# D206 Performance Assessment By Krista Moik

In [1]:
```
#import all libraries
import pandas as pd
import math
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats
import seaborn as sns
```

In [2]:
```
#import CSV
df = pd.read_csv('C:/Users/Kmoik WGU/Desktop/medical_raw_data.csv')
```

In [3]:
```
#view the data
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 53 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   Unnamed: 0          10000 non-null  int64  
 1   CaseOrder           10000 non-null  int64  
 2   Customer_id         10000 non-null  object 
 3   Interaction         10000 non-null  object 
 4   UID                 10000 non-null  object 
 5   City                10000 non-null  object 
 6   State               10000 non-null  object 
 7   County              10000 non-null  object 
 8   Zip                 10000 non-null  int64  
 9   Lat                 10000 non-null  float64
 10  Lng                 10000 non-null  float64
 11  Population          10000 non-null  int64  
 12  Area                10000 non-null  object 
 13  Timezone            10000 non-null  object 
 14  Job                 10000 non-null  object 
 15  Children            7412 non-null   float64
 16  Age                 7586 non-null   float64
 17  Education           10000 non-null  object 
 18  Employment          10000 non-null  object 
 19  Income              7536 non-null   float64
 20  Marital             10000 non-null  object 
 21  Gender              10000 non-null  object 
 22  ReAdmis             10000 non-null  object 
 23  VitD_levels         10000 non-null  float64
 24  Doc_visits          10000 non-null  int64  
 25  Full_meals_eaten    10000 non-null  int64  
 26  VitD_supp           10000 non-null  int64  
 27  Soft_drink          7533 non-null   object 
 28  Initial_admin       10000 non-null  object 
 29  HighBlood           10000 non-null  object 
 30  Stroke              10000 non-null  object 
 31  Complication_risk   10000 non-null  object 
 32  Overweight          9018 non-null   float64
 33  Arthritis           10000 non-null  object 
 34  Diabetes            10000 non-null  object 
 35  Hyperlipidemia      10000 non-null  object 
 36  BackPain            10000 non-null  object 
 37  Anxiety             9016 non-null   float64
 38  Allergic_rhinitis   10000 non-null  object 
 39  Reflux_esophagitis  10000 non-null  object 
 40  Asthma              10000 non-null  object 
 41  Services            10000 non-null  object 
 42  Initial_days        8944 non-null   float64
 43  TotalCharge         10000 non-null  float64
 44  Additional_charges  10000 non-null  float64
 45  Item1               10000 non-null  int64  
 46  Item2               10000 non-null  int64  
 47  Item3               10000 non-null  int64  
 48  Item4               10000 non-null  int64  
 49  Item5               10000 non-null  int64  
 50  Item6               10000 non-null  int64  
 51  Item7               10000 non-null  int64  
 52  Item8               10000 non-null  int64  
dtypes: float64(11), int64(15), object(27)
memory usage: 4.0+ MB
```

This view of the data shows us we have 53 columns and 10,000 rows, which is correct. This view also shows that we do have NULL values that we will need to look at.

In [4]:
```
#Viewing the number of duplicates in the data
print(df.duplicated().value_counts())
```

```
False    10000
Name: count, dtype: int64
```

The output of the sum of duplicate shows we do not have any duplicate values.

In [5]:
```
# We know from our df.info() view there are NULL values.  We will search for the sum o
df.isnull().sum()
```

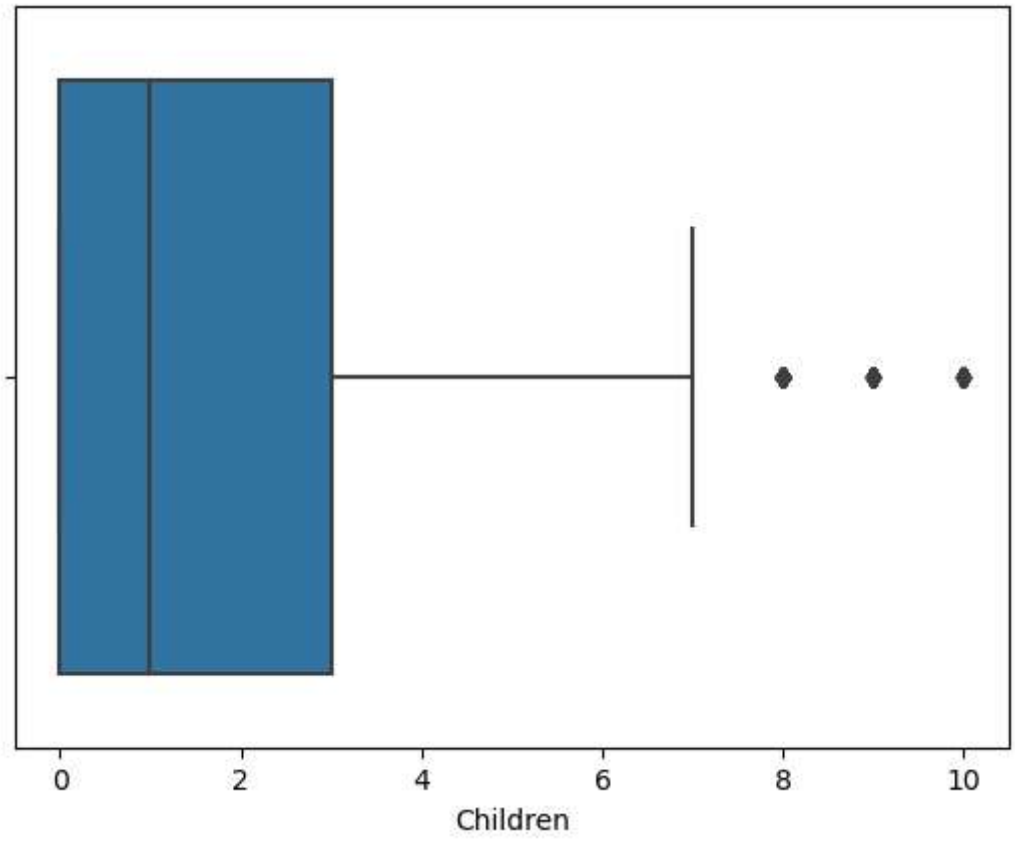
Out[5]:
```
Unnamed: 0              0
CaseOrder               0
Customer_id             0
Interaction             0
UID                     0
City                    0
State                   0
County                  0
Zip                     0
Lat                     0
Lng                     0
Population              0
Area                    0
Timezone                0
Job                     0
Children             2588
Age                  2414
Education               0
Employment              0
Income               2464
Marital                 0
Gender                  0
ReAdmis                 0
VitD_levels             0
Doc_visits              0
Full_meals_eaten        0
VitD_supp               0
Soft_drink           2467
Initial_admin           0
HighBlood               0
Stroke                  0
Complication_risk       0
Overweight            982
Arthritis               0
Diabetes                0
Hyperlipidemia          0
BackPain                0
Anxiety               984
Allergic_rhinitis       0
Reflux_esophagitis      0
Asthma                  0
Services                0
Initial_days         1056
TotalCharge             0
Additional_charges      0
Item1                   0
Item2                   0
Item3                   0
Item4                   0
Item5                   0
Item6                   0
Item7                   0
Item8                   0
dtype: int64
```

The output shows that there are 2588 NULL values for children, 2414 NULL values for Age, 2464 NULL values for Income, 2467 NULL values for soft_drink, 982 NULL values for overweight, 984 NULL values for anxiety, and 1056 NULL values for Initial_days.

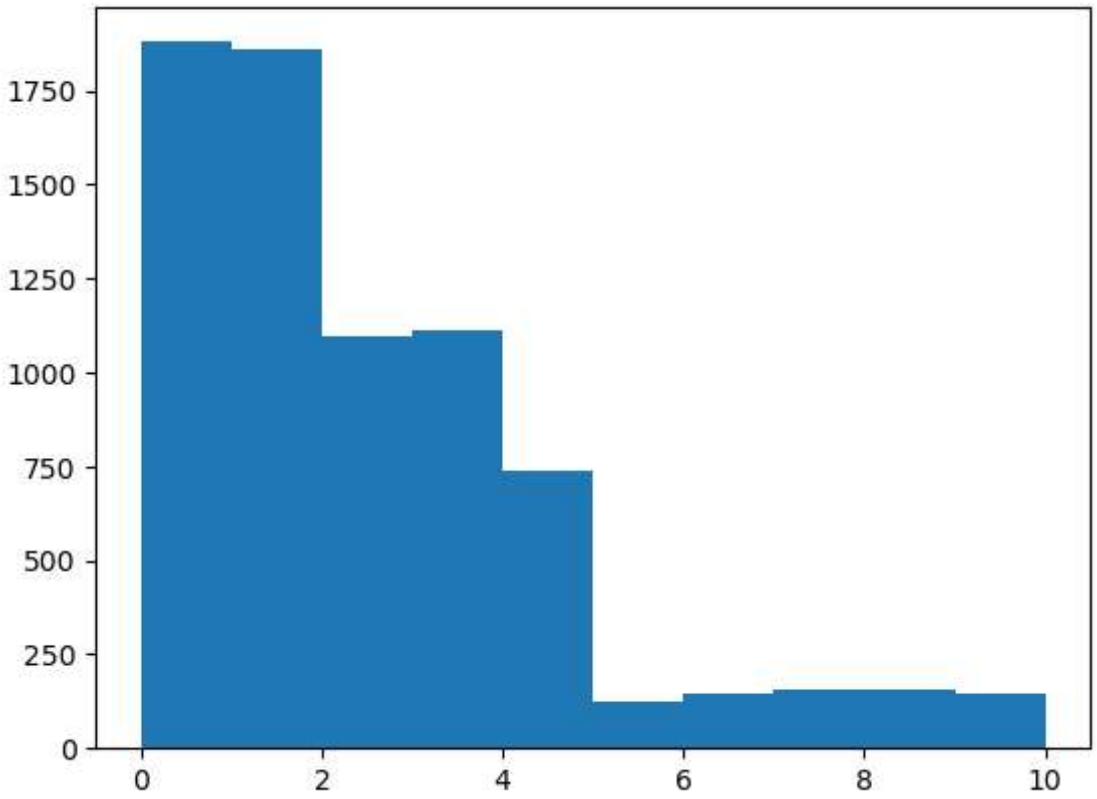
We will now examine the NULL values of the Quantitative columns - Children, Age, Income, and Initial_days using Boxplots and Histograms

In [6]:
```
#Boxplot for Children
sns.boxplot(df, x='Children')
```

Out[6]:
```
<Axes: xlabel='Children'>
```




In [7]:
```
#Histogram for Children
plt.hist(df['Children'])
plt.show()
```



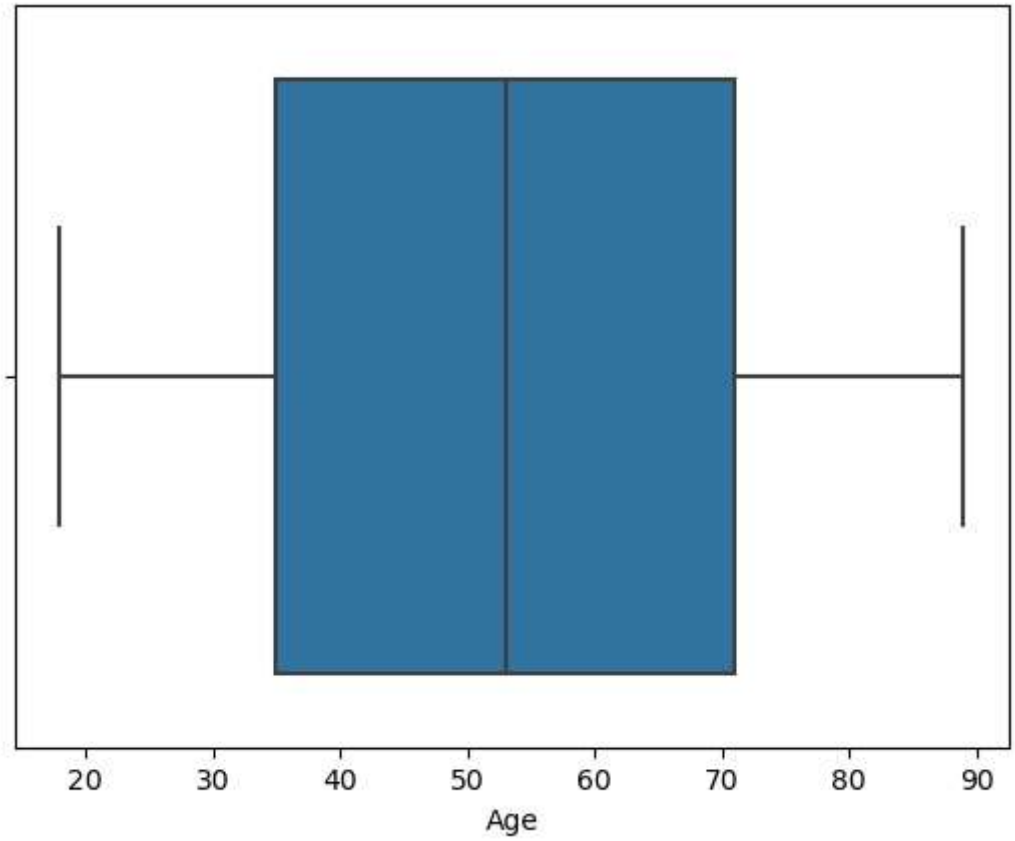

In [8]:
```
#obtain statistical information regarding the column
df.Children.describe()
```

Out[8]:
```
count    7412.000000
mean        2.098219
std         2.155427
min         0.000000
25%         0.000000
50%         1.000000
75%         3.000000
max        10.000000
Name: Children, dtype: float64
```

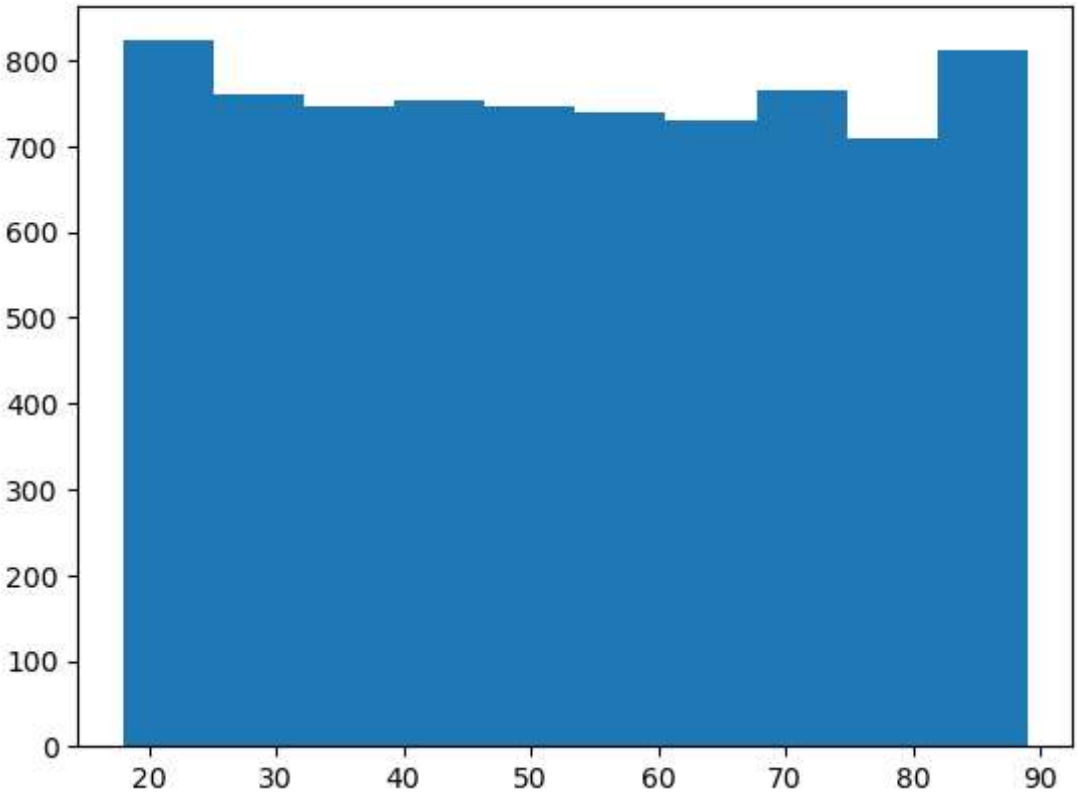
There appear to be 3 outliers in this column as well as NULL values. This histogram shows a right skew.

In [9]:
```
#Boxplot for Age
sns.boxplot(df, x='Age')
```

Out[9]:
```
<Axes: xlabel='Age'>
```




In [10]:
```
#Histogram for Age
plt.hist(df['Age'])
plt.show()
```



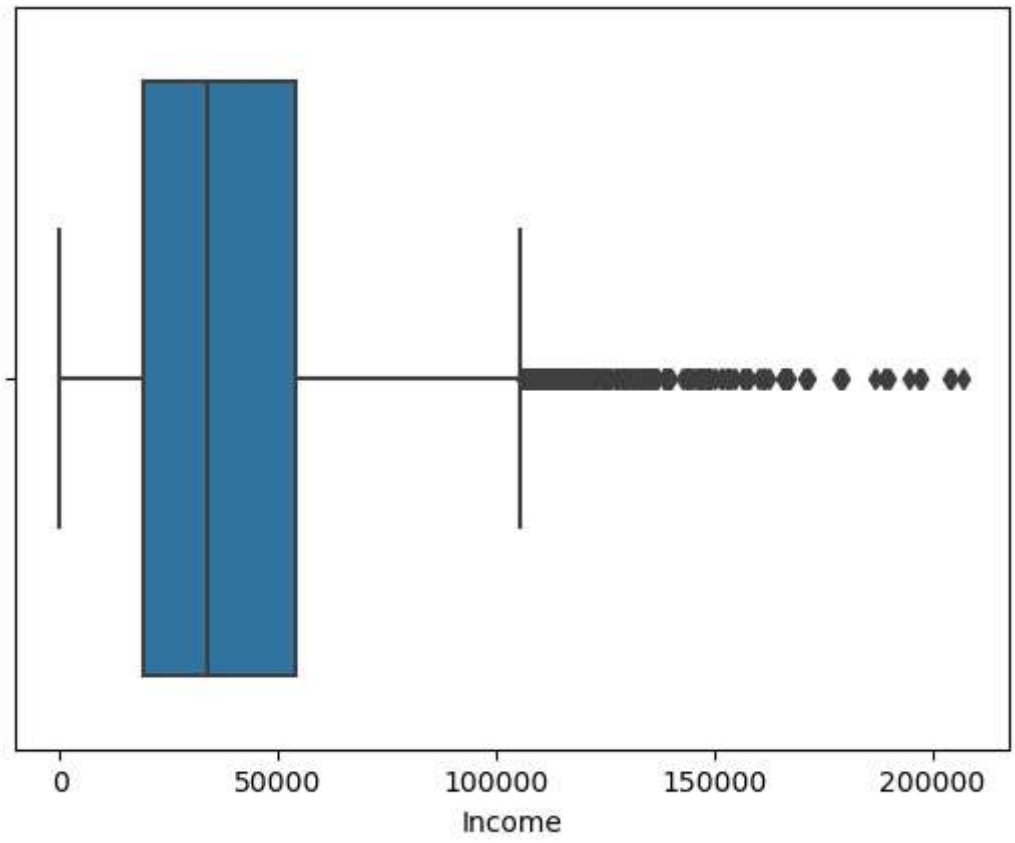

In [11]:
```
#obtain statistical information regarding the column
df.Age.describe()
```

Out[11]:
```
count    7586.000000
mean       53.295676
std        20.659182
min        18.000000
25%        35.000000
50%        53.000000
75%        71.000000
max        89.000000
Name: Age, dtype: float64
```

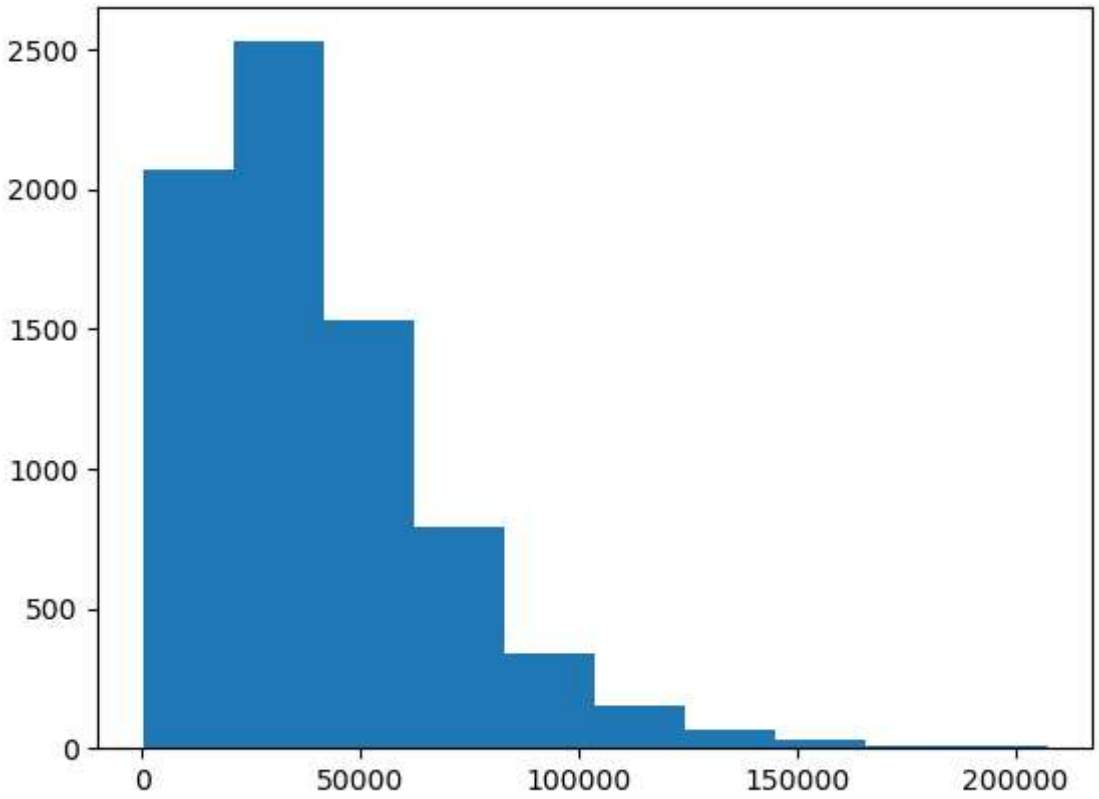
There do not appear to be any outliers in this column. This histogram shows a mostly uniform distribution.

In [12]:
```
#Boxplot for Income
sns.boxplot(df, x='Income')
```

Out[12]:
```
<Axes: xlabel='Income'>
```




In [13]:
```
#Histogram for Income
plt.hist(df['Income'])
plt.show()
```



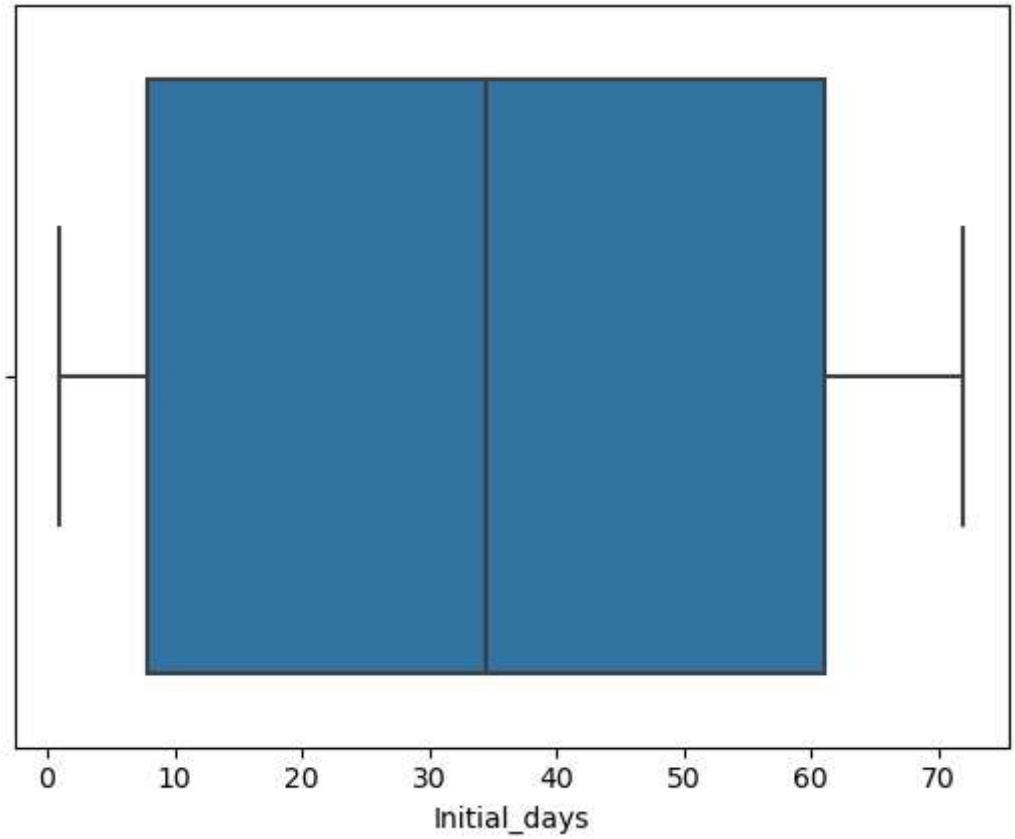

In [14]:
```
#Obtain statistical information for the column
df.Income.describe()
```

Out[14]:
```
count      7536.000000
mean      40484.438268
std       28664.861050
min         154.080000
25%       19450.792500
50%       33942.280000
75%       54075.235000
max      207249.130000
Name: Income, dtype: float64
```

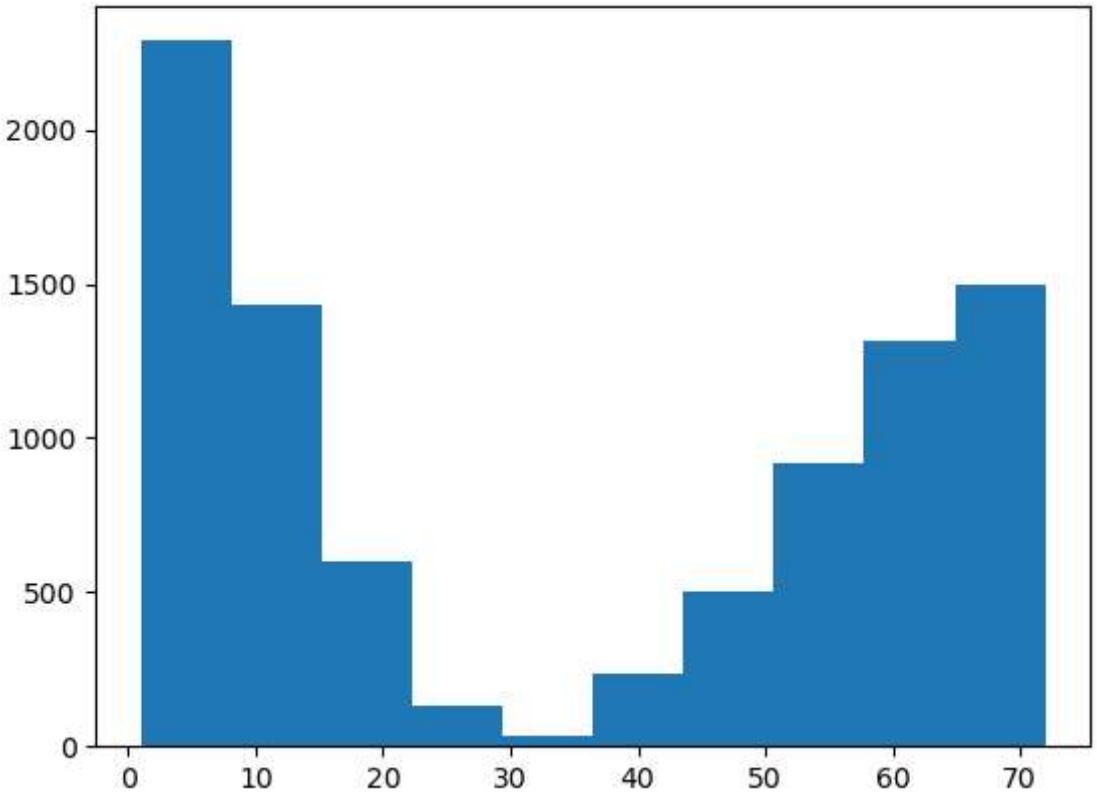
There appear to be many outliers in addition to NULL values in this column. This Histogram shows a right skew.

In [15]:
```
#Boxplot for Initial Days
sns.boxplot(df, x='Initial_days')
```

Out[15]:
```
<Axes: xlabel='Initial_days'>
```




In [16]:
```
#Histogram of Initial days
plt.hist(df['Initial_days'])
plt.show()
```




In [17]:
```
#Obtain statistical information about this column
df.Initial_days.describe()
```

Out[17]:
```
count    8944.000000
mean       34.432082
std        26.287050
min         1.001981
25%         7.911709
50%        34.446941
75%        61.124654
max        71.981486
Name: Initial_days, dtype: float64
```

There do not appear to be any outliers in this column. This histogram shows a bimodal skew.

We will now impute the NULL values before handling the outliers. We will use the mean for the normal distribution of Age, and the median for the right skew and bimodel columns.

In [18]:
```
#imputing null values median and mean with code found in WGU Course Materials
df['Children'].fillna(df['Children'].median(), inplace=True)
df['Income'].fillna(df['Income'].median(), inplace=True)
df['Initial_days'].fillna(df['Initial_days'].median(), inplace=True)
df['Age'].fillna(df['Age'].mean(), inplace=True)
```

In [19]:
```
#verify our imputation of NULL values in the quantitative columns
df.isnull().sum()
```

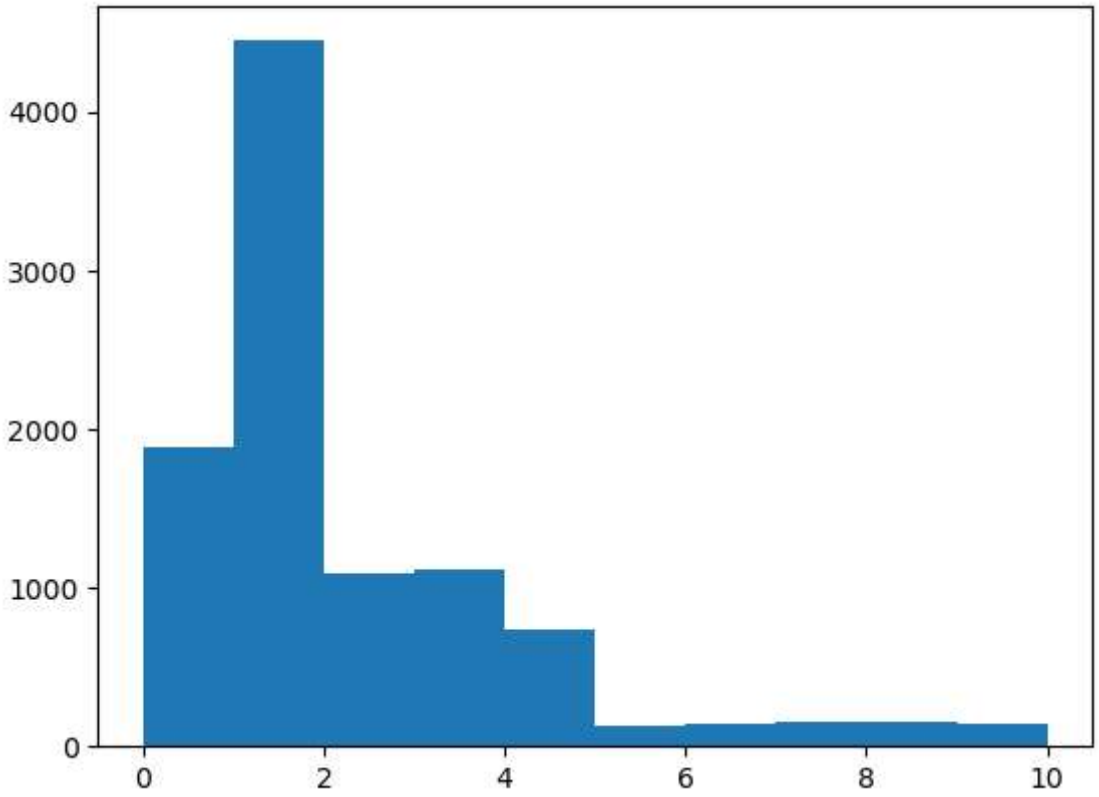
Out[19]:

```
Unnamed: 0              0
CaseOrder               0
Customer_id             0
Interaction             0
UID                     0
City                    0
State                   0
County                  0
Zip                     0
Lat                     0
Lng                     0
Population              0
Area                    0
Timezone                0
Job                     0
Children                0
Age                     0
Education               0
Employment              0
Income                  0
Marital                 0
Gender                  0
ReAdmis                 0
VitD_levels             0
Doc_visits              0
Full_meals_eaten        0
VitD_supp               0
Soft_drink           2467
Initial_admin           0
HighBlood               0
Stroke                  0
Complication_risk       0
Overweight            982
Arthritis               0
Diabetes                0
Hyperlipidemia          0
BackPain                0
Anxiety               984
Allergic_rhinitis       0
Reflux_esophagitis      0
Asthma                  0
Services                0
Initial_days            0
TotalCharge             0
Additional_charges      0
Item1                   0
Item2                   0
Item3                   0
Item4                   0
Item5                   0
Item6                   0
Item7                   0
Item8                   0
dtype: int64
```

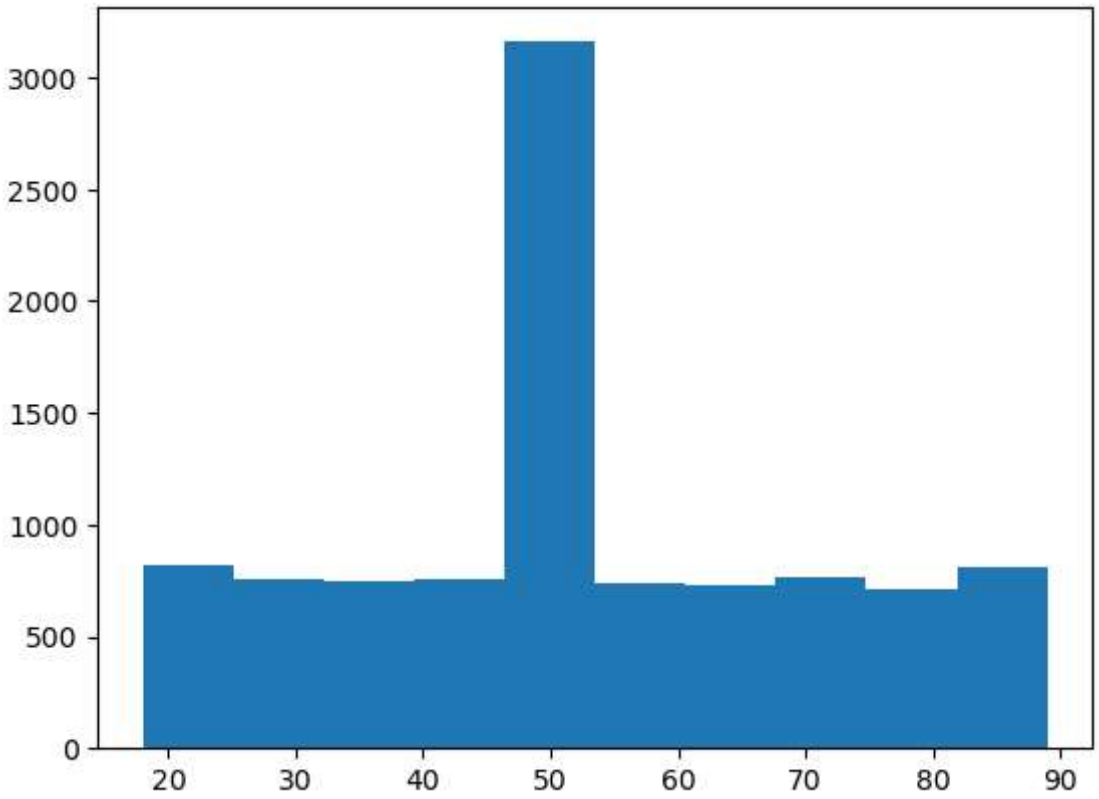
This shows our imputation of the quantitative columns was successful. The remaining NULLS are in qualitative columns.

Visual Confirmation Imputation of the NULLS was successful

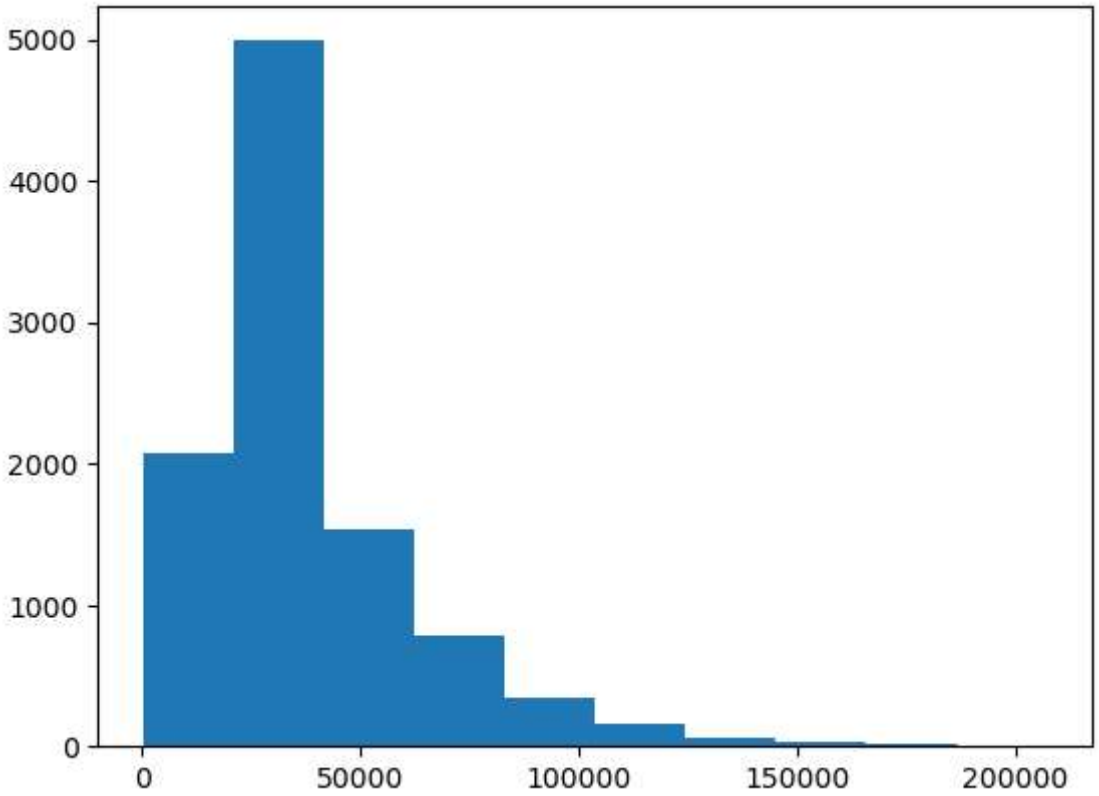
In [20]:
```
#Children column
plt.hist(df['Children'])
plt.show()
```




In [21]:
```
#Age column
plt.hist(df['Age'])
plt.show()
```

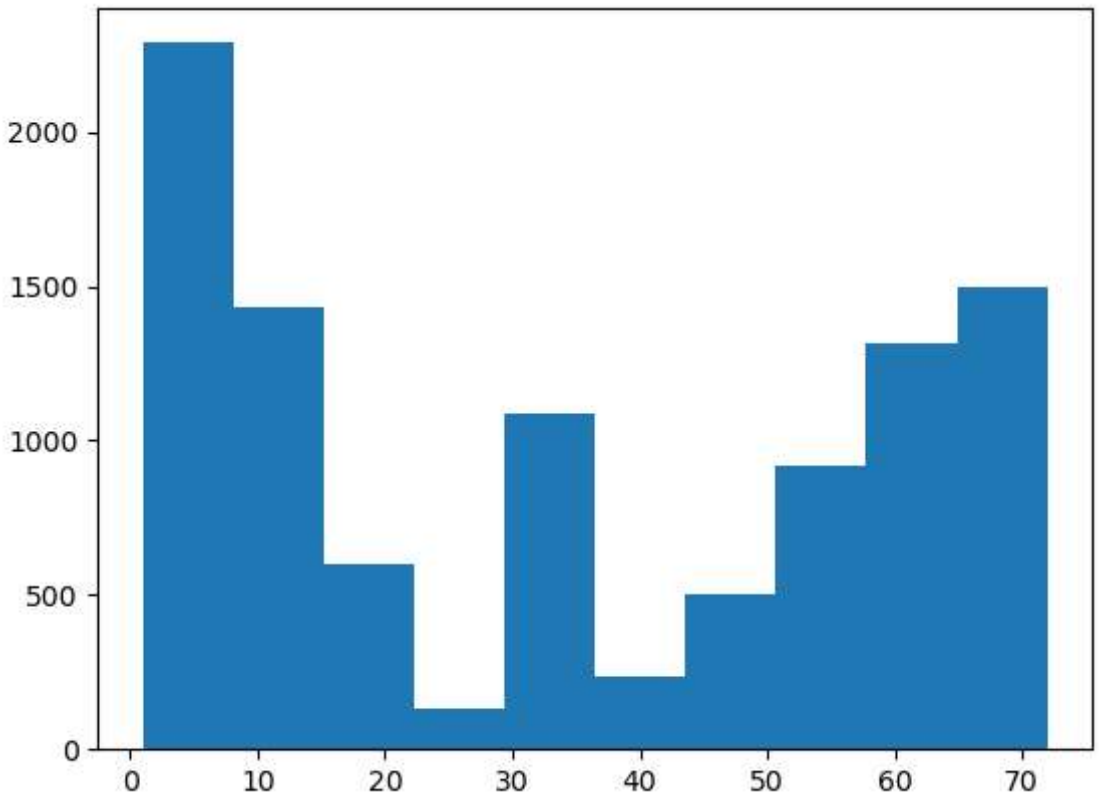



In [22]:
```
#Income Column
plt.hist(df['Income'])
plt.show()
```




In [23]:
```
#Initial_days column
plt.hist(df['Initial_days'])
```

```
plt.show()
```




Imputing NULLS in qualitative columns

In [24]:
```
# Find value counts excluding NULL values in Qualitative column Soft Drinks using code
df['Soft_drink'].value_counts(normalize=True, sort=True, dropna=True)
```

Out[24]:
```
Soft_drink
No     0.741935
Yes    0.258065
Name: proportion, dtype: float64
```

In [25]:
```
#Find value counts excluding NULL values in Qualitative column Overweight using code f
df['Overweight'].value_counts(normalize=True, sort=True, dropna=True)
```

Out[25]:
```
Overweight
1.0    0.709137
0.0    0.290863
Name: proportion, dtype: float64
```

In [26]:
```
#Find value counts excluded NULL values in Qualitative column Anxiety using code from
df['Anxiety'].value_counts(normalize=True, sort=True, dropna=True)
```

Out[26]:
```
Anxiety
0.0    0.677684
1.0    0.322316
Name: proportion, dtype: float64
```

Now that we have the count of the yes/no 0/1 responses and because these values are categorical, we will replace the NULL values with the most common response in each column.

In [27]:
```
# 0 is the most common response for Soft drinks so we will impute the NULLS to No as t
df['Soft_drink'].fillna('No', inplace=True)
```

In [28]:
```
# Using Ordinal encoding - Replicate the variable in preparation for replacing its cat
df['Soft_drink_numeric'] = df['Soft_drink']
```

In [29]:
```
# Set up a dictionary specifically for converting the categorical values to numeric va
dict_soft_drink = {'Soft_drink_numeric': {'No': 0, 'Yes': 1}}
```

In [30]:
```
# Use the dictionary to replace the variable's values. The replace() function will rep
df.replace(dict_soft_drink, inplace=True)
```

In [31]:
```
#check to see if our dictionary worked for soft_drink_numeric using code from WGU cour
df['Soft_drink_numeric'].value_counts(normalize=True, sort=True, dropna=True)
```

Out[31]:
```
Soft_drink_numeric
0    0.8056
1    0.1944
Name: proportion, dtype: float64
```

We can see our Ordinal encoding was successful. We will now continue replacing the remaining NULLS

In [32]:
```
# 1 is the most common response for Overweight so we will impute the NULLS to 1 using
df['Overweight'].fillna(1, inplace=True)
```

In [33]:
```
# 0 is the most common response for Anxiety so we will impute the NULLS to 0 using coa
df['Anxiety'].fillna(0, inplace=True)
```

In [34]:
```
# View dataset to verify all NULLS have been addressed
df.info()
```

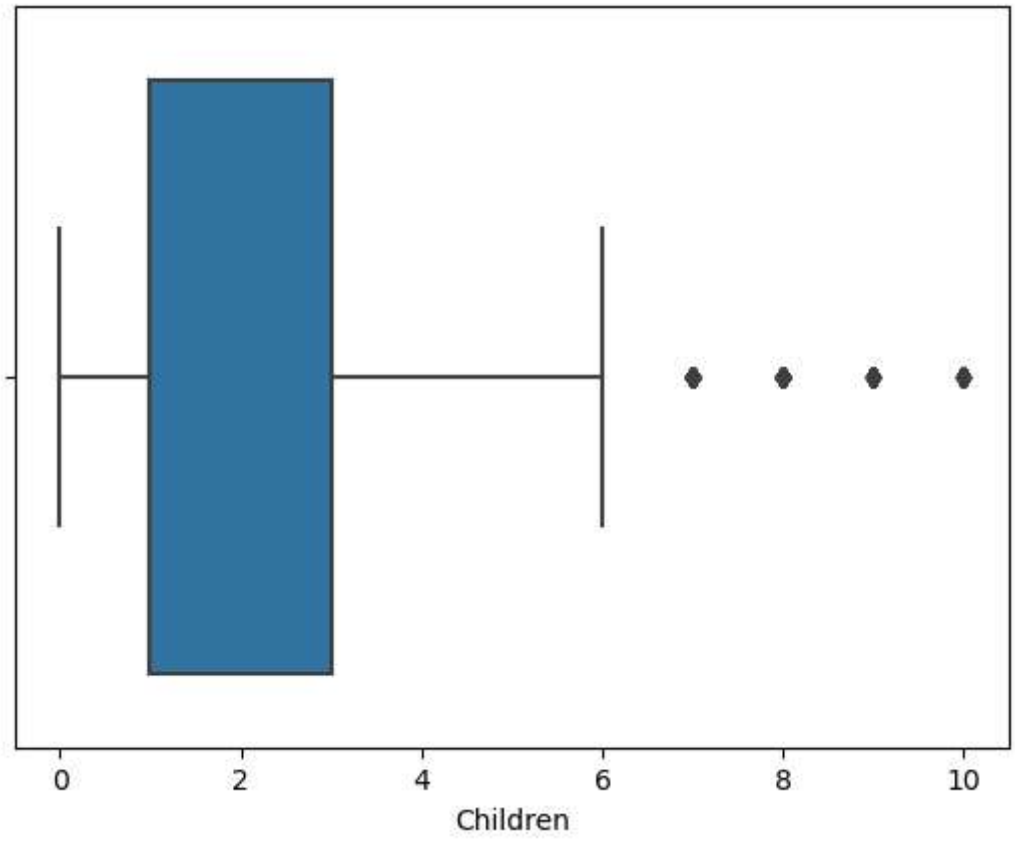
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 54 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   Unnamed: 0          10000 non-null  int64  
 1   CaseOrder           10000 non-null  int64  
 2   Customer_id         10000 non-null  object 
 3   Interaction         10000 non-null  object 
 4   UID                 10000 non-null  object 
 5   City                10000 non-null  object 
 6   State               10000 non-null  object 
 7   County              10000 non-null  object 
 8   Zip                 10000 non-null  int64  
 9   Lat                 10000 non-null  float64
 10  Lng                 10000 non-null  float64
 11  Population          10000 non-null  int64  
 12  Area                10000 non-null  object 
 13  Timezone            10000 non-null  object 
 14  Job                 10000 non-null  object 
 15  Children            10000 non-null  float64
 16  Age                 10000 non-null  float64
 17  Education           10000 non-null  object 
 18  Employment          10000 non-null  object 
 19  Income              10000 non-null  float64
 20  Marital             10000 non-null  object 
 21  Gender              10000 non-null  object 
 22  ReAdmis             10000 non-null  object 
 23  VitD_levels         10000 non-null  float64
 24  Doc_visits          10000 non-null  int64  
 25  Full_meals_eaten    10000 non-null  int64  
 26  VitD_supp           10000 non-null  int64  
 27  Soft_drink          10000 non-null  object 
 28  Initial_admin       10000 non-null  object 
 29  HighBlood           10000 non-null  object 
 30  Stroke              10000 non-null  object 
 31  Complication_risk   10000 non-null  object 
 32  Overweight          10000 non-null  float64
 33  Arthritis           10000 non-null  object 
 34  Diabetes            10000 non-null  object 
 35  Hyperlipidemia      10000 non-null  object 
 36  BackPain            10000 non-null  object 
 37  Anxiety             10000 non-null  float64
 38  Allergic_rhinitis   10000 non-null  object 
 39  Reflux_esophagitis  10000 non-null  object 
 40  Asthma              10000 non-null  object 
 41  Services            10000 non-null  object 
 42  Initial_days        10000 non-null  float64
 43  TotalCharge         10000 non-null  float64
 44  Additional_charges  10000 non-null  float64
 45  Item1               10000 non-null  int64  
 46  Item2               10000 non-null  int64  
 47  Item3               10000 non-null  int64  
 48  Item4               10000 non-null  int64  
 49  Item5               10000 non-null  int64  
 50  Item6               10000 non-null  int64  
 51  Item7               10000 non-null  int64  
 52  Item8               10000 non-null  int64  
 53  Soft_drink_numeric  10000 non-null  int64  
```

```
dtypes: float64(11), int64(16), object(27)
memory usage: 4.1+ MB
```

There are no more NULL values in our dataset. We will now move on to outliers. We will obtain updated boxplots of the variables with outliers - Children and Income to see how they might have changed after imputation

In [35]:
```
#Boxplot for Children
sns.boxplot(df, x='Children')
```

Out[35]:
```
<Axes: xlabel='Children'>
```

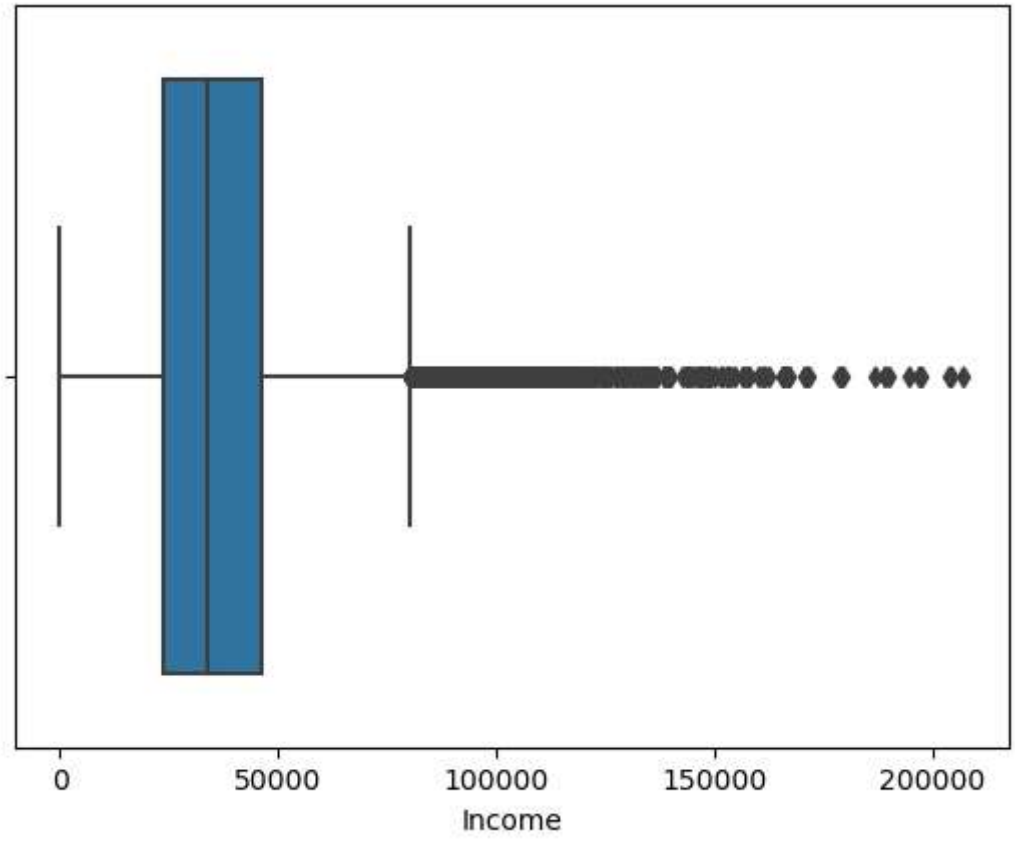



We can now see there appear to be 4 outliers in the number of children per patient - those patients who have 7-10 children are the outlier.

In [36]:
```
#Obtain count of how many patients had >6 children
df['Children'][df['Children']>6].count()
```

Out[36]:
```
457
```

In [37]:
```
#Box plot for Income
sns.boxplot(df, x='Income')
whisker_lines=plt.gca().lines
```




In [38]:
```
#Obtain value of upper_whisker
upper_whisker=whisker_lines[3].get_xdata()[0]
print(upper_whisker)
```

```
80229.88
```

This shows that the outliers are patients who have an income greater than $80,229

1. However, there are still too many outliers to be able to count with the boxplot, so we will need to use the count() function

In [39]:
```
#Counting outliers in Income
df['Income'][df['Income'] > 80229.88].count()
```

Out[39]:
```
705
```

The output shows us that there are 704 outliers, or patients with income greater than $80229.88, in the Income column.

We will now review the statistiscs of Children and income

In [40]:
```
df.Children.describe()
```

Out[40]:
```
count    10000.000000
mean         1.814000
std          1.916969
min          0.000000
25%          1.000000
50%          1.000000
75%          3.000000
max         10.000000
Name: Children, dtype: float64
```

In [41]:
```
df.Income.describe()
```

Out[41]:
```
count     10000.000000
mean      38872.450471
std       25042.796229
min         154.080000
25%       23956.162500
50%       33942.280000
75%       46466.797500
max      207249.130000
Name: Income, dtype: float64
```

Determining outliers involves determining if the values are acceptable or not. While 10 children may seem like a lot, it is not unheard of. Additionally, $207,249 income is a lot but also not unreasonable. At this time, without more information, I am choosing to retain all outliers in my dataset as I find the ranges to be reasonable.

We will now move on to apply Principal Component Analysis (PCA) to identify the significant features of the data set.

In [42]:
```
#import additional package for PCA
from sklearn.decomposition import PCA
```

We will normalize all quantitative columns and store them in df_pca

In [43]:
```
#storing quantitative columns in df_pca using code from WGU course materials.
df_pca=df[['Population','Children','Age','Income','VitD_levels','Doc_visits','Full_mea
```

In [44]:
```
# Normalize the columns by mean divided by standard deviation using code from WGU cour
df_pca_normalized = (df_pca-df_pca.mean())/df_pca.std()
```

In [45]:
```
#PCA using code from WGU course materials.
pca=PCA(n_components=df_pca.shape[1])
print(pca)
```

```
PCA(n_components=11)
```

We can see there are 11 principal components

In [46]:
```
#Normalizing PCA using code from WGU course materials.
pca.fit(df_pca_normalized)
```

Out[46]:
```
PCA(n_components=11)
```

In [47]:
```
#PCA Normatlization continued using code from WGU course materials.
df_pca=pd.DataFrame(pca.transform(df_pca_normalized), columns=['PC1','PC2','PC3','PC4'
```

In [48]:
```
#PCA Loadings using code from WGU course materials.
pca_loadings=pd.DataFrame(pca.components_.T, columns=['PC1','PC2','PC3','PC4','PC5','P
```

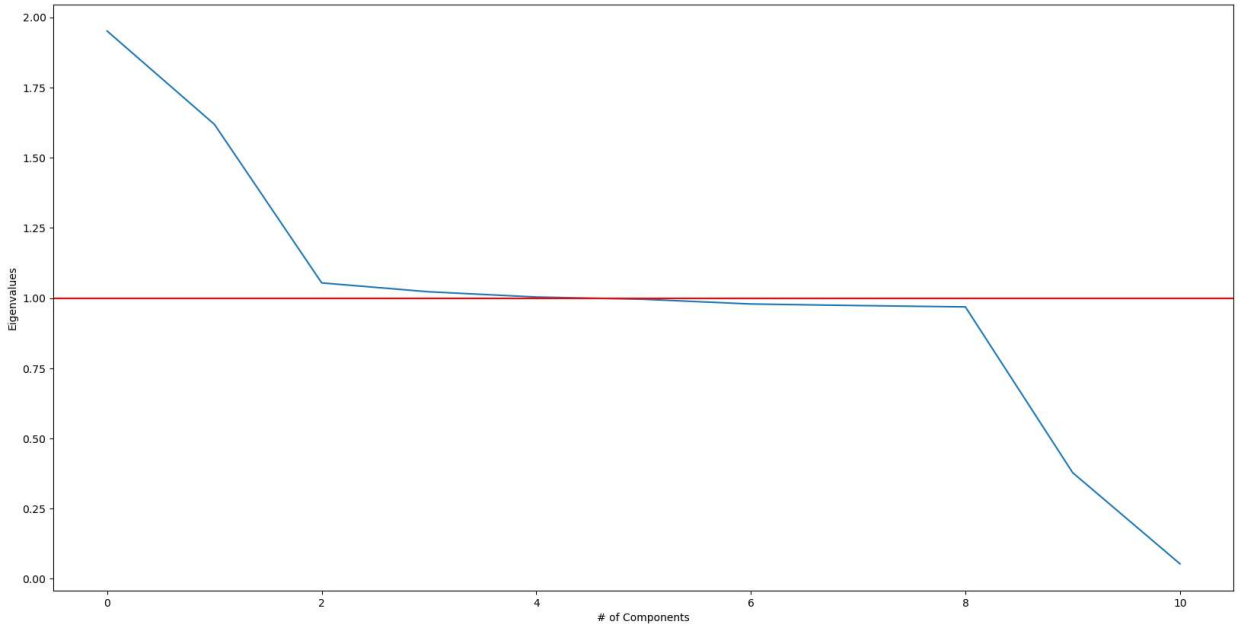
In [49]:
```
#print Loadings using code from WGU course materials.
pca_loadings
```

Out[49]:

| | PC1 | PC2 | PC3 | PC4 | PC5 | PC6 | PC7 | P |
|---|---|---|---|---|---|---|---|---|
| **Population** | 0.020664 | -0.027012 | 0.503294 | 0.011609 | -0.021267 | 0.545859 | 0.155000 | 0.2929 |
| **Children** | 0.004321 | 0.011337 | 0.143254 | -0.055997 | 0.886951 | 0.143688 | 0.277812 | -0.2468 |
| **Age** | 0.083046 | 0.700956 | 0.022109 | -0.024584 | -0.014235 | 0.004055 | -0.015253 | 0.0219 |
| **Income** | -0.006759 | -0.005324 | 0.153030 | 0.616420 | 0.311495 | -0.335102 | -0.300044 | 0.5449 |
| **VitD_levels** | 0.540331 | -0.052887 | -0.291266 | 0.267675 | -0.069787 | 0.088190 | 0.467171 | 0.1360 |
| **Doc_visits** | -0.005253 | 0.012817 | 0.175189 | 0.626189 | -0.179331 | 0.406596 | -0.216702 | -0.5299 |
| **Full_meals_eaten** | -0.009220 | 0.036698 | -0.552469 | 0.164700 | 0.231970 | -0.008979 | -0.227844 | -0.2773 |
| **VitD_supp** | 0.033965 | 0.010677 | 0.424358 | 0.159839 | -0.126966 | -0.621372 | 0.382475 | -0.3909 |
| **Initial_days** | 0.446473 | -0.073504 | 0.316725 | -0.315729 | 0.093103 | -0.098161 | -0.587243 | -0.1618 |
| **TotalCharge** | 0.702174 | -0.078309 | -0.023420 | 0.003263 | 0.002287 | 0.005498 | -0.016137 | 0.0025 |
| **Additional_charges** | 0.083681 | 0.701297 | 0.023997 | -0.004434 | -0.000742 | 0.011580 | -0.003613 | 0.0212 |

In [50]:
```
#obtaining eigenvalues using code from WGU course materials.
covariance_matrix = np.dot(df_pca_normalized.T, df_pca_normalized) / df_pca.shape[0]
eigenvalues = [np.dot(eigenvector.T, np.dot(covariance_matrix, eigenvector)) for eiger
```

In [51]:
```
#plotting the eigenvalues using code from WGU course materials.
plt.figure(figsize = [20,10])
plt.plot(eigenvalues)
plt.xlabel('# of Components')
plt.ylabel('Eigenvalues')
plt.axhline(y=1, color='red')
plt.show()
```




In [52]:
```
#list eigenvalue values using code from WGU course materials.
eigenvalues
```

Out[52]:
```
[1.9511903999693965,
 1.6188378421815532,
 1.0541224873896506,
 1.0224182228569607,
 1.0037755363342356,
 0.9958246800154101,
 0.9790089455874965,
 0.9733461883648581,
 0.9686663365082094,
 0.3781461270225378,
 0.05356323376974442]
```

This shows us that the first 5 Principal Components have a value greater than 1

In [53]:
```
#confirm data set as cleaned
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 54 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   Unnamed: 0          10000 non-null  int64  
 1   CaseOrder           10000 non-null  int64  
 2   Customer_id         10000 non-null  object 
 3   Interaction         10000 non-null  object 
 4   UID                 10000 non-null  object 
 5   City                10000 non-null  object 
 6   State               10000 non-null  object 
 7   County              10000 non-null  object 
 8   Zip                 10000 non-null  int64  
 9   Lat                 10000 non-null  float64
 10  Lng                 10000 non-null  float64
 11  Population          10000 non-null  int64  
 12  Area                10000 non-null  object 
 13  Timezone            10000 non-null  object 
 14  Job                 10000 non-null  object 
 15  Children            10000 non-null  float64
 16  Age                 10000 non-null  float64
 17  Education           10000 non-null  object 
 18  Employment          10000 non-null  object 
 19  Income              10000 non-null  float64
 20  Marital             10000 non-null  object 
 21  Gender              10000 non-null  object 
 22  ReAdmis             10000 non-null  object 
 23  VitD_levels         10000 non-null  float64
 24  Doc_visits          10000 non-null  int64  
 25  Full_meals_eaten    10000 non-null  int64  
 26  VitD_supp           10000 non-null  int64  
 27  Soft_drink          10000 non-null  object 
 28  Initial_admin       10000 non-null  object 
 29  HighBlood           10000 non-null  object 
 30  Stroke              10000 non-null  object 
 31  Complication_risk   10000 non-null  object 
 32  Overweight          10000 non-null  float64
 33  Arthritis           10000 non-null  object 
 34  Diabetes            10000 non-null  object 
 35  Hyperlipidemia      10000 non-null  object 
 36  BackPain            10000 non-null  object 
 37  Anxiety             10000 non-null  float64
 38  Allergic_rhinitis   10000 non-null  object 
 39  Reflux_esophagitis  10000 non-null  object 
 40  Asthma              10000 non-null  object 
 41  Services            10000 non-null  object 
 42  Initial_days        10000 non-null  float64
 43  TotalCharge         10000 non-null  float64
 44  Additional_charges  10000 non-null  float64
 45  Item1               10000 non-null  int64  
 46  Item2               10000 non-null  int64  
 47  Item3               10000 non-null  int64  
 48  Item4               10000 non-null  int64  
 49  Item5               10000 non-null  int64  
 50  Item6               10000 non-null  int64  
 51  Item7               10000 non-null  int64  
 52  Item8               10000 non-null  int64  
 53  Soft_drink_numeric  10000 non-null  int64  
```

```
dtypes: float64(11), int64(16), object(27)
memory usage: 4.1+ MB
```

In [54]:
```
# Create copy of data set to perform remaining cleaning steps on
clean_df = df.copy(deep=True)
```

In [56]:
```
clean_df.to_csv('C:/Users/Kmoik WGU/Desktop/KMoikclean_medical.csv') #This is showing
```

```
---------------------------------------------------------------------------
PermissionError                           Traceback (most recent call last)
Cell In[56], line 1
----> 1 clean_df.to_csv('C:/Users/Kmoik WGU/Desktop/KMoikclean_medical.csv')

File ~\anaconda3\Lib\site-packages\pandas\core\generic.py:3772, in NDFrame.to_csv(self, path_or_buf, sep, na_rep, float_format, columns, header, index, index_label, mode, encoding, compression, quoting, quotechar, lineterminator, chunksize, date_format, doublequote, escapechar, decimal, errors, storage_options)
   3761 df = self if isinstance(self, ABCDataFrame) else self.to_frame()
   3763 formatter = DataFrameFormatter(
   3764     frame=df,
   3765     header=header,
   (...)
   3769     decimal=decimal,
   3770 )
-> 3772 return DataFrameRenderer(formatter).to_csv(
   3773     path_or_buf,
   3774     lineterminator=lineterminator,
   3775     sep=sep,
   3776     encoding=encoding,
   3777     errors=errors,
   3778     compression=compression,
   3779     quoting=quoting,
   3780     columns=columns,
   3781     index_label=index_label,
   3782     mode=mode,
   3783     chunksize=chunksize,
   3784     quotechar=quotechar,
   3785     date_format=date_format,
   3786     doublequote=doublequote,
   3787     escapechar=escapechar,
   3788     storage_options=storage_options,
   3789 )

File ~\anaconda3\Lib\site-packages\pandas\io\formats\format.py:1186, in DataFrameRenderer.to_csv(self, path_or_buf, encoding, sep, columns, index_label, mode, compression, quoting, quotechar, lineterminator, chunksize, date_format, doublequote, escapechar, errors, storage_options)
   1165     created_buffer = False
   1167 csv_formatter = CSVFormatter(
   1168     path_or_buf=path_or_buf,
   1169     lineterminator=lineterminator,
   (...)
   1184     formatter=self.fmt,
   1185 )
-> 1186 csv_formatter.save()
   1188 if created_buffer:
   1189     assert isinstance(path_or_buf, StringIO)

File ~\anaconda3\Lib\site-packages\pandas\io\formats\csvs.py:240, in CSVFormatter.save(self)
    236 """
    237 Create the writer & save.
    238 """
    239 # apply compression and byte/text conversion
--> 240 with get_handle(
    241     self.filepath_or_buffer,
    242     self.mode,
    243     encoding=self.encoding,
```

```
    244     errors=self.errors,
    245     compression=self.compression,
    246     storage_options=self.storage_options,
    247 ) as handles:
    248     # Note: self.encoding is irrelevant here
    249     self.writer = csvlib.writer(
    250         handles.handle,
    251         lineterminator=self.lineterminator,
   (...)
    256         quotechar=self.quotechar,
    257     )
    259     self._save()

File ~\anaconda3\Lib\site-packages\pandas\io\common.py:859, in get_handle(path_or_buf, mode, encoding, compression, memory_map, is_text, errors, storage_options)
    854 elif isinstance(handle, str):
    855     # Check whether the filename is to be opened in binary mode.
    856     # Binary mode does not support 'encoding' and 'newline'.
    857     if ioargs.encoding and "b" not in ioargs.mode:
    858         # Encoding
--> 859         handle = open(
    860             handle,
    861             ioargs.mode,
    862             encoding=ioargs.encoding,
    863             errors=errors,
    864             newline="",
    865         )
    866     else:
    867         # Binary mode
    868         handle = open(handle, ioargs.mode)

PermissionError: [Errno 13] Permission denied: 'C:/Users/Kmoik WGU/Desktop/KMoikclean_medical.csv'
```

In [ ]: