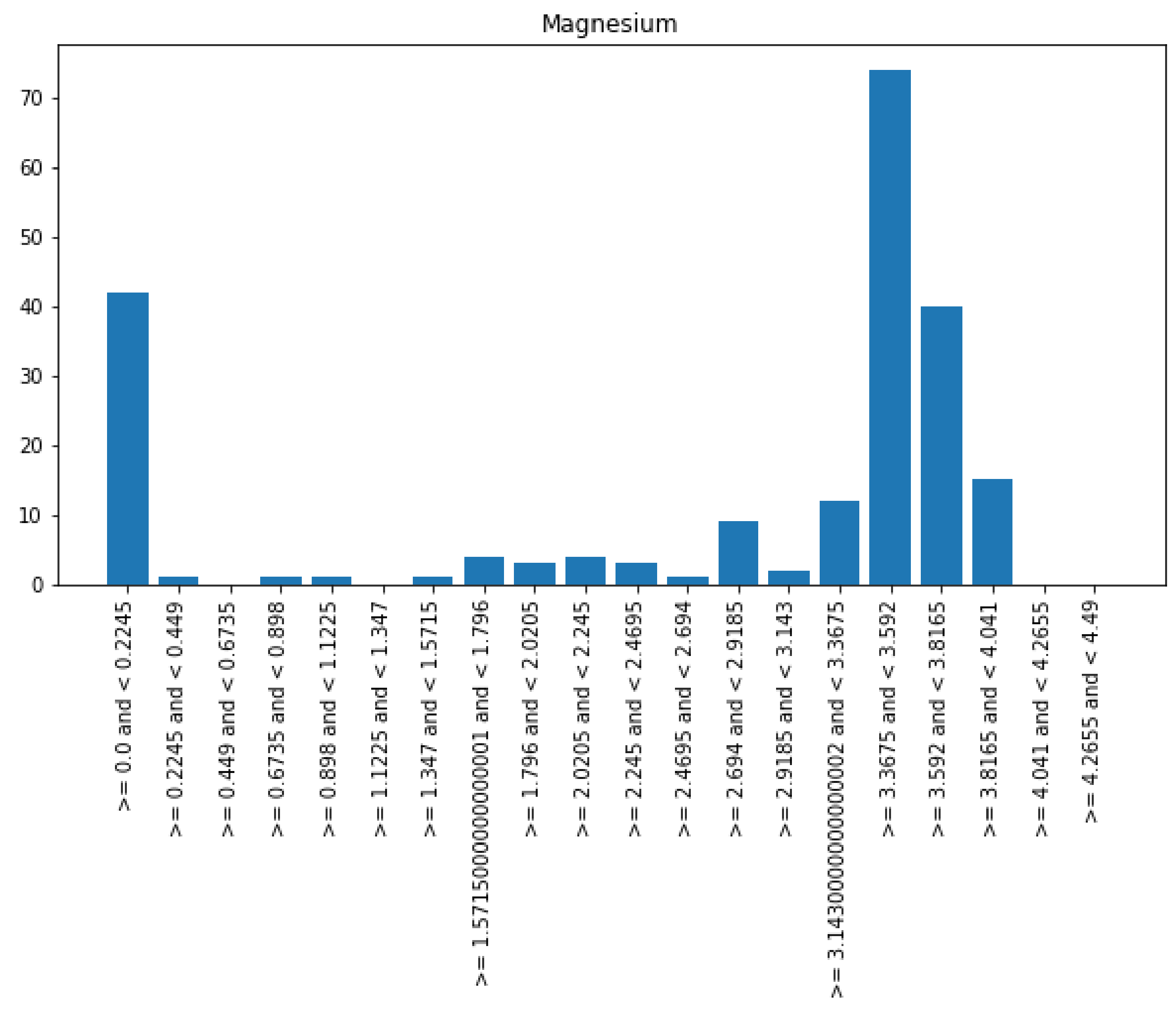
All that remains is to show plots for the actual histograms.

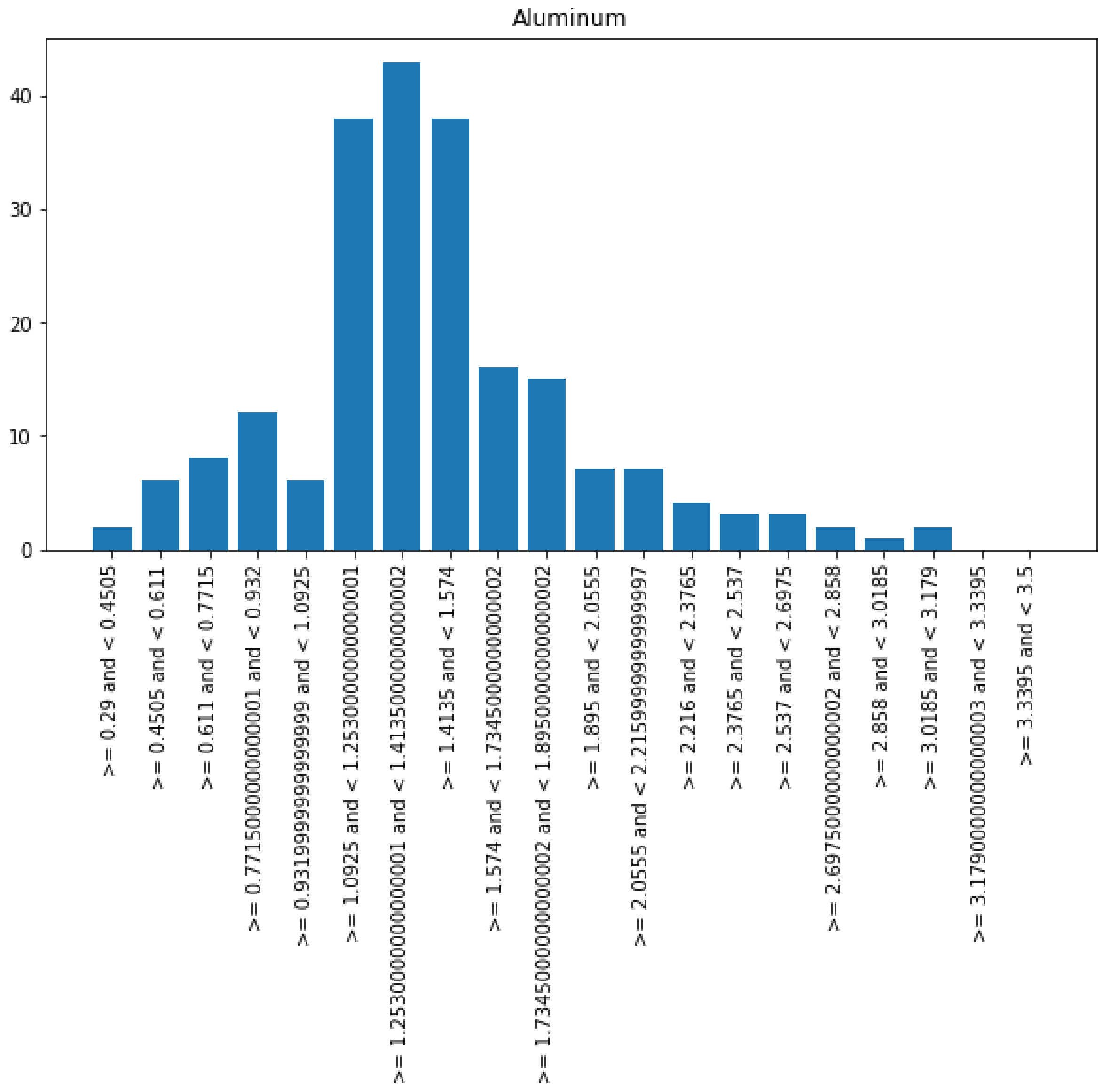
In [6]:
```
for k,v in glass_hist_dict.items():
    if len(v)>0:
        x = []
        x_labels = []
        for k2,v2 in v.items():
            x.append(v2)
            x_labels.append(k2)
        plt.figure(figsize=(10,5))
        plt.bar(x_labels,x)
        plt.xticks(x_labels,rotation='vertical')
        plt.title(k)
        plt.show()
```

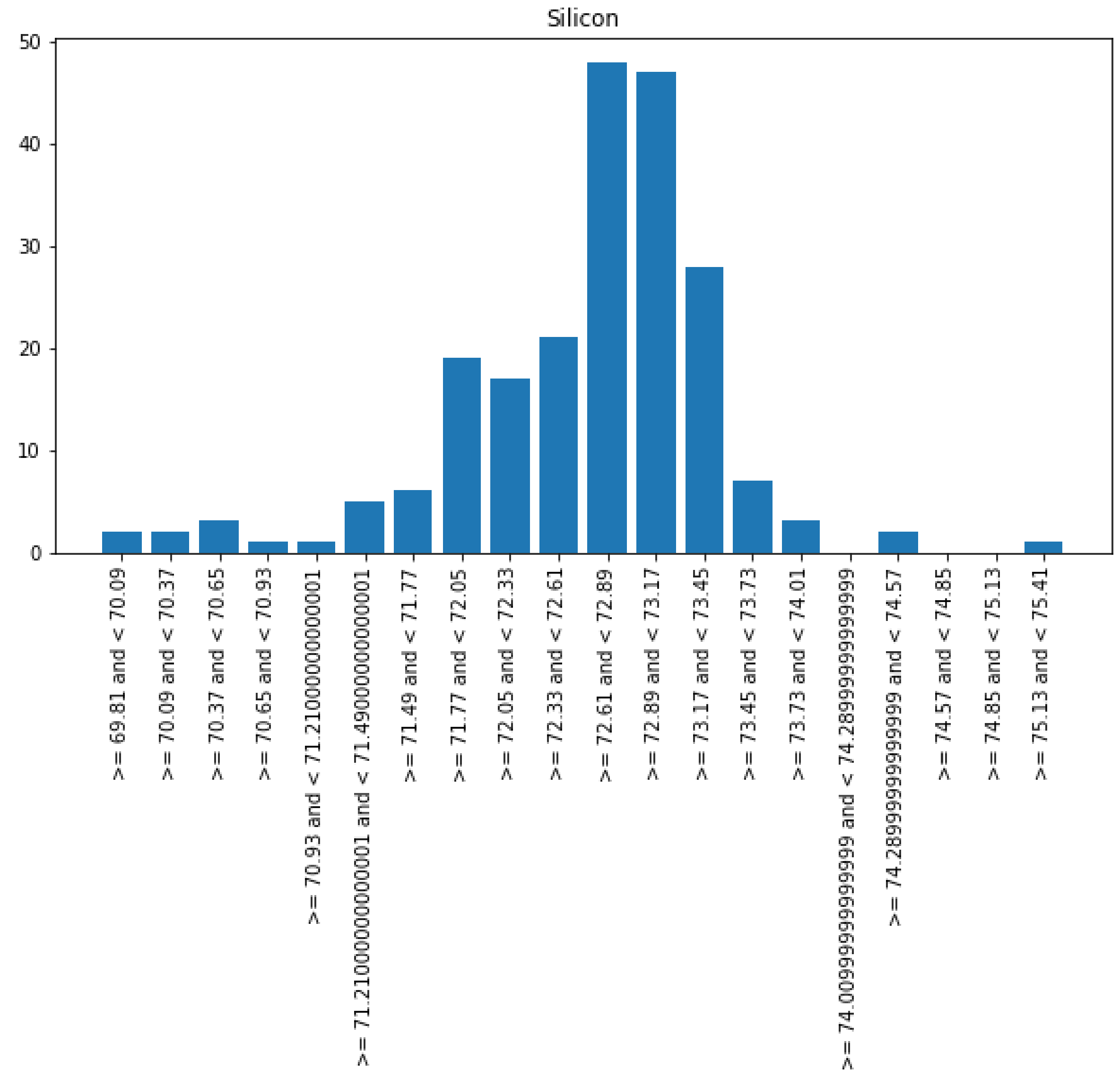


refractive_index
0
10
20
30
40
50
>= 1.51115 and < 1.512289
>= 1.512289 and < 1.513428
>= 1.513428 and < 1.514567
>= 1.514567 and < 1.515706
>= 1.515706 and < 1.516845
>= 1.516845 and < 1.517984
>= 1.517984 and < 1.519123
>= 1.519123 and < 1.520262
>= 1.520262 and < 1.521401
>= 1.521401 and < 1.52254
>= 1.52254 and < 1.523679
>= 1.523679 and < 1.524818
>= 1.524818 and < 1.525957
>= 1.525957 and < 1.527096
>= 1.527096 and < 1.528235
>= 1.528235 and < 1.529374
>= 1.529374 and < 1.530513
>= 1.530513 and < 1.531652
>= 1.531652 and < 1.532791
>= 1.532791 and < 1.53393

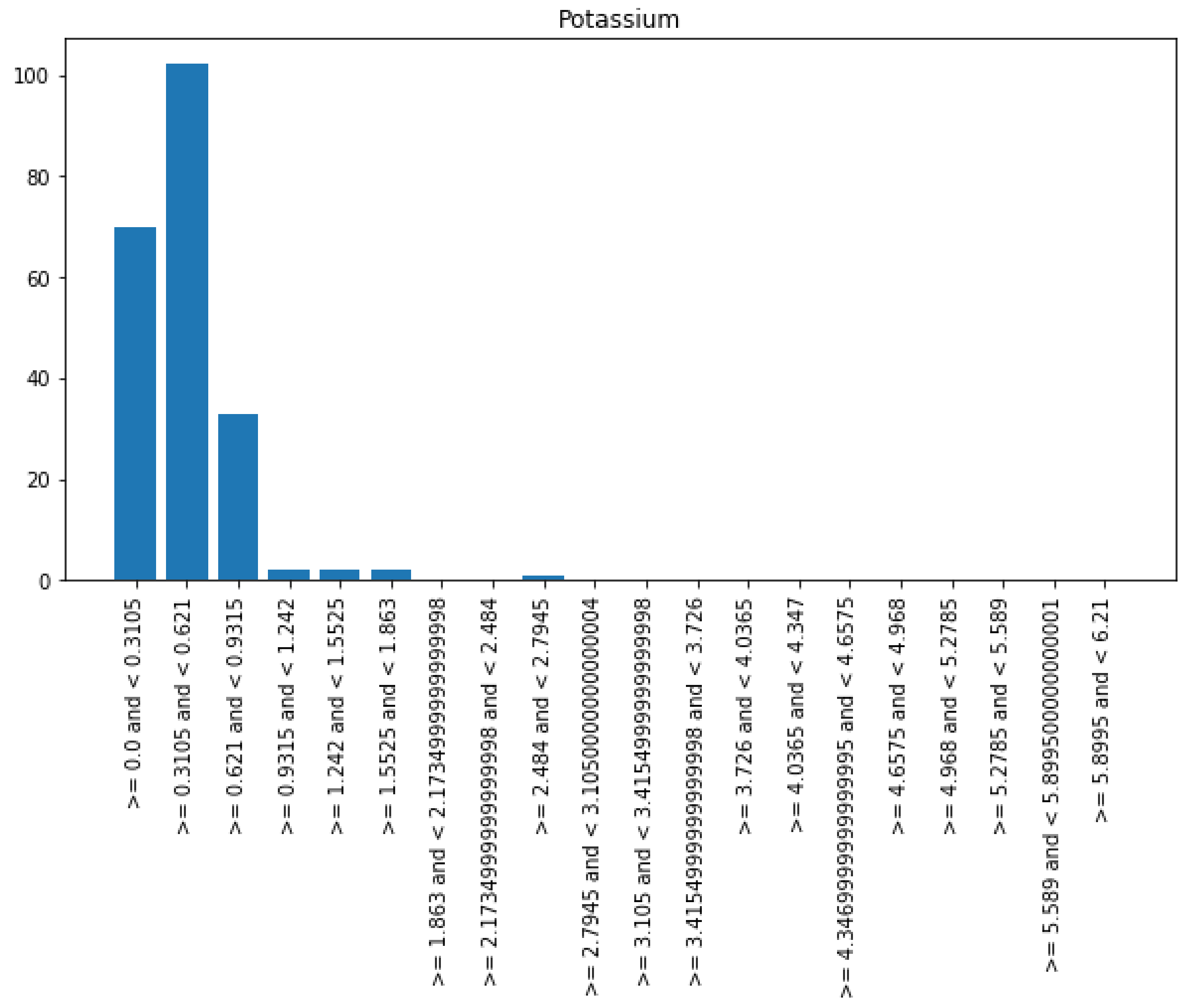

Sodium
0
10
20
30
40
50
>= 10.73 and < 11.0625
>= 11.0625 and < 11.395
>= 11.395 and < 11.7275
>= 11.727500000000001 and < 12.06
>= 12.06 and < 12.3925
>= 12.3925 and < 12.725
>= 12.725 and < 13.0575
>= 13.0575 and < 13.389999999999999
>= 13.39 and < 13.7225
>= 13.7225 and < 14.055
>= 14.055 and < 14.3875
>= 14.3875 and < 14.719999999999999
>= 14.719999999999999 and < 15.052499999999998
>= 15.052499999999998 and < 15.384999999999998
>= 15.384999999999998 and < 15.717499999999998
>= 15.7175 and < 16.05
>= 16.049999999999997 and < 16.382499999999997
>= 16.3825 and < 16.715
>= 16.715 and < 17.0475
>= 17.0475 and < 17.38


Magnesium
0
10
20
30
40
50
60
70
>= 0.0 and < 0.2245
>= 0.2245 and < 0.449
>= 0.449 and < 0.6735
>= 0.6735 and < 0.898
>= 0.898 and < 1.1225
>= 1.1225 and < 1.347
>= 1.347 and < 1.5715
>= 1.5715000000000001 and < 1.796
>= 1.796 and < 2.0205
>= 2.0205 and < 2.245
>= 2.245 and < 2.4695
>= 2.4695 and < 2.694
>= 2.694 and < 2.9185
>= 2.9185 and < 3.143
>= 3.1430000000000002 and < 3.3675
>= 3.3675 and < 3.592
>= 3.592 and < 3.8165
>= 3.8165 and < 4.041
>= 4.041 and < 4.2655
>= 4.2655 and < 4.49

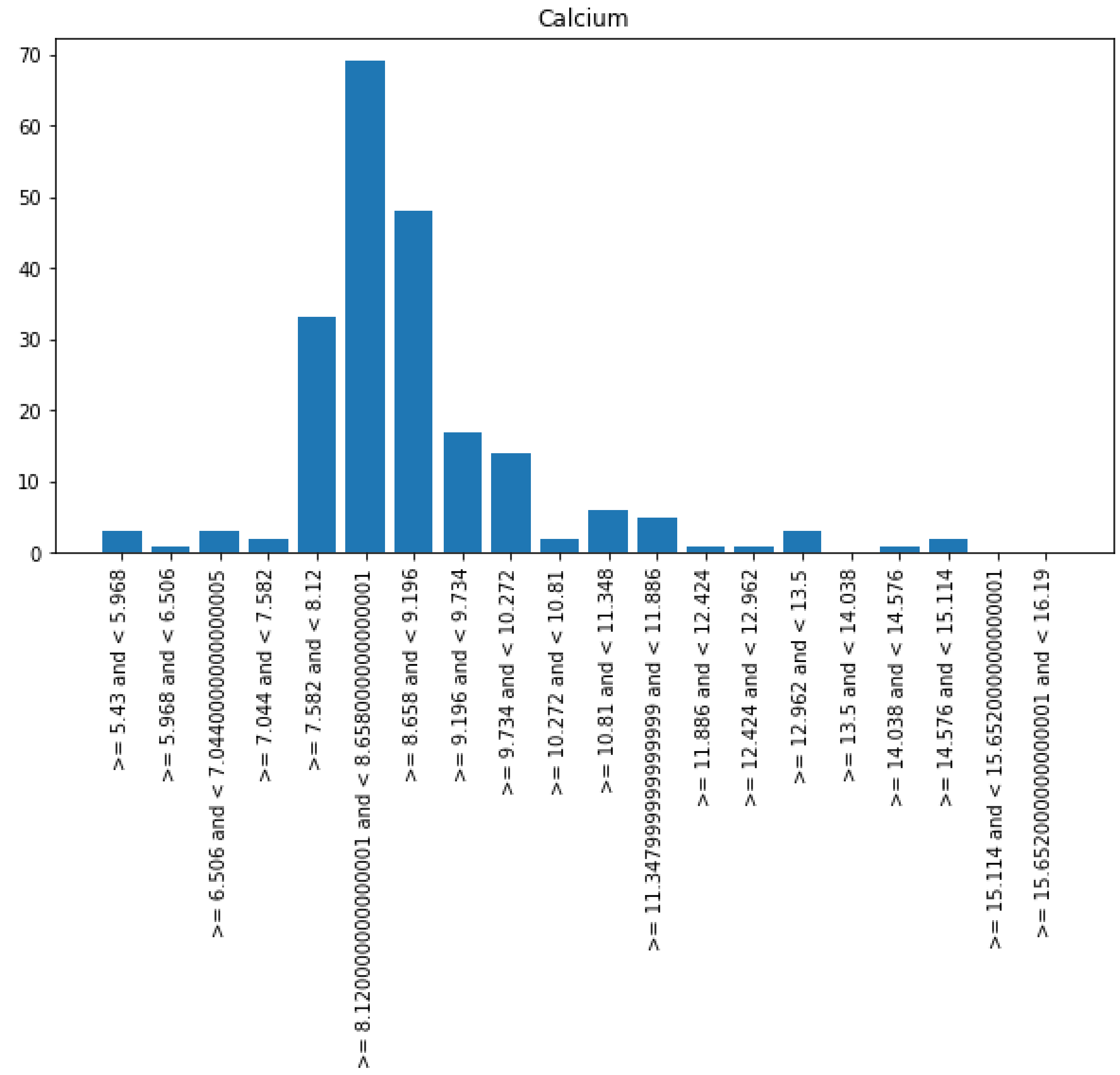

Aluminum
0
10
20
30
40
>= 0.29 and < 0.4505
>= 0.4505 and < 0.611
>= 0.611 and < 0.7715
>= 0.7715000000000001 and < 0.932
>= 0.9319999999999999 and < 1.0925
>= 1.0925 and < 1.2530000000000001
>= 1.2530000000000001 and < 1.4135000000000002
>= 1.4135 and < 1.574
>= 1.574 and < 1.7345000000000002
>= 1.7345000000000002 and < 1.8950000000000002
>= 1.895 and < 2.0555
>= 2.0555 and < 2.2159999999999997
>= 2.216 and < 2.3765
>= 2.3765 and < 2.537
>= 2.537 and < 2.6975
>= 2.6975000000000002 and < 2.858
>= 2.858 and < 3.0185
>= 3.0185 and < 3.179
>= 3.1790000000000003 and < 3.3395
>= 3.3395 and < 3.5

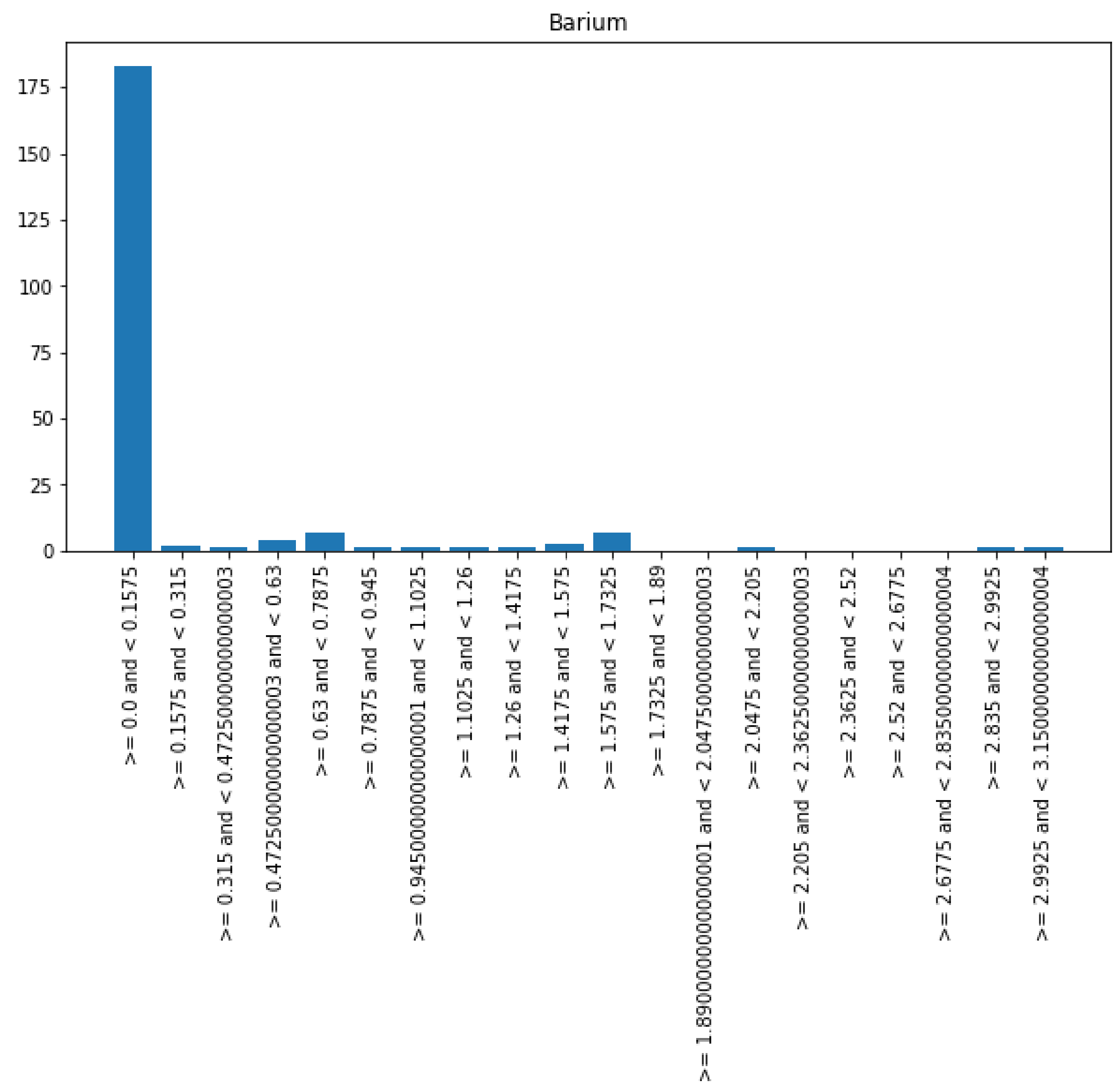


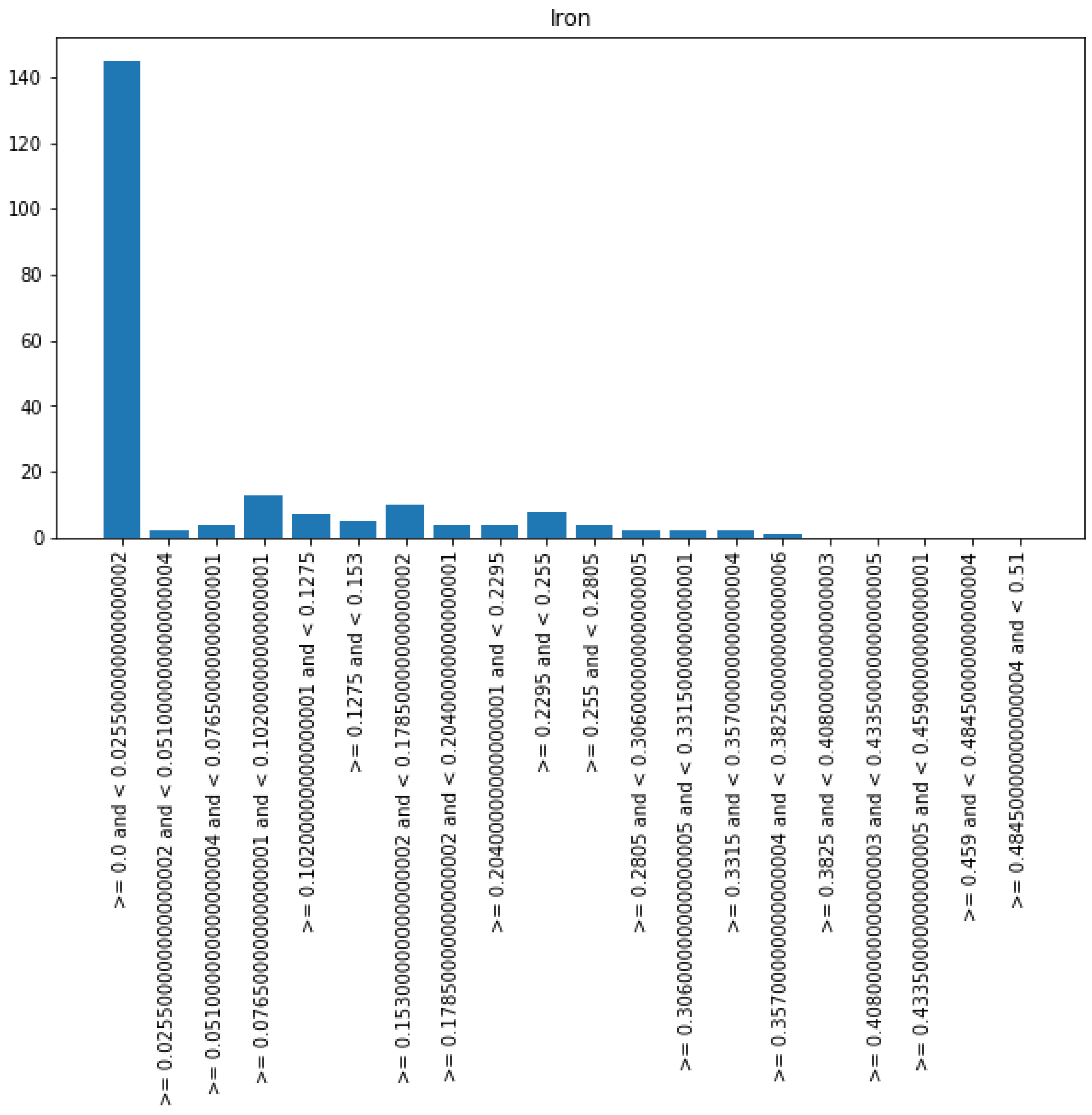
Silicon
0
10
20
30
40
50
>= 69.81 and < 70.09
>= 70.09 and < 70.37
>= 70.37 and < 70.65
>= 70.65 and < 70.93
>= 70.93 and < 71.21000000000001
>= 71.21000000000001 and < 71.49000000000001
>= 71.49 and < 71.77
>= 71.77 and < 72.05
>= 72.05 and < 72.33
>= 72.33 and < 72.61
>= 72.61 and < 72.89
>= 72.89 and < 73.17
>= 73.17 and < 73.45
>= 73.45 and < 73.73
>= 73.73 and < 74.01
>= 74.00999999999999 and < 74.28999999999999
>= 74.28999999999999 and < 74.57
>= 74.57 and < 74.85
>= 74.85 and < 75.13
>= 75.13 and < 75.41



Potassium
0
20
40
60
80
100
>= 0.0 and < 0.3105
>= 0.3105 and < 0.621
>= 0.621 and < 0.9315
>= 0.9315 and < 1.242
>= 1.242 and < 1.5525
>= 1.5525 and < 1.863
>= 1.863 and < 2.1734999999999998
>= 2.1734999999999998 and < 2.484
>= 2.484 and < 2.7945
>= 2.7945 and < 3.1050000000000004
>= 3.105 and < 3.4154999999999998
>= 3.4154999999999998 and < 3.726
>= 3.726 and < 4.0365
>= 4.0365 and < 4.347
>= 4.3469999999999995 and < 4.6575
>= 4.6575 and < 4.968
>= 4.968 and < 5.2785
>= 5.2785 and < 5.589
>= 5.589 and < 5.8995000000000001
>= 5.8995 and < 6.21


Calcium
0
10
20
30
40
50
60
70
>= 5.43 and < 5.968
>= 5.968 and < 6.506
>= 6.506 and < 7.044000000000005
>= 7.044 and < 7.582
>= 7.582 and < 8.12
>= 8.120000000000001 and < 8.658000000000001
>= 8.658 and < 9.196
>= 9.196 and < 9.734
>= 9.734 and < 10.272
>= 10.272 and < 10.81
>= 10.81 and < 11.348
>= 11.347999999999999 and < 11.886
>= 11.886 and < 12.424
>= 12.424 and < 12.962
>= 12.962 and < 13.5
>= 13.5 and < 14.038
>= 14.038 and < 14.576
>= 14.576 and < 15.114
>= 15.114 and < 15.652000000000001
>= 15.652000000000001 and < 16.19


Barium
175
150
125
100
75
50
25
0
>= 0.0 and < 0.1575
>= 0.1575 and < 0.315
>= 0.315 and < 0.47250000000000003
>= 0.47250000000000003 and < 0.63
>= 0.63 and < 0.7875
>= 0.7875 and < 0.945
>= 0.9450000000000001 and < 1.1025
>= 1.1025 and < 1.26
>= 1.26 and < 1.4175
>= 1.4175 and < 1.575
>= 1.575 and < 1.7325
>= 1.7325 and < 1.89
>= 1.8900000000000001 and < 2.0475000000000003
>= 2.0475 and < 2.205
>= 2.205 and < 2.3625000000000003
>= 2.3625 and < 2.52
>= 2.52 and < 2.6775
>= 2.6775 and < 2.8350000000000004
>= 2.835 and < 2.9925
>= 2.9925 and < 3.1500000000000004



Moving on to the iris dataset, where we need to create discretized intervals with consistent distributions, unlike we processed for the Glass dataset.

In [7]:
```
for i in range(len(iris_data)):
    if i < 11:
        print(iris_data[i])

print('\nTotal instances',len(iris_data)-1)
```

```
['petallength', 'petalwidth', ' class']
['1.4', '0.2', 'Iris-setosa']
['1.4', '0.2', 'Iris-setosa']
['1.3', '0.2', 'Iris-setosa']
['1.5', '0.2', 'Iris-setosa']
['1.4', '0.2', 'Iris-setosa']
['1.7', '0.4', 'Iris-setosa']
['1.4', '0.3', 'Iris-setosa']
['1.5', '0.2', 'Iris-setosa']
['1.4', '0.2', 'Iris-setosa']
['1.5', '0.1', 'Iris-setosa']

Total instances 150
```

Since there are just two attributes, we will store them in individual variables.

In [8]:
```
petallength,petalwidth = [],[]
for i in range(len(iris_data)):
    if i>0:
        petallength.append(iris_data[i][0])
        petalwidth.append(iris_data[i][1])
```

To discretize the data into n categories, we need only sort the data, determine the number of items per category as $length/n$, and find the values at indexes of multiples of that value.

However, given that the iris data is relatively short in terms of significant digits, it is entirely possible that a chosen value could be in the middle of a collection of identical values, which will somewhat disrupt distributions.

In [9]:
```
def create_indexes(variable,n):
    length = len(variable)
    cat_size = length/n
    indexes = []
    for i in range(n):
        min_val = cat_size * i
        max_val = (cat_size * (i+1))-1
        indexes.append((min_val,max_val))
    return(indexes)


n = 10
index_vals = create_indexes(petallength,n)
for i in index_vals:
    print(i)
```

```
(0.0, 14.0)
(15.0, 29.0)
(30.0, 44.0)
(45.0, 59.0)
(60.0, 74.0)
(75.0, 89.0)
(90.0, 104.0)
(105.0, 119.0)
(120.0, 134.0)
(135.0, 149.0)
```

Next we need to convert the indexes to actual values in the dataset. If we skip this step the resulting categories will be completely equal distributions, but we may also see identical values in different categories arbitrarily.

In [10]:
```
def find_values_by_index(variable,indexes):
    values = []
    for i in indexes:
        low = int(i[0])
        high = int(i[1])
        sorted_var = sorted(variable)
        values.append((sorted_var[low],sorted_var[high]))
    return(values)

plength_cats = find_values_by_index(petallength,index_vals)
pwidth_cats = find_values_by_index(petalwidth,index_vals)

print(plength_cats)
print(pwidth_cats)
```

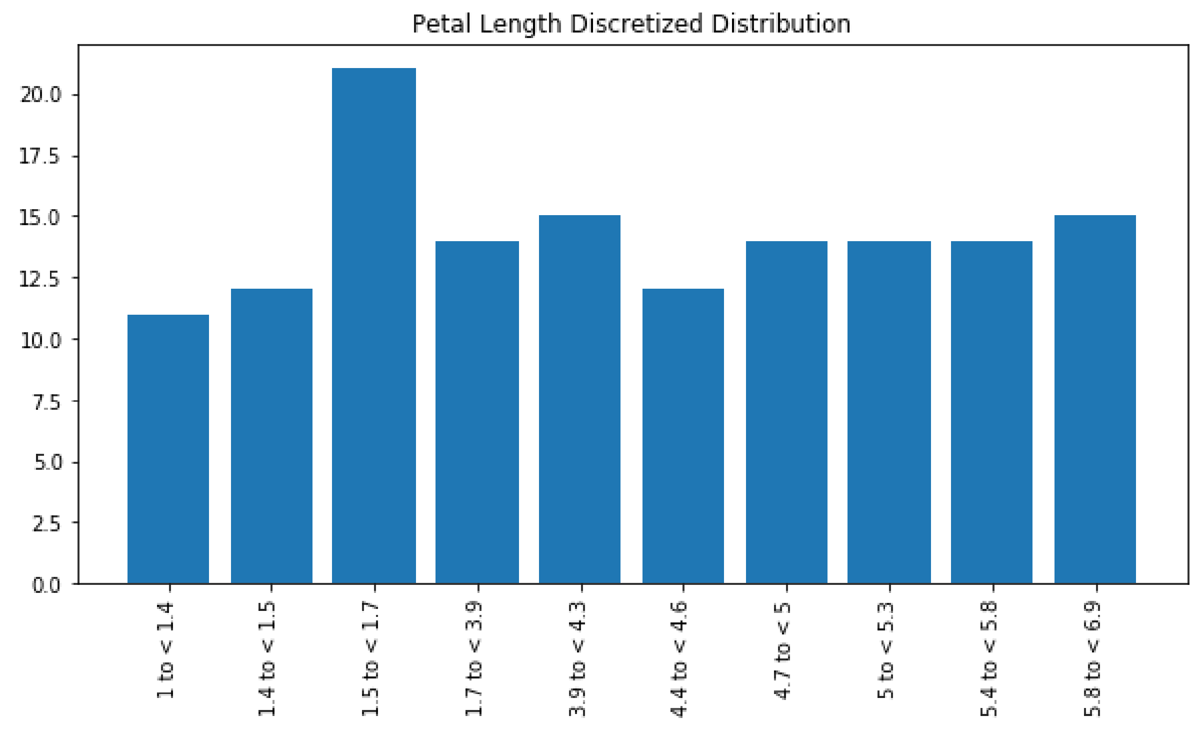
```
[('1', '1.4'), ('1.4', '1.5'), ('1.5', '1.7'), ('1.7', '3.9'), ('3.9',
'4.3'), ('4.4', '4.6'), ('4.7', '5'), ('5', '5.3'), ('5.4', '5.8'),
('5.8', '6.9')]
[('0.1', '0.2'), ('0.2', '0.2'), ('0.2', '0.4'), ('0.4', '1.1'), ('1.
2', '1.3'), ('1.3', '1.5'), ('1.5', '1.8'), ('1.8', '1.9'), ('1.9', '2.
2'), ('2.2', '2.5')]
```

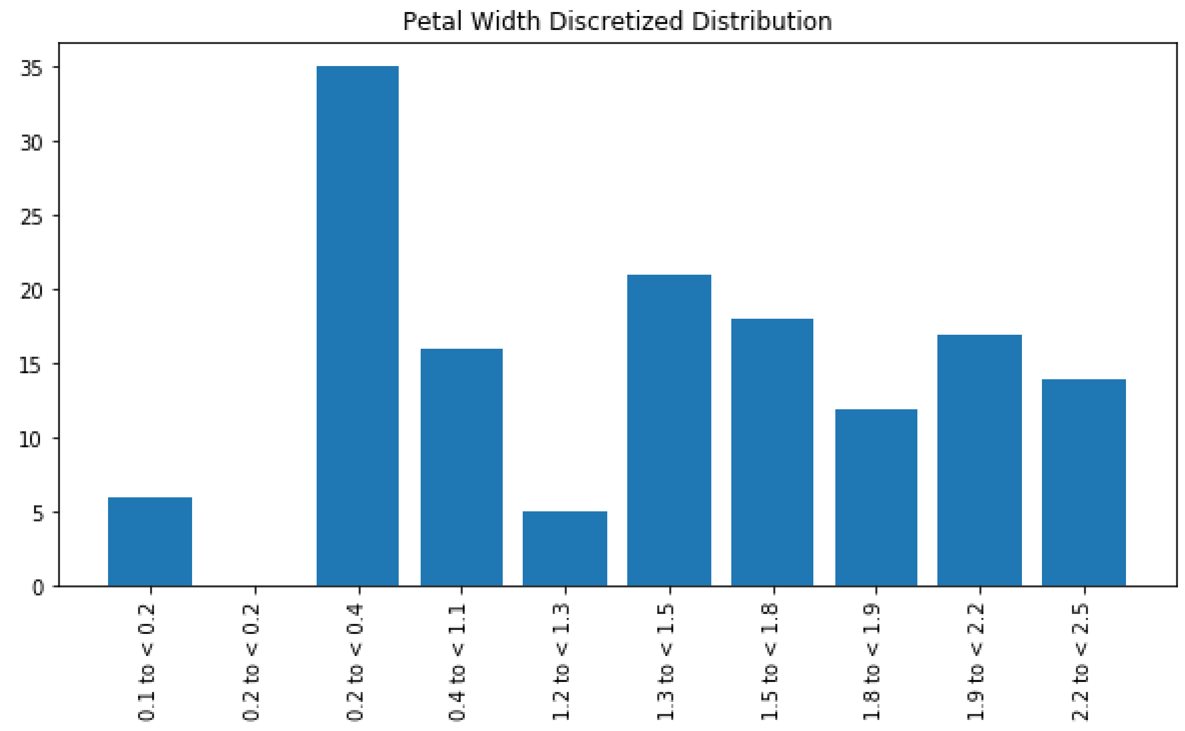
It is readily apparent that the distributions will be unequal. This is an artifact of having data points which do not divide evenly over our number of categories. For datasets that contain no duplicate values this will not be the case.

We will use >= the lower value, and < the upper value, so that no data point is used twice and so that identical values are not in multiple bins. At this point, all we need to do is show that our intervals are approximately even distributions, which we will do in histograms.

In [11]:
```
def get_vis_data(indexes,variable):
    histo_data = {}
    for i in range(len(indexes)):
        name = str(indexes[i][0])+' to < '+str(indexes[i][1])
        histo_data[name] = 0
        for j in variable:
            if j >= indexes[i][0] and j < indexes[i][1]:
                histo_data[name]+=1
    return(histo_data)

plength_hist_data = get_vis_data(plength_cats,petallength)
pwidth_hist_data = get_vis_data(pwidth_cats,petalwidth)
```

```
In [12]: def display_vis(histo_dict,string):
             x_labels,x = [],[]
             for k,v in histo_dict.items():
                 x_labels.append(k)
                 x.append(v)
             plt.figure(figsize=(10,5))
             plt.bar(x_labels,x)
             plt.xticks(x_labels,rotation='vertical')
             plt.title(string)
             plt.show()

         display_vis(plength_hist_data,'Petal Length Discretized Distribution')
         display_vis(pwidth_hist_data,'Petal Width Discretized Distribution')
```





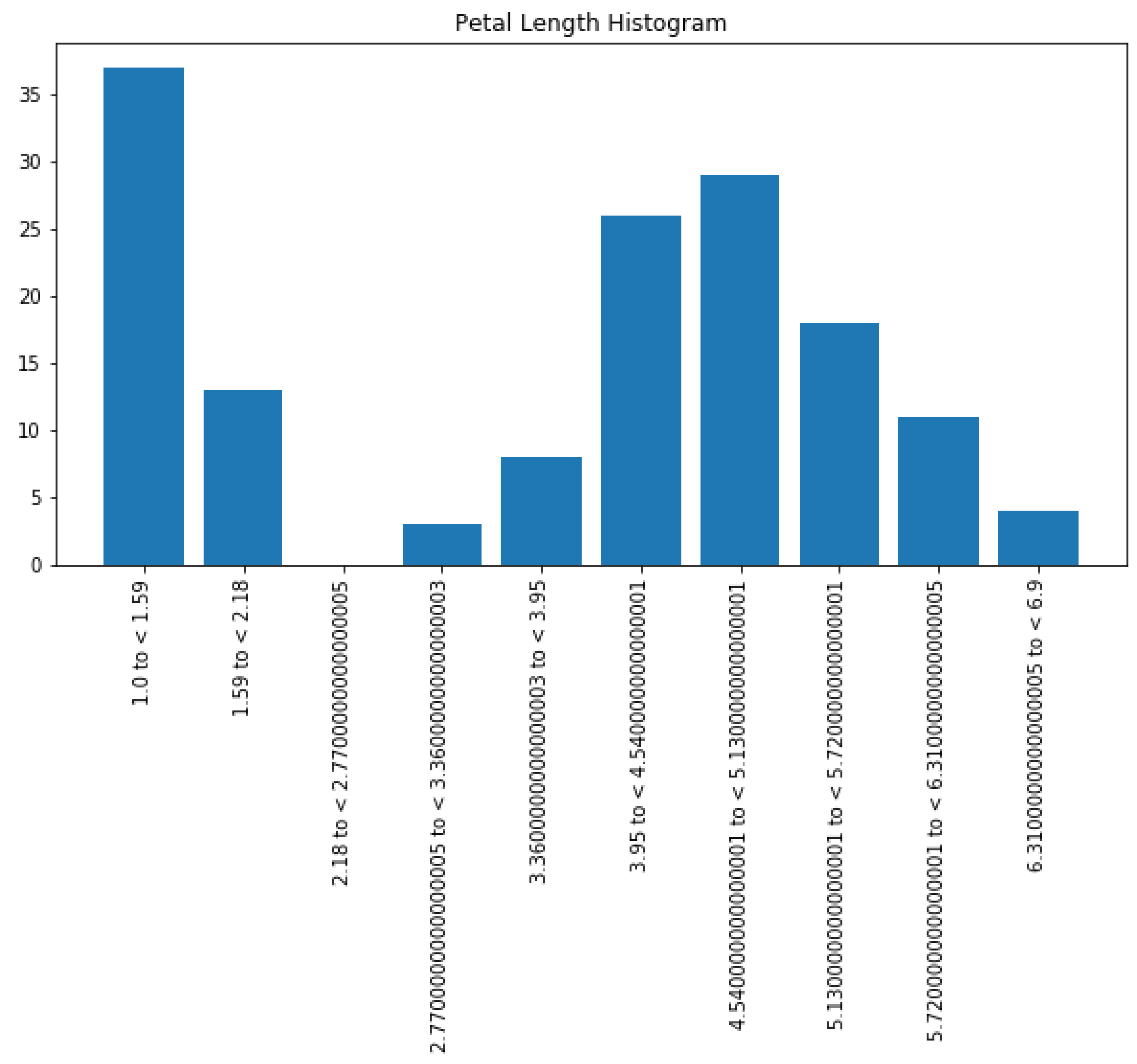
Petal length discretizes easily into bins, while petal width has some apparent issues. For one, there is a category that was completely encompassed by a single measurement in the data. Since the script looks for >= a minimum and < a maximum, this means that it is an empty category. This could be dropped by adding a simple if value > 0 statement when building the visualizations.

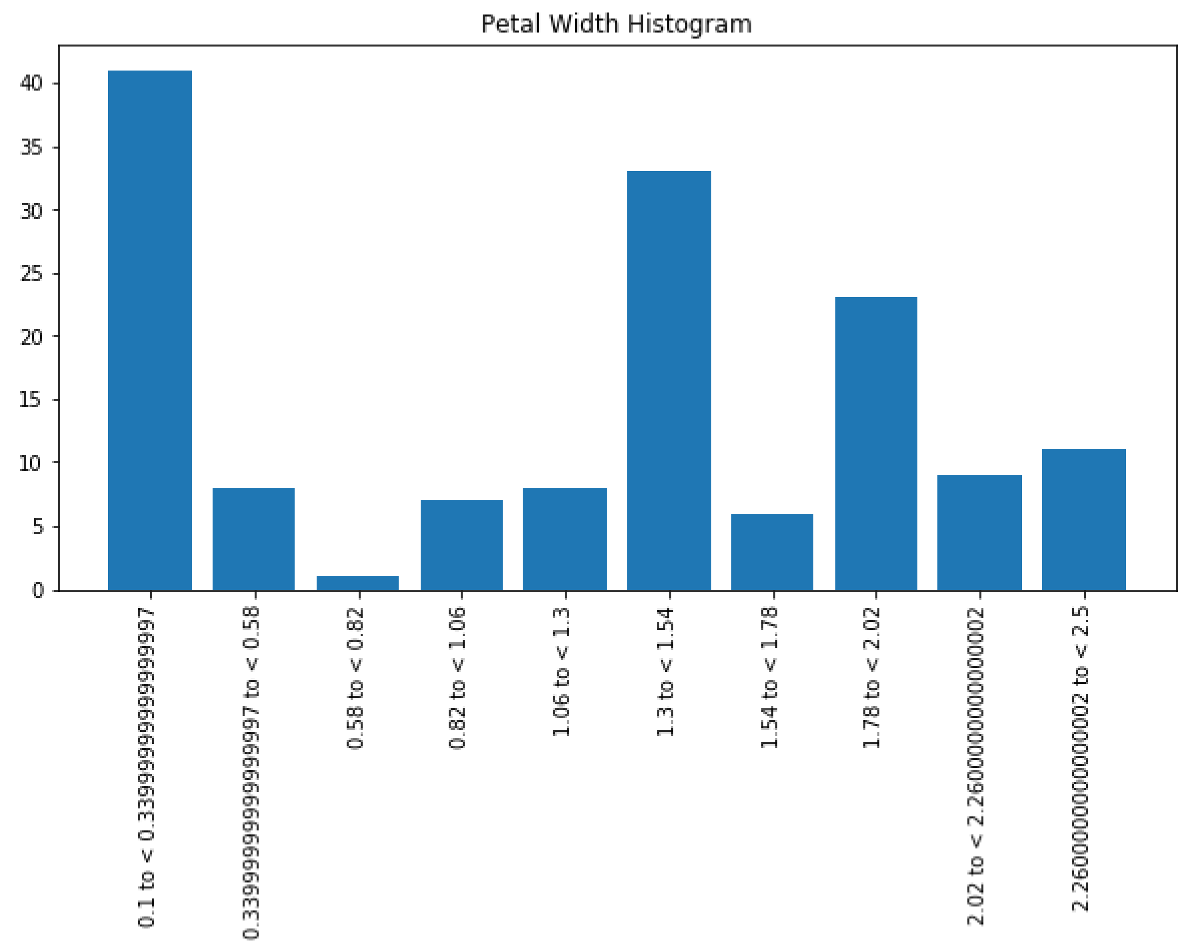
Further, because of the quantity of values with a 0.2 measurement in Petal Width, there is a large category where they did get counted.

A manual distribution may be a better choice, but let's first consider the actual histogram.

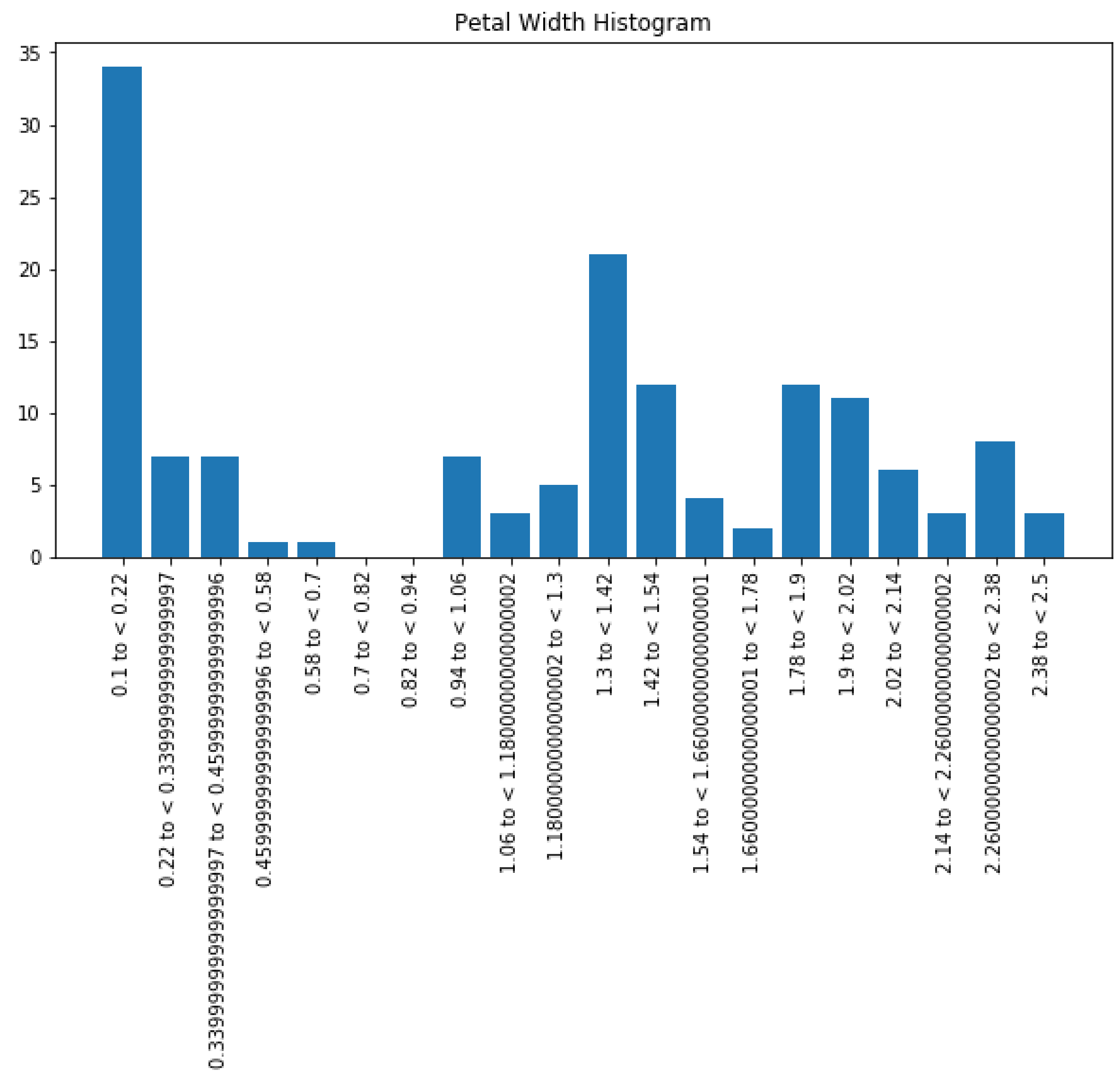
```
In [ ]:
```

In [13]:
```
def create_true_hist(variable,n,string):
    min_val = float(min(variable))
    max_val = float(max(variable))
    range_var = max_val-min_val
    step = range_var/n
    x,x_lab = [],[]
    for i in range(n):
        min_step = min_val + (step*i)
        max_step = min_val + (step *(i+1))
        x_lab.append(str(min_step)+' to < '+str(max_step))
        count = 0
        for j in range(len(variable)):
            if float(variable[j]) >= min_step and float(variable[j]) < m
ax_step:
                count+=1
        x.append(count)
    plt.figure(figsize=(10,5))
    plt.bar(x_lab,x)
    plt.xticks(x_lab,rotation='vertical')
    plt.title(string)
    plt.show()

n = 10
create_true_hist(petallength,n,'Petal Length Histogram')
create_true_hist(petalwidth,n,'Petal Width Histogram')
```

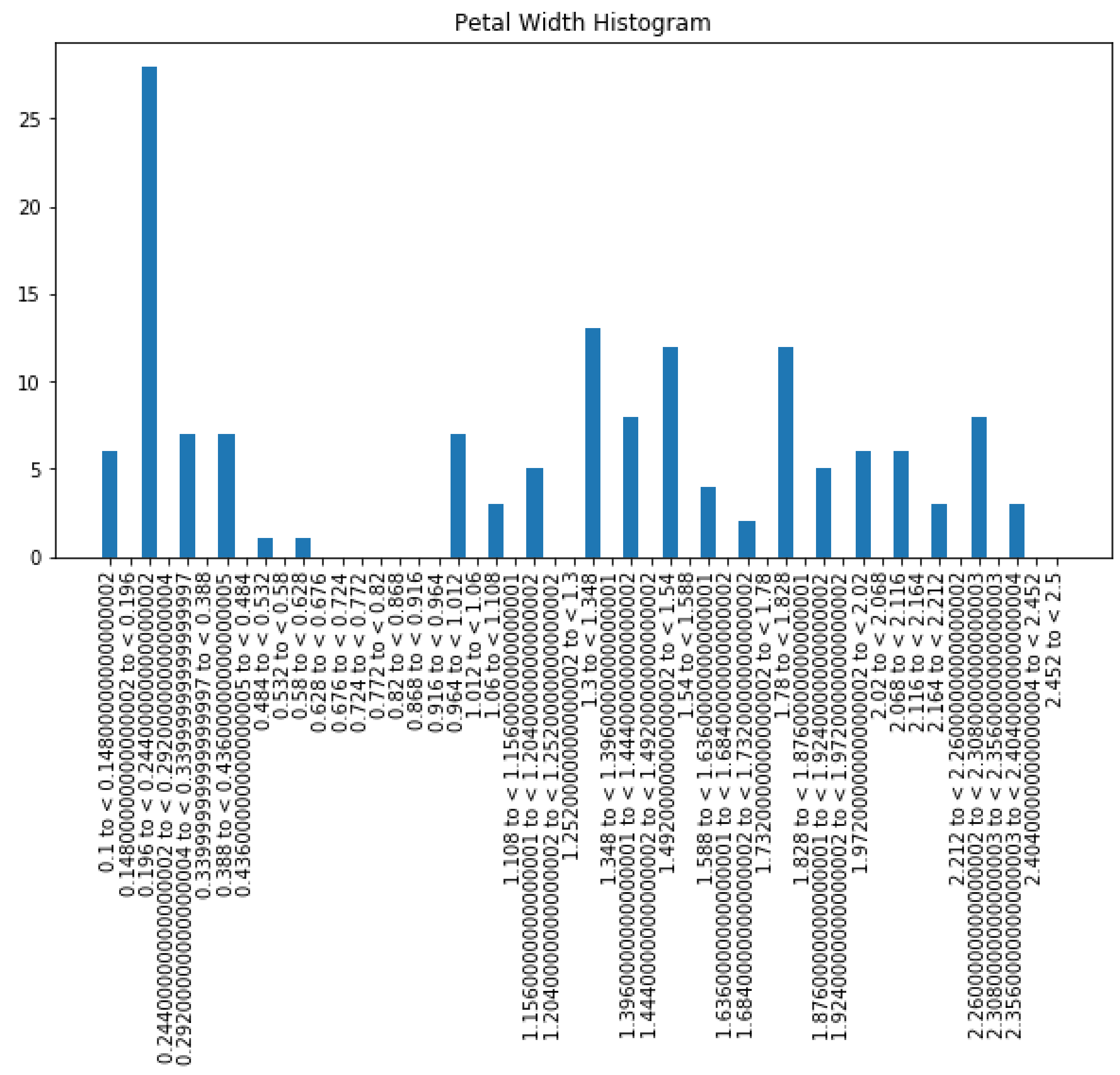

Petal Length Histogram
35
30
25
20
15
10
5
0
1.0 to < 1.59
1.59 to < 2.18
2.18 to < 2.7700000000000005
2.7700000000000005 to < 3.3600000000000003
3.3600000000000003 to < 3.95
3.95 to < 4.540000000000001
4.540000000000001 to < 5.130000000000001
5.130000000000001 to < 5.720000000000001
5.720000000000001 to < 6.3100000000000005
6.3100000000000005 to < 6.9


Petal Width Histogram
40
35
30
25
20
15
10
5
0
0.1 to < 0.33999999999999997
0.33999999999999997 to < 0.58
0.58 to < 0.82
0.82 to < 1.06
1.06 to < 1.3
1.3 to < 1.54
1.54 to < 1.78
1.78 to < 2.02
2.02 to < 2.2600000000000002
2.2600000000000002 to < 2.5

```
In [14]: n=20
         create_true_hist(petalwidth,n,'Petal Width Histogram')
```



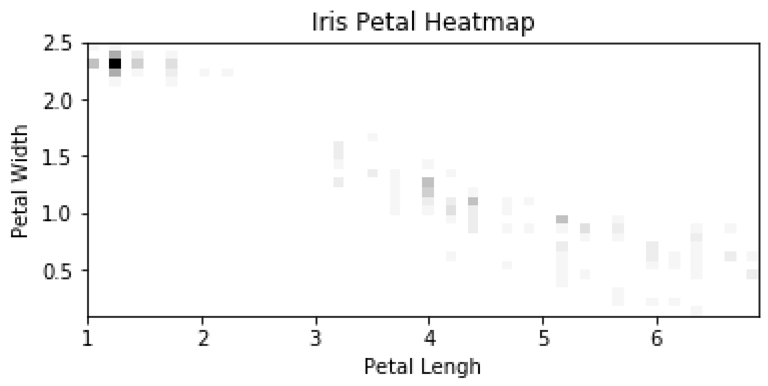
```
In [15]: n=50
         create_true_hist(petalwidth,n,'Petal Width Histogram')
```



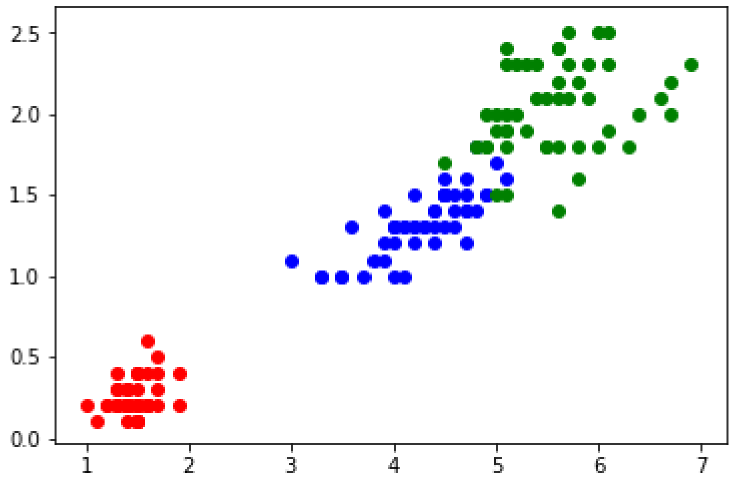
Here we have reached a point where we have more categories in the visualization than we have categories in the data. It is unlikely, then, that we will be able to separate the data for pedal width algorithmically into level distributions entirely without having attributes with the same measurement in multiple categories.

Just for fun, since I haven't done it before, let's look at a heat map comparing the quantities of data points using a heatmap with both variables.

In [16]:
```
def convert_to_numeric(data):
    for i in range(len(data)):
        for j in range(len(data[i])):
            try:
                data_point = float(data[i][j])
            except:
                data_point = 'na'
            if data_point != 'na':
                data[i][j] = data_point
    return(data)

def create_heatmap(dataset):
    # petal length will be x, at i[0]
    # petal width will be y, at i[1]
    headers = dataset[0]
    data = dataset[1:]
    x,y = [],[]
    for i in data:
        x.append(i[0])
        y.append(i[1])
    heatmap, xedges, yedges = np.histogram2d(x, y, bins=(30,60))
    extent = [xedges[0], xedges[-1], yedges[0], yedges[-1]]
    plt.clf()
    plt.title('Iris Petal Heatmap')
    plt.ylabel('Petal Width')
    plt.xlabel('Petal Lengh')
    plt.imshow(heatmap, extent=extent,cmap='Greys')
    plt.show()

converted_iris_data = convert_to_numeric(iris_data)
create_heatmap(converted_iris_data)
```



```
In [17]: def create_plot(dataset):
             headers = dataset[0]
             data = dataset[1:]
             x,y,c = [],[],[]
             unique_species = []
             for i in data:
                 x.append(i[0])
                 y.append(i[1])
                 if i[2] == 'Iris-setosa':
                     c.append('r')
                 elif i[2] == 'Iris-versicolor':
                     c.append('b')
                 else: c.append('g')
             for i in range(len(x)):
                 plt.scatter(x[i], y[i], color=c[i])
             plt.show()

         create_plot(converted_iris_data)
```



While my heatmap doesn't actually have the Y axis correct (it is inverted relative to the Y label), it does show something I'd never noticed before. In scatter plots, a point appearring exactly on top of another is hidden. A heatmap gets color values from this type of occurence, so looking at iris this way shows just how many values must be at 0.2 width and just over 1 length.

In conclusion, I will stand by my initial reaction that there is likely not a good way to divide petal width to completely unbundle the 0.2 measurement.