mid_proj

May 4, 2024

1 Data Processing

[ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import entropy

df = pd.read_csv("heart_disease_health_indicators_BRFSS2015.csv")
```

1.0.1 construct Y column

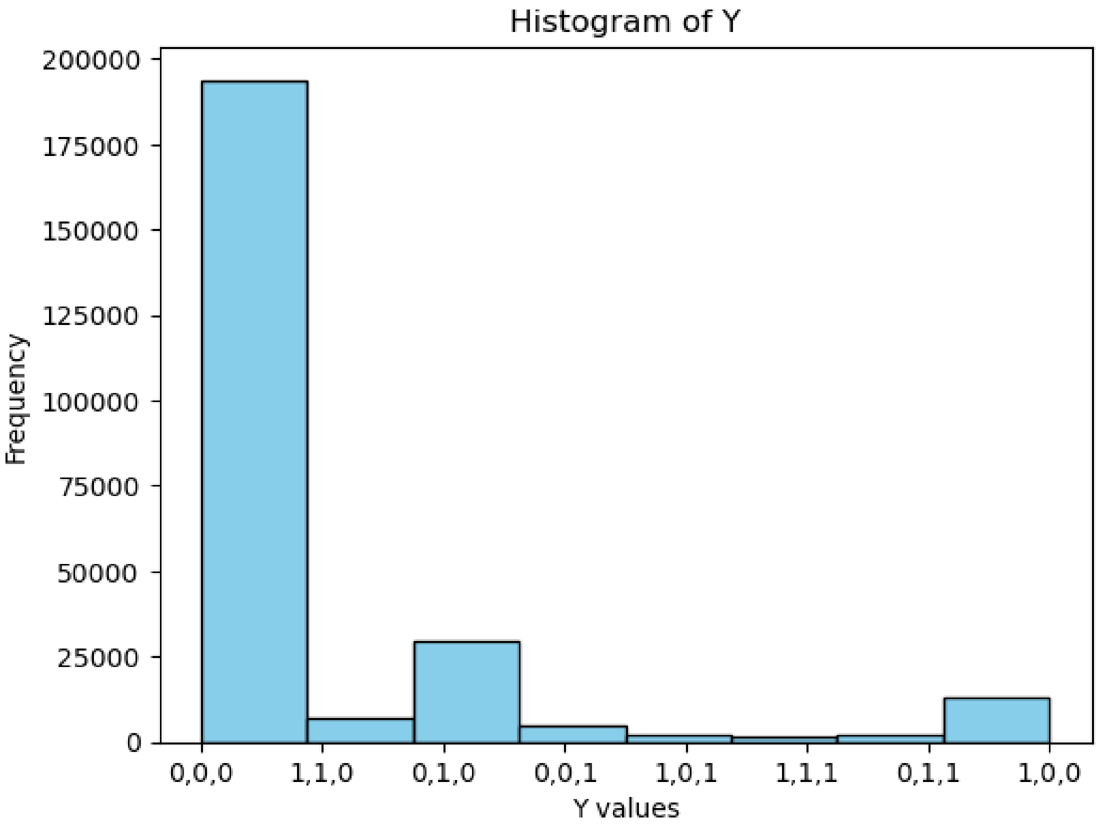
[ ]:
```
df['Diabetes'] = df['Diabetes'].replace(2, 1)
df['Y'] = df['HeartDiseaseorAttack'].astype(int).astype(str) + ',' +␣
 ↪df['Diabetes'].astype(int).astype(str) + ',' + df['Stroke'].astype(int).
 ↪astype(str)
```

[ ]:
```
print("Dimensions of the dataset : ", df.shape)
df['Y'].value_counts()
```

```
Dimensions of the dataset :  (253680, 23)
```

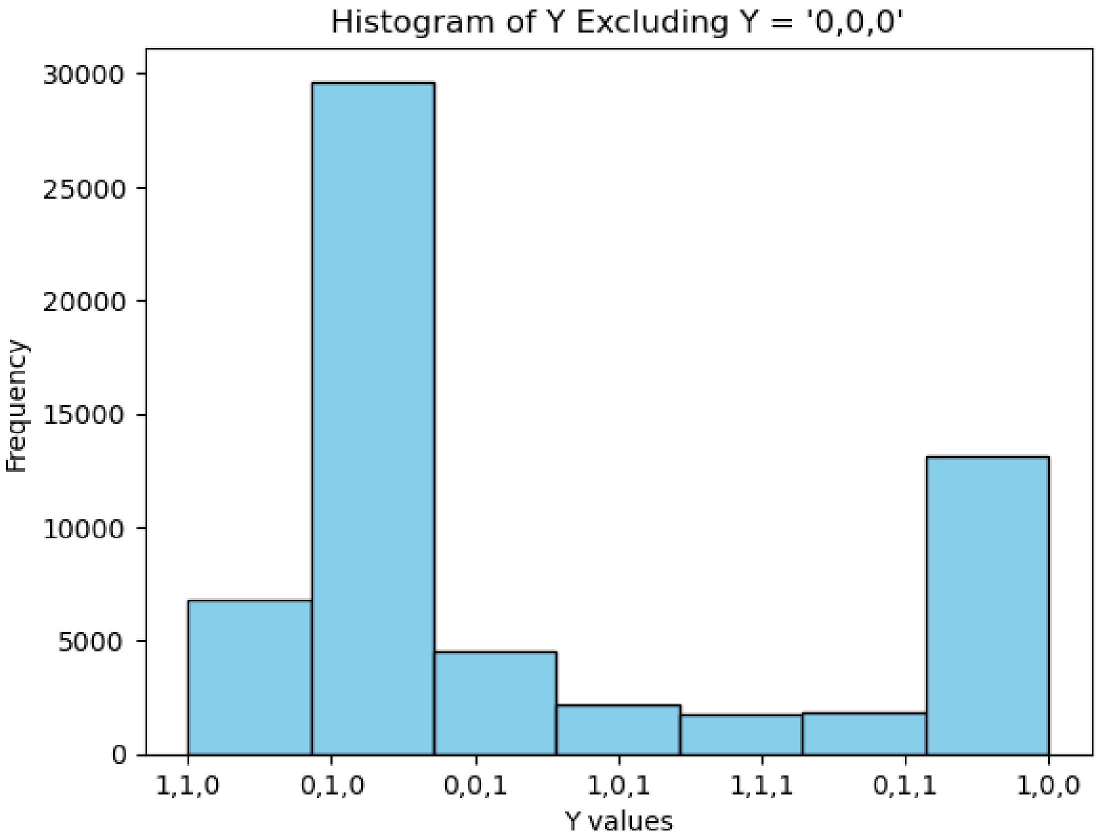
[ ]:
```
Y
0,0,0    193793
0,1,0     29639
1,0,0     13151
1,1,0      6805
0,0,1      4559
1,0,1      2200
0,1,1      1796
1,1,1      1737
Name: count, dtype: int64
```

[ ]:
```
plt.hist(df['Y'], bins=8, color='skyblue', edgecolor='black')
plt.title('Histogram of Y')
plt.xlabel('Y values')
plt.ylabel('Frequency')
plt.show()
```



[ ]:

```
df_no000 = df[df['Y'] != '0,0,0']
plt.hist(df_no000['Y'], bins=7, color='skyblue', edgecolor='black')
plt.title('Histogram of Y Excluding Y = \'0,0,0\'')
plt.xlabel('Y values')
plt.ylabel('Frequency')
plt.show()
```



```
[ ]: df['Age'].unique()
```

```
[ ]: array([ 9.,  7., 11., 10.,  8., 13.,  4.,  6.,  2., 12.,  5.,  1.,  3.])
```

```
[ ]: print(df.columns)
```

```
Index(['HeartDiseaseorAttack', 'HighBP', 'HighChol', 'CholCheck', 'BMI',
       'Smoker', 'Stroke', 'Diabetes', 'PhysActivity', 'Fruits', 'Veggies',
       'HvyAlcoholConsump', 'AnyHealthcare', 'NoDocbcCost', 'GenHlth',
       'MentHlth', 'PhysHlth', 'DiffWalk', 'Sex', 'Age', 'Education', 'Income',
       'Y'],
      dtype='object')
```

1.0.2 create Age_5Groups column

```
[ ]: def create_age_groups(age):
         if age in [1, 2, 3]:
             return 'Age(15~34)'
         elif age in [4, 5, 6]:
             return 'Age(35~49)'
```

```
    elif age in [7, 8, 9]:
        return 'Age(50~64)'
    elif age in [10, 11, 12]:
        return 'Age(65~79)'
    elif age in [13]:
        return 'Age(>=80)'
    else:
        print('Unknown value')


df['Age_5Groups'] = df['Age'].apply(create_age_groups)
```

[ ]:

```
df['Age_5Groups'].value_counts()
```

[ ]:

```
Age_5Groups
Age(50~64)    90390
Age(65~79)    71707
Age(35~49)    49799
Age(15~34)    24421
Age(>=80)     17363
Name: count, dtype: int64
```

[ ]:

```
# def one_hist(column_name):
#     group_counts = df[column_name].value_counts()

#     # index gives the name of the class
#     labels, counts = zip(*sorted(zip(group_counts.index, group_counts)))

#     plt.hist(df[column_name], bins=np.arange(len(labels)+1)-0.5,␣
 ↪color='skyblue', edgecolor='black')

#     plt.xticks(range(len(labels)), labels, rotation=45)

#     plt.xlabel(column_name)
#     plt.ylabel('Frequency')

#     plt.show()
```

2 Test

[ ]:

```
```

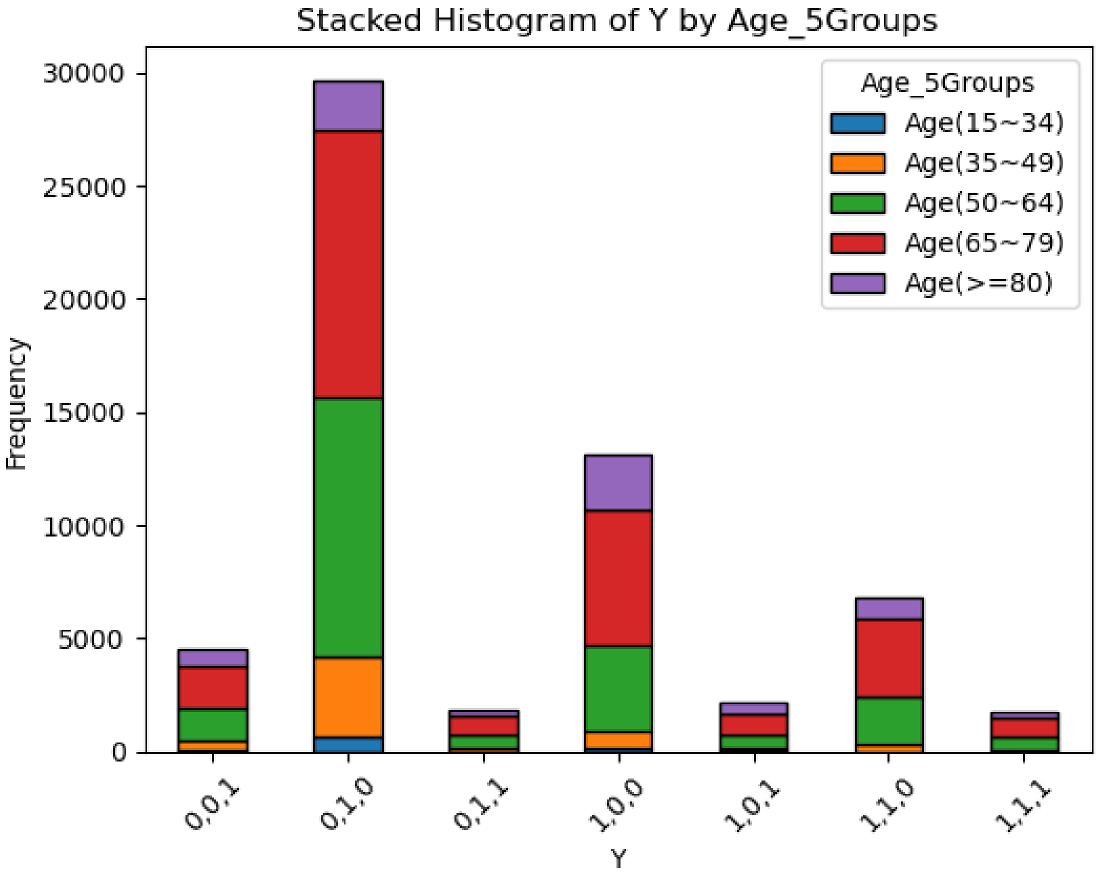
3 Test end

4 Stacked Histograms

```
[ ]: def stacked_hist(df, x_var, stack_var, drop_Y000=False):
         '''
         Draws a stacked histogram for the specified variables in the dataframe.
         @df:             dataframe
         @x_var:          name of the feature on the x-axis
         @stack_var:      name of feature used to stack the histogram
         @drop_Y000:      drop the datapoints with Y = '0,0,0'
         '''
         df_copy = df.copy()
         if drop_Y000:
             df_copy = df_copy[df_copy['Y'] != '0,0,0']

         stacked_data = df_copy.groupby([x_var, stack_var]).size().unstack()

         num_stacks = stacked_data.shape[1]
         colors = plt.cm.tab10.colors[:num_stacks]
         stacked_data.plot(kind='bar', stacked=True, color=colors, edgecolor='black')

         plt.xlabel(x_var)
         plt.ylabel('Frequency')
         plt.title('Stacked Histogram of ' + x_var + ' by ' + stack_var)
         plt.xticks(rotation=45)
         plt.show()
```

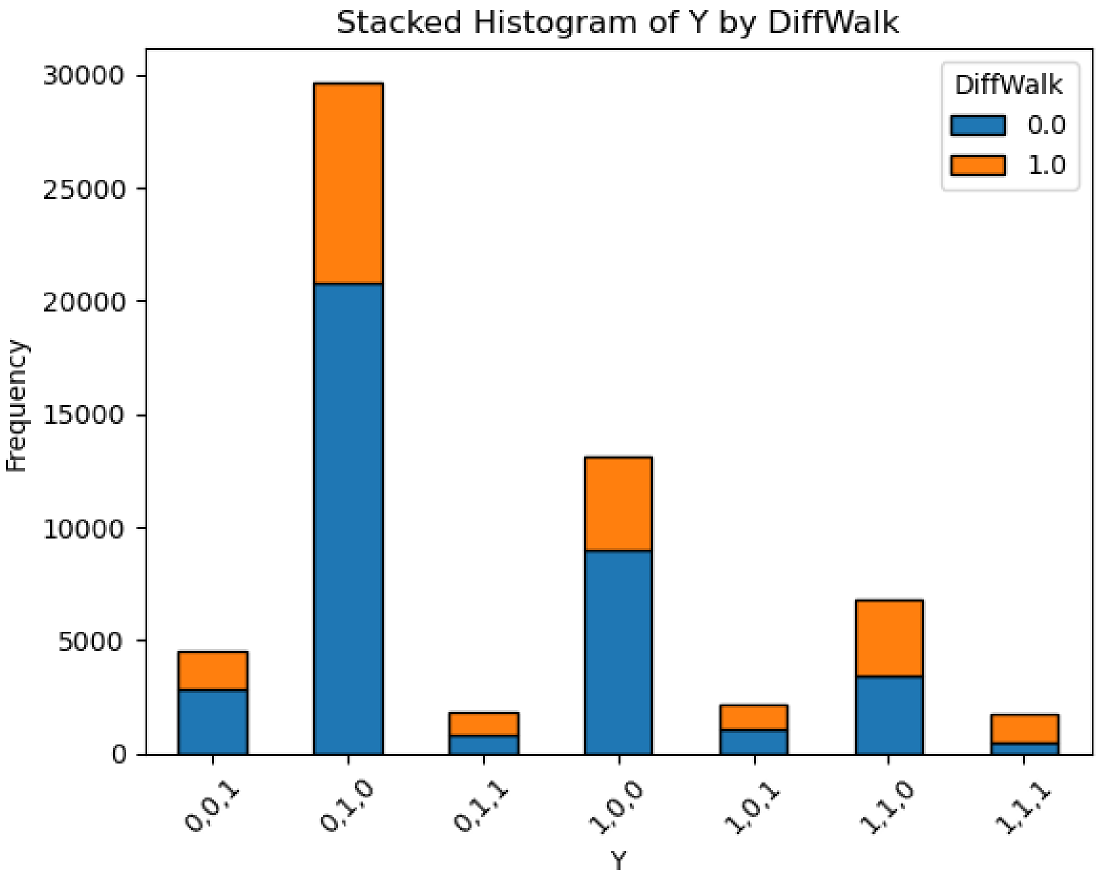
Example where we drop the datapoint with Y = 000

```
[ ]: stacked_hist(df, 'Y', 'Age_5Groups', True)
```



Example for the function where we want to get the histogram with respect to `Y` and use `DiffWalk` to stack

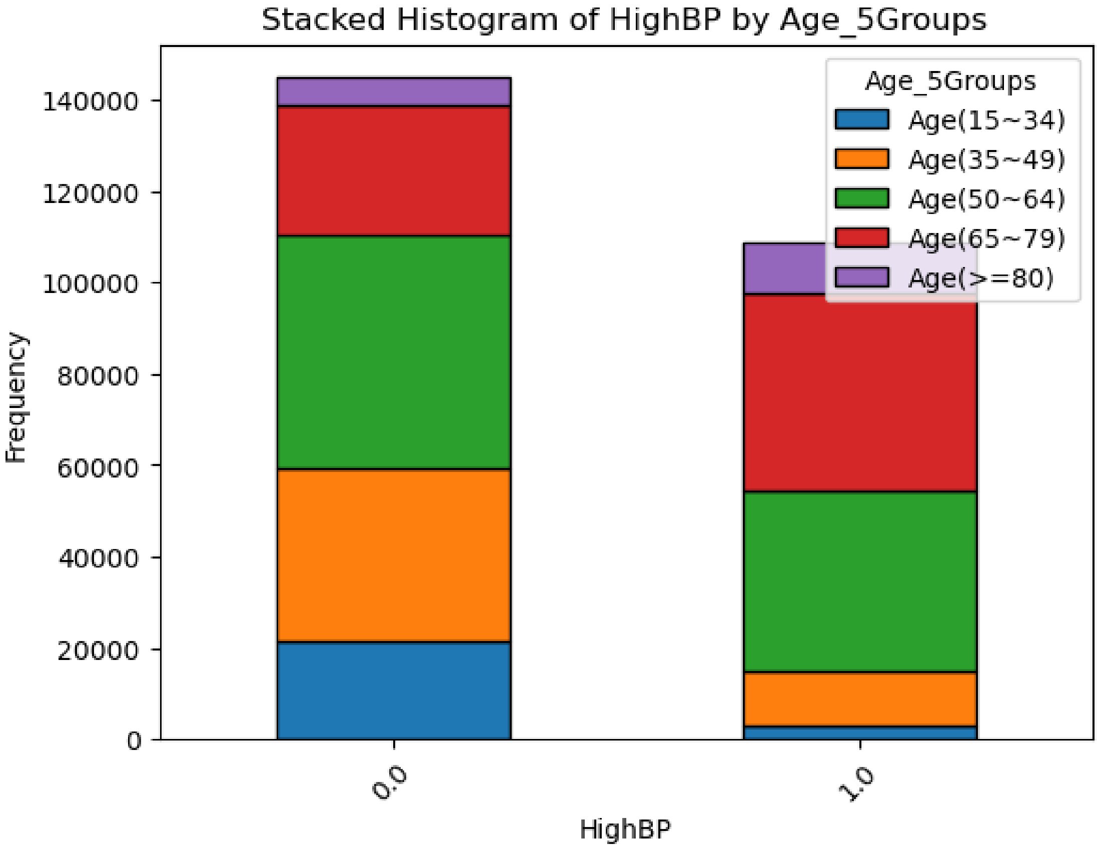
```
[ ]: stacked_hist(df, 'Y', 'DiffWalk', drop_Y000=True)
```

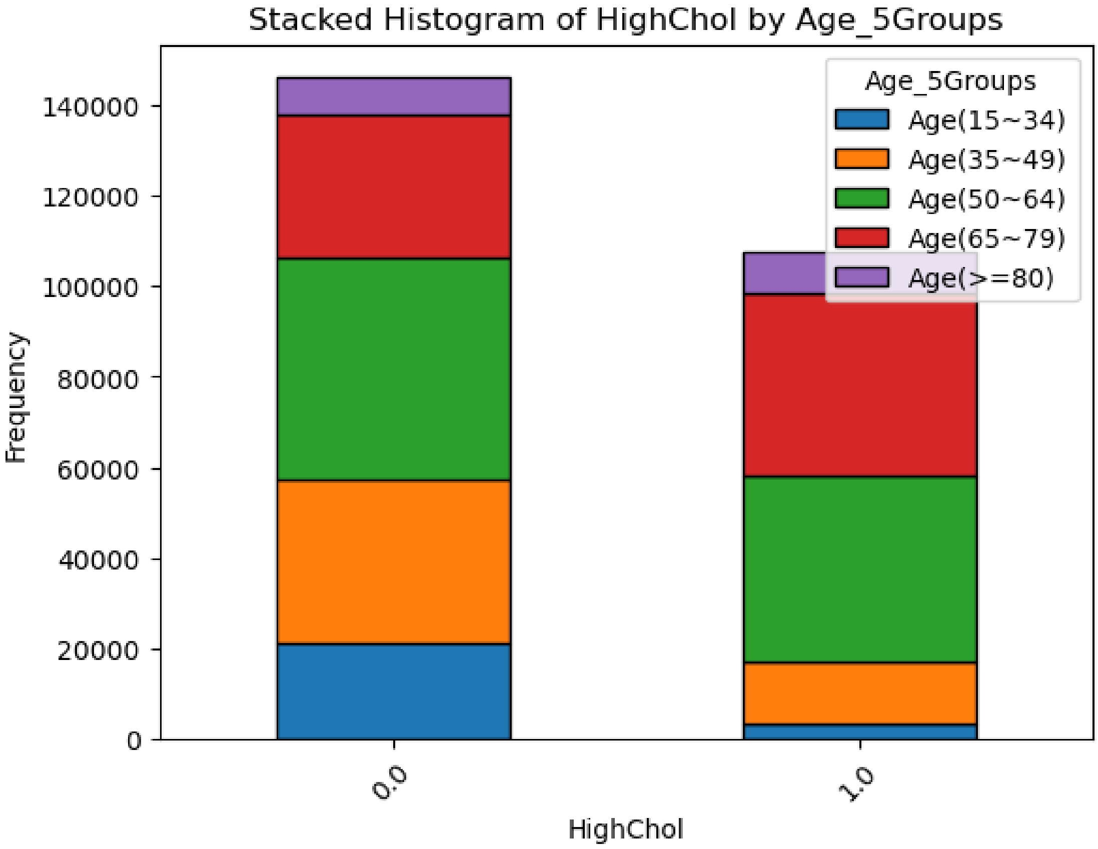


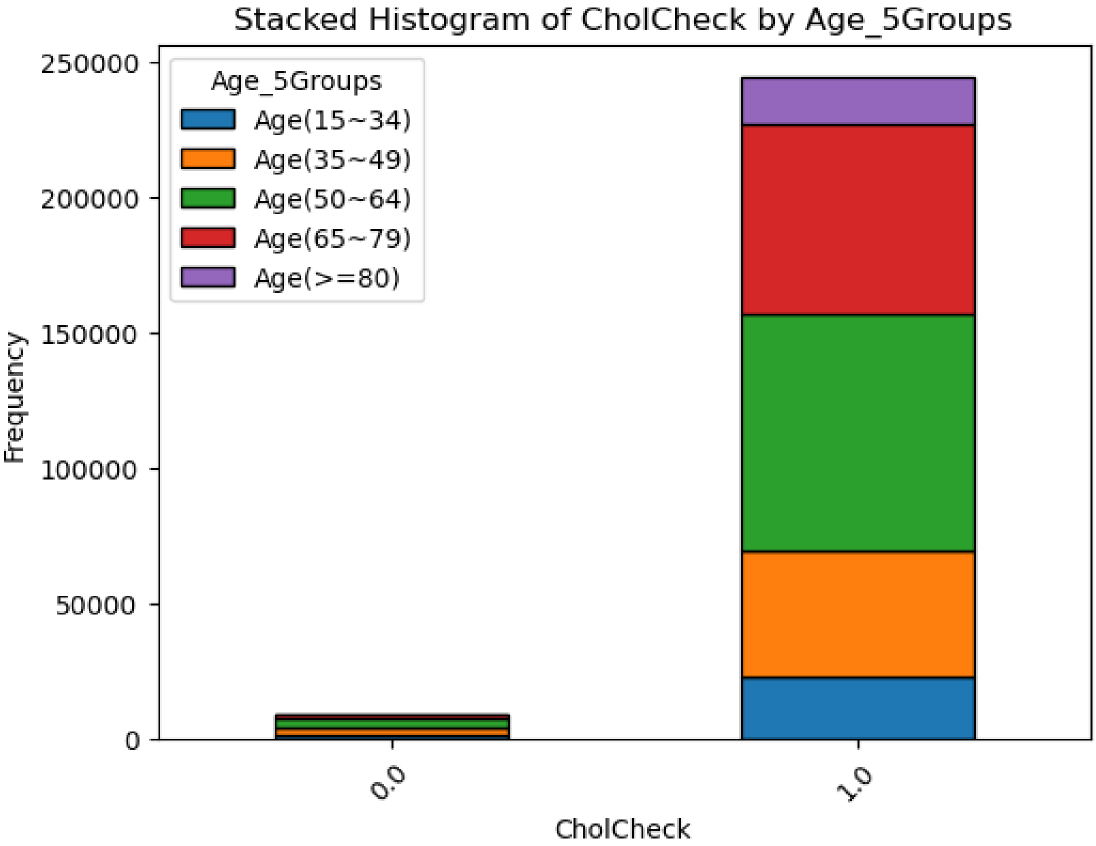
4.1 Histograms for each variable stacked on Age groups

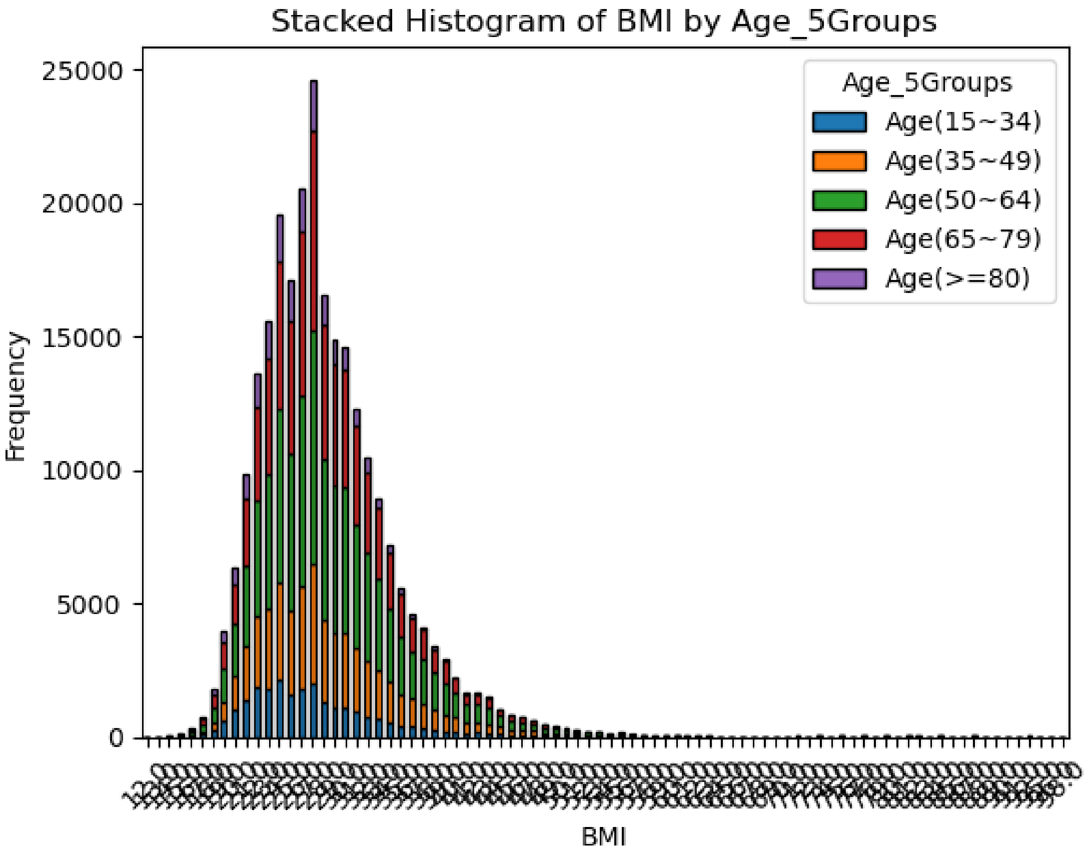
[ ]:

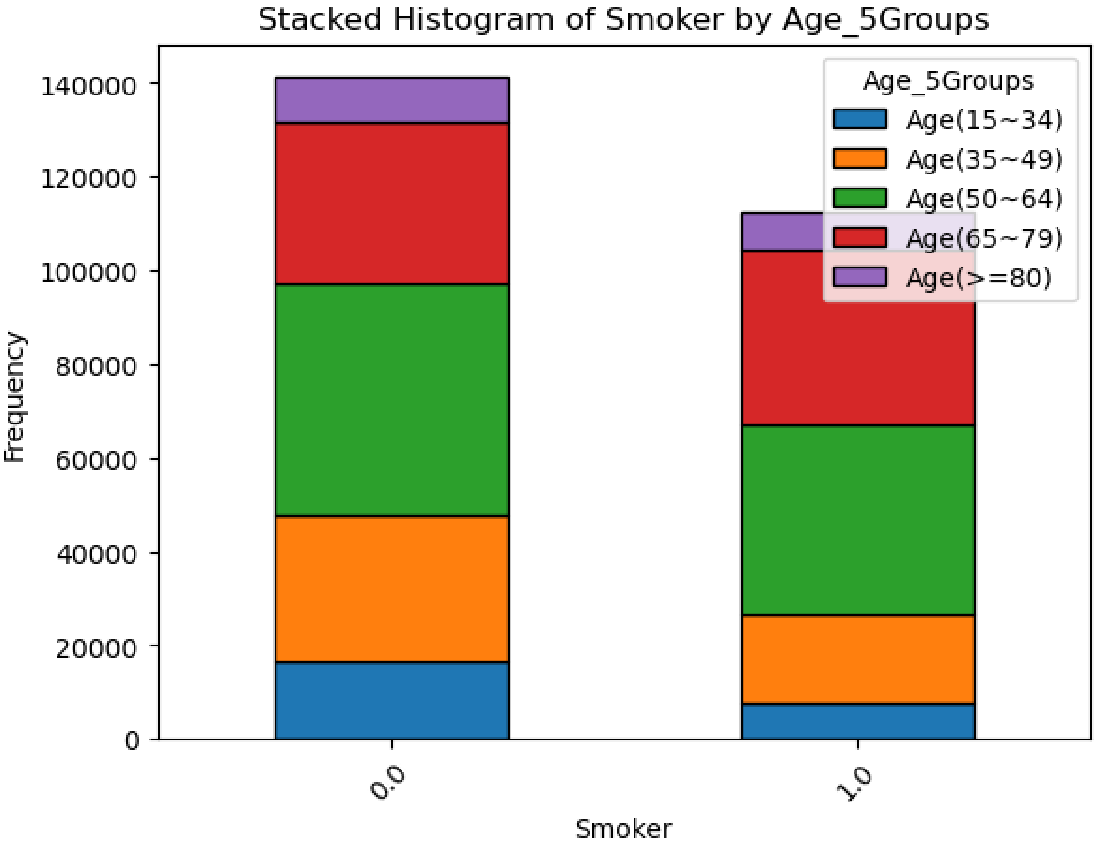
```
columns_to_drop = ['HeartDiseaseorAttack', 'Diabetes', 'Stroke', 'Age',␣
 ↪'Age_5Groups']
feature_names = df.drop(columns=columns_to_drop).columns
for name in feature_names:
    stacked_hist(df, name, 'Age_5Groups')
```

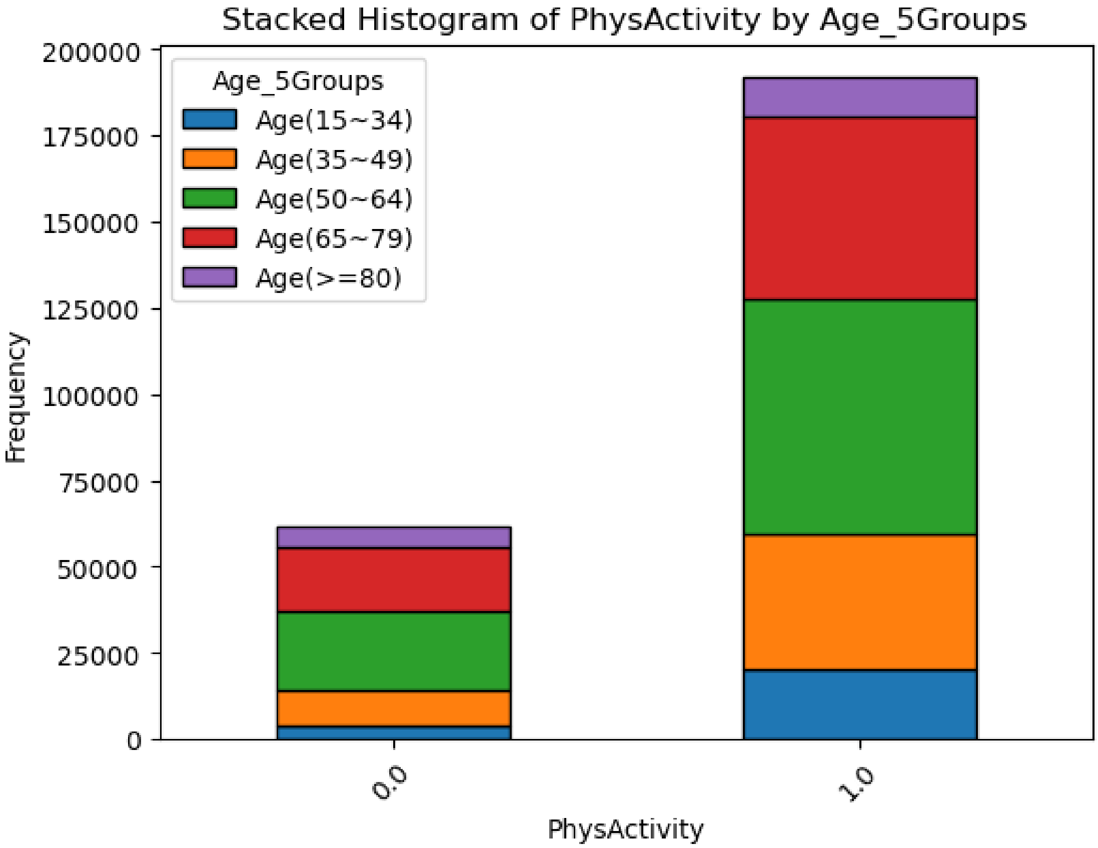

Stacked Histogram of HighBP by Age_5Groups
Age_5Groups
Age(15~34)
Age(35~49)
Age(50~64)
Age(65~79)
Age(>=80)
Frequency
0
20000
40000
60000
80000
100000
120000
140000
0.0
1.0
HighBP

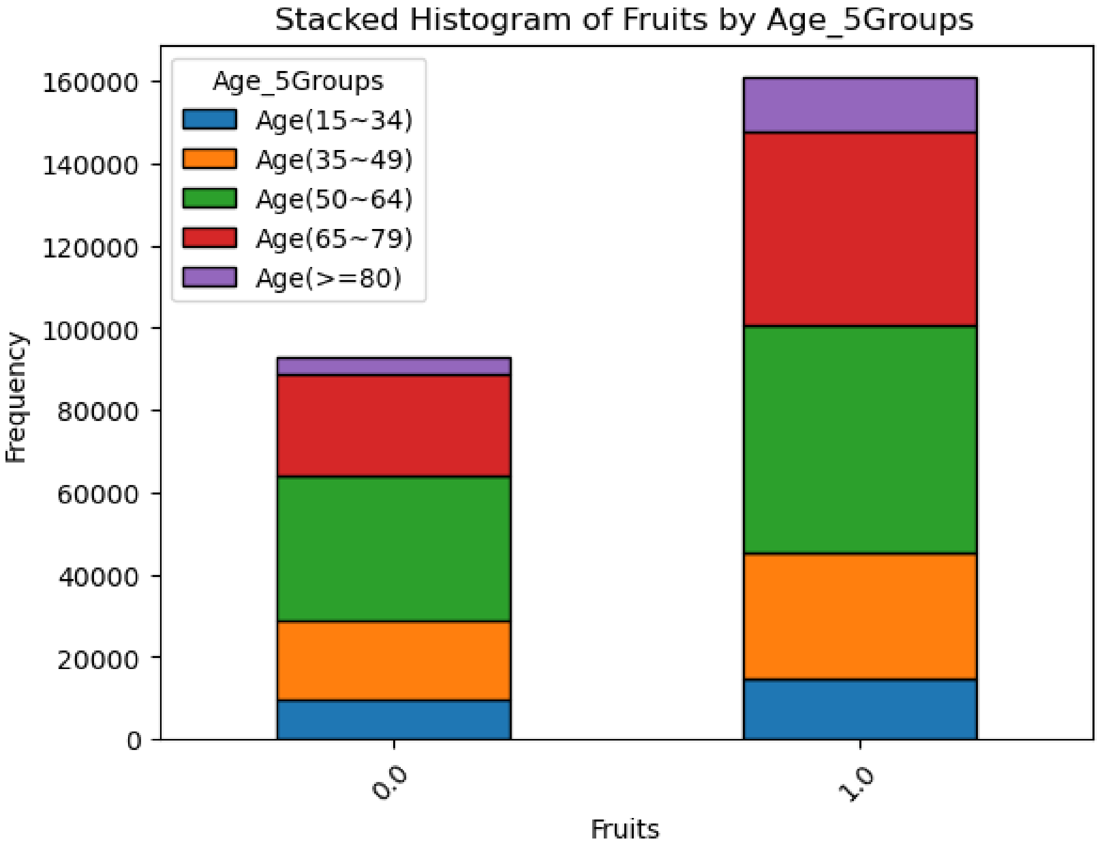

Stacked Histogram of HighChol by Age_5Groups
Frequency
0
20000
40000
60000
80000
100000
120000
140000
0.0
1.0
HighChol
Age_5Groups
Age(15~34)
Age(35~49)
Age(50~64)
Age(65~79)
Age(>=80)

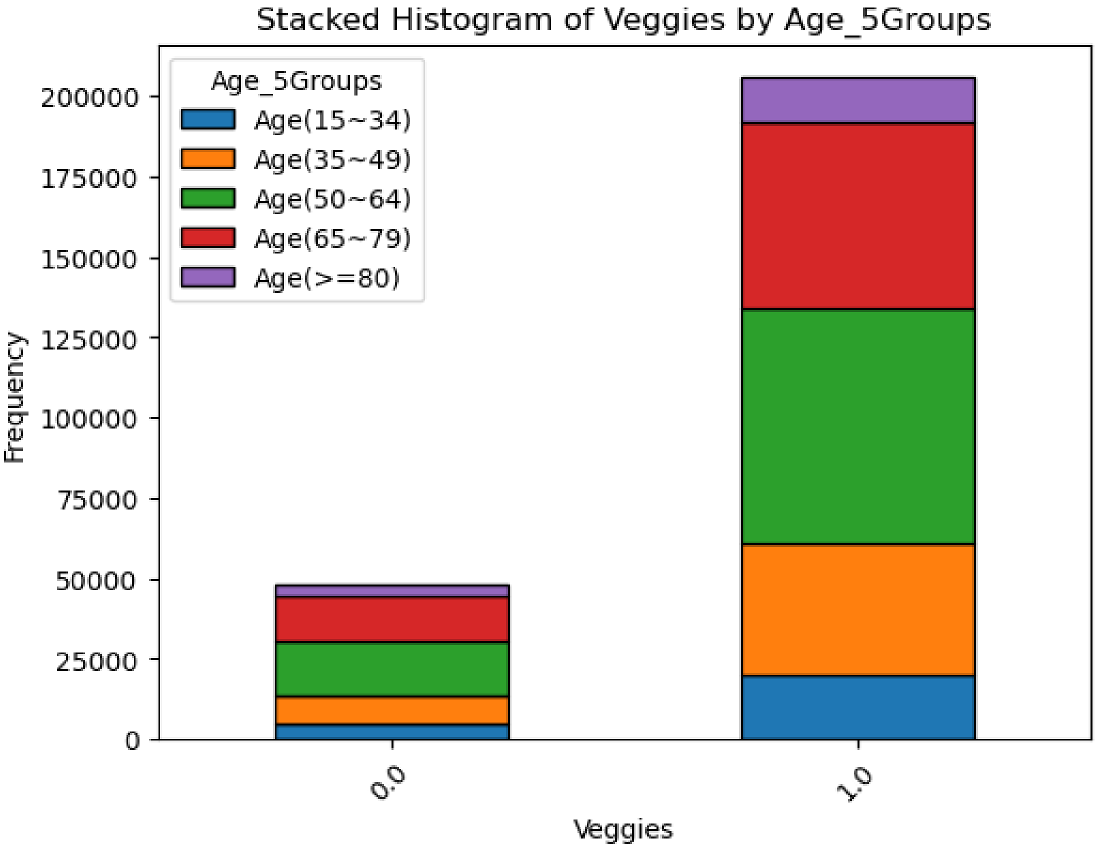

Stacked Histogram of CholCheck by Age_5Groups
Age_5Groups
Age(15~34)
Age(35~49)
Age(50~64)
Age(65~79)
Age(>=80)
Frequency
0
50000
100000
150000
200000
250000
0.0
1.0
CholCheck

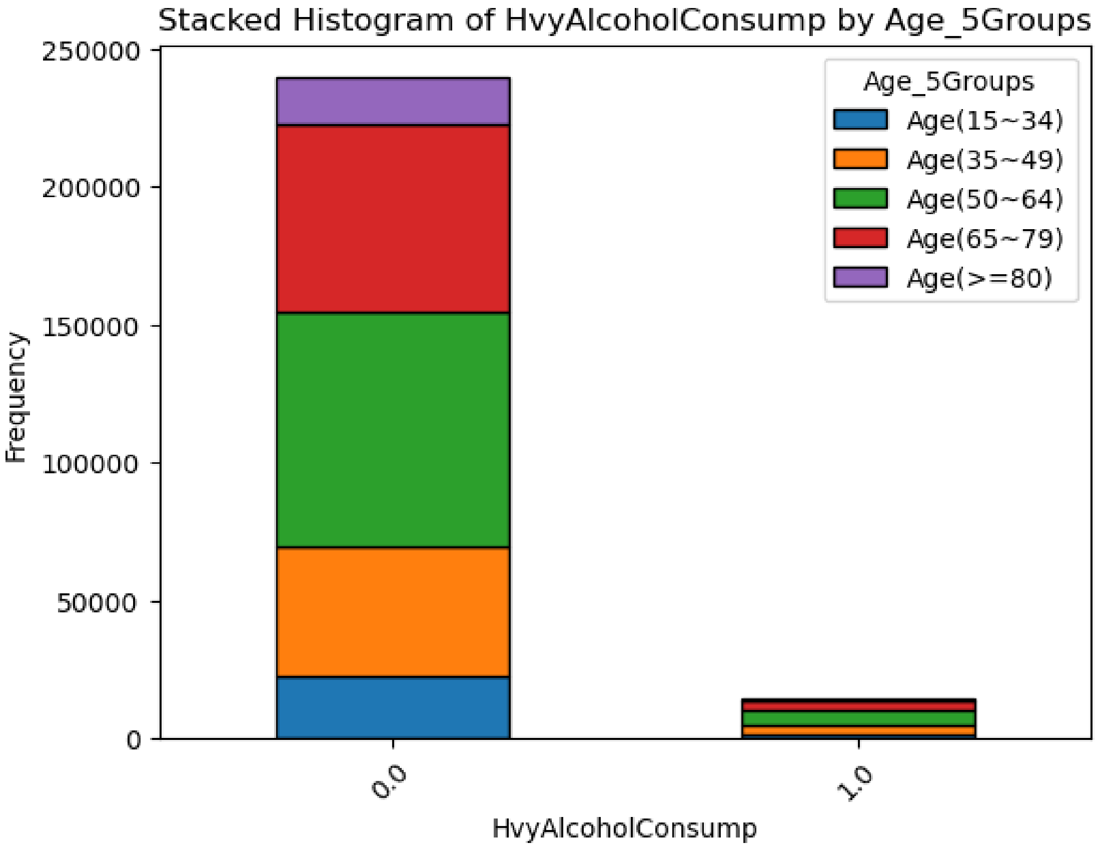

Stacked Histogram of BMI by Age_5Groups
Frequency
25000
20000
15000
10000
5000
0
BMI
Age_5Groups
Age(15~34)
Age(35~49)
Age(50~64)
Age(65~79)
Age(>=80)

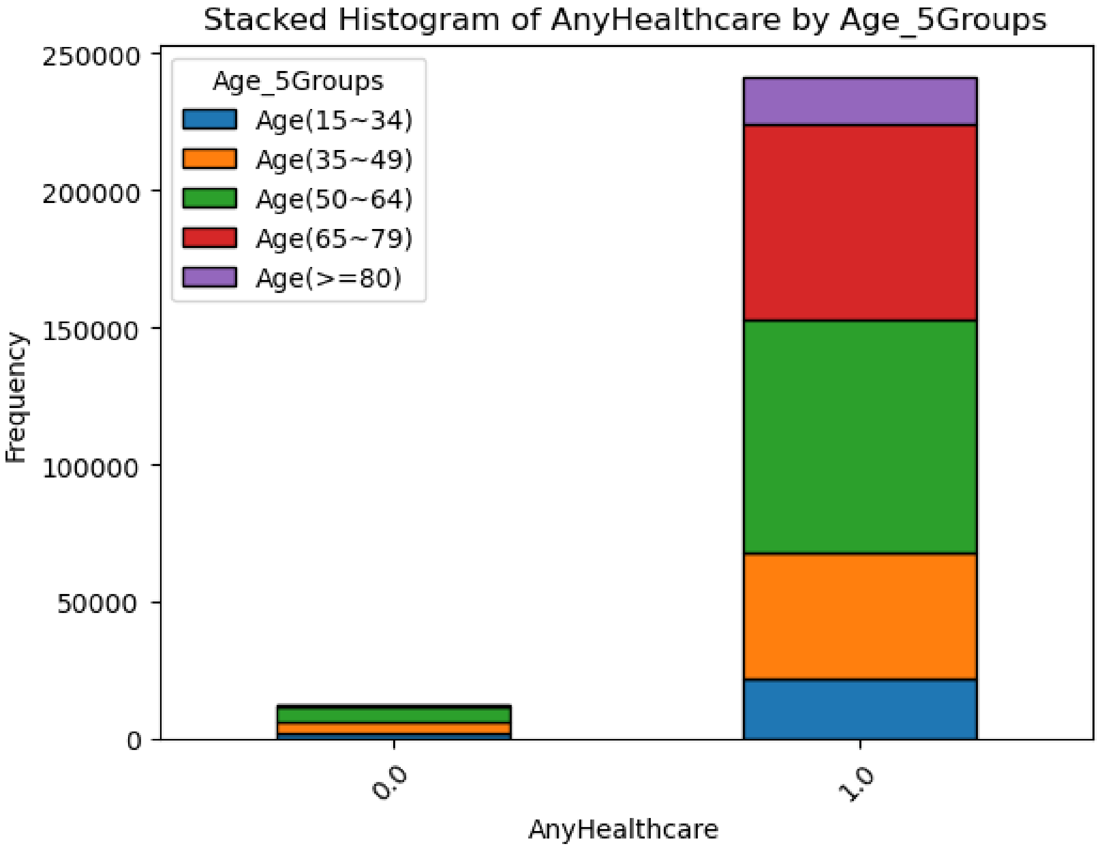

Stacked Histogram of Smoker by Age_5Groups
Frequency
0
20000
40000
60000
80000
100000
120000
140000
Smoker
0.0
1.0
Age_5Groups
Age(15~34)
Age(35~49)
Age(50~64)
Age(65~79)
Age(>=80)

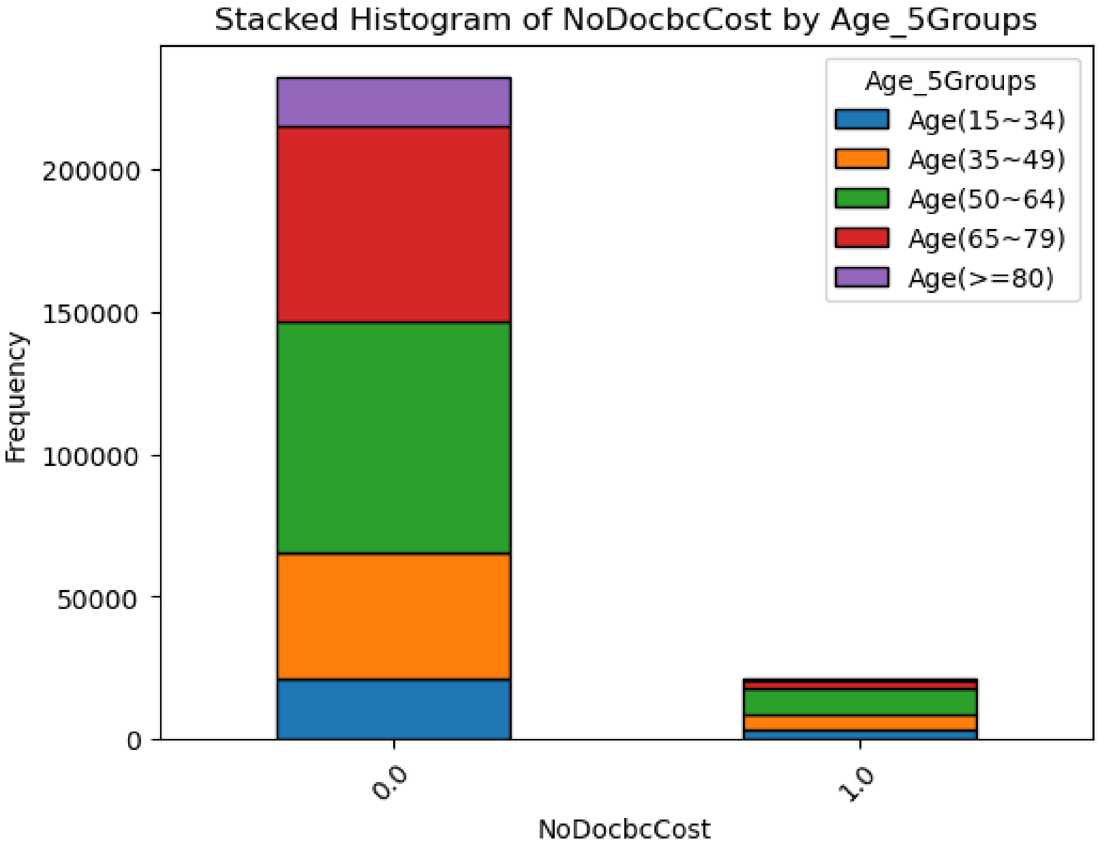

Stacked Histogram of PhysActivity by Age_5Groups
Age_5Groups
Age(15~34)
Age(35~49)
Age(50~64)
Age(65~79)
Age(>=80)
Frequency
0
25000
50000
75000
100000
125000
150000
175000
200000
0.0
1.0
PhysActivity

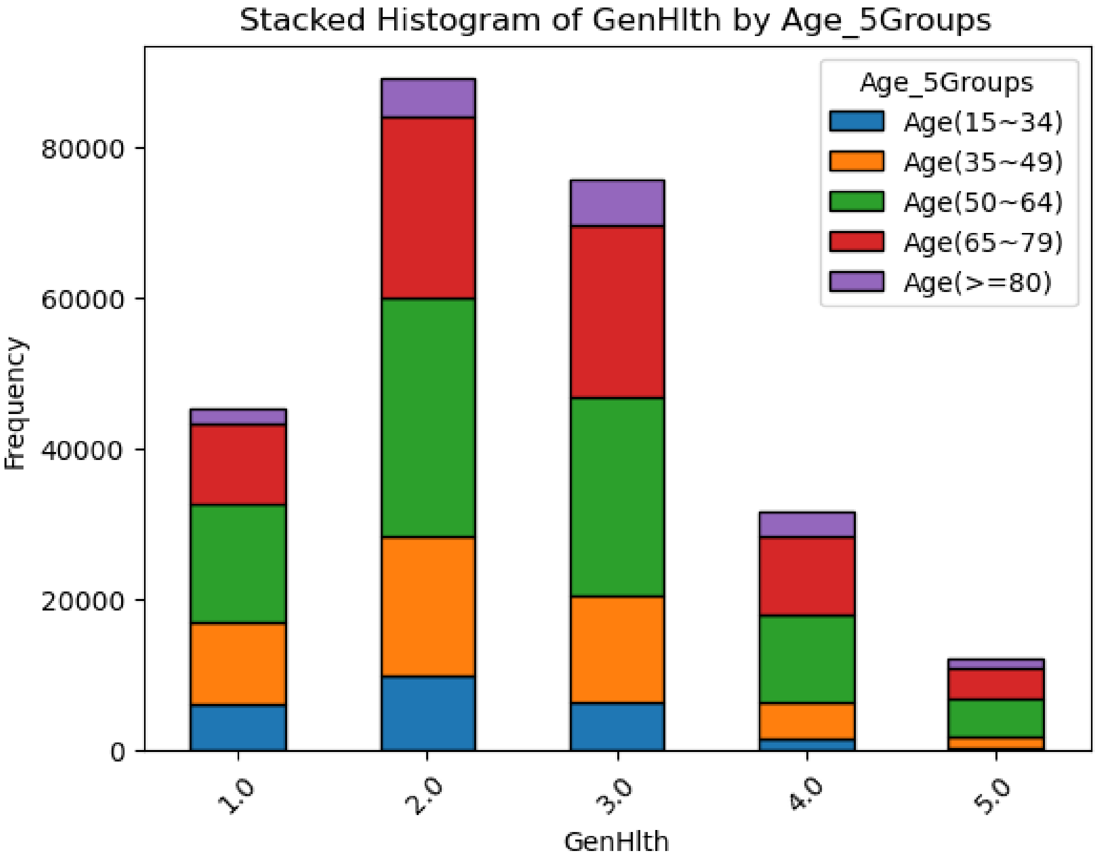

Stacked Histogram of Fruits by Age_5Groups
Age_5Groups
Age(15~34)
Age(35~49)
Age(50~64)
Age(65~79)
Age(>=80)
Frequency
160000
140000
120000
100000
80000
60000
40000
20000
0
0.0
1.0
Fruits

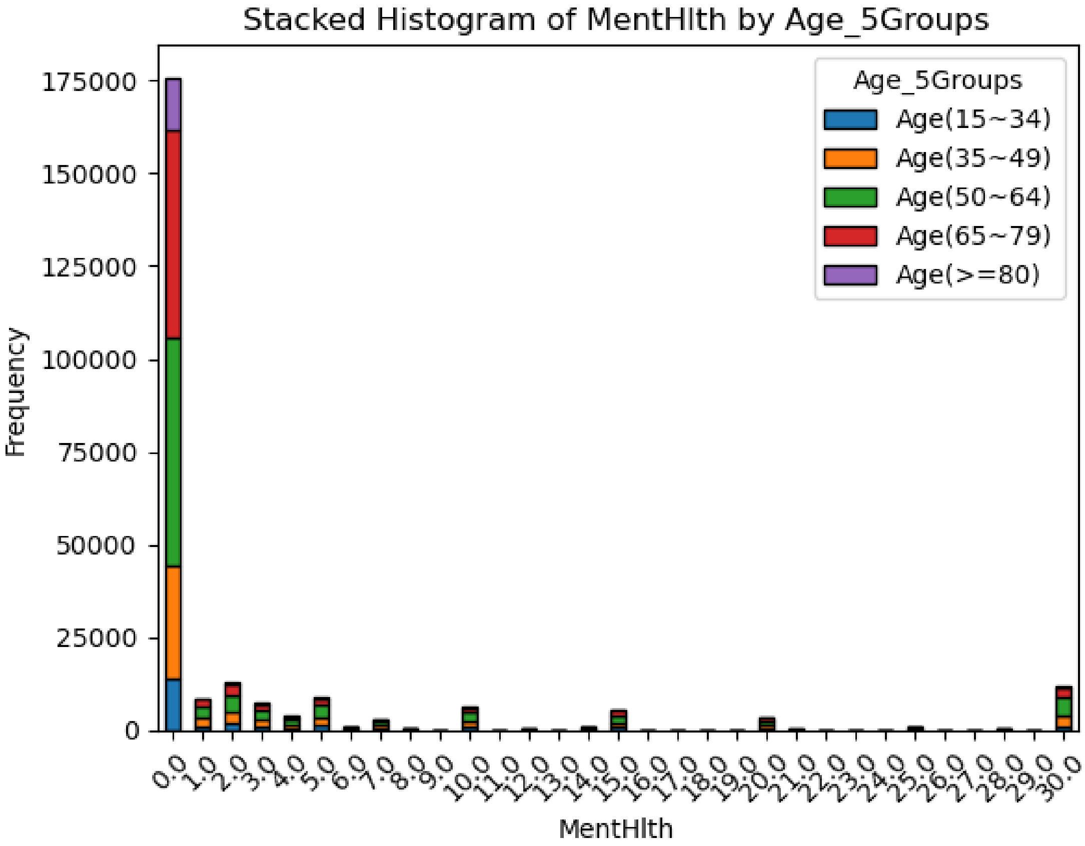

Stacked Histogram of Veggies by Age_5Groups
Age_5Groups
Age(15~34)
Age(35~49)
Age(50~64)
Age(65~79)
Age(>=80)
Frequency
0
25000
50000
75000
100000
125000
150000
175000
200000
0.0
1.0
Veggies

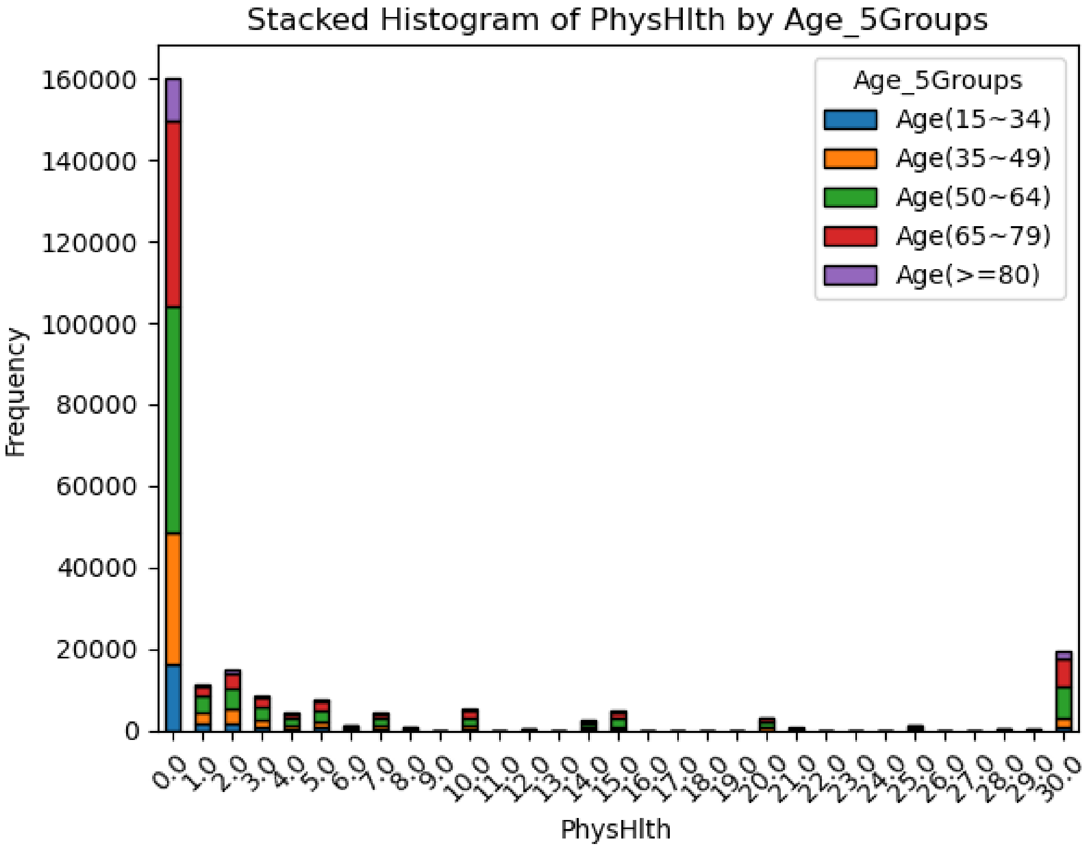

Stacked Histogram of HvyAlcoholConsump by Age_5Groups
Frequency
250000
200000
150000
100000
50000
0
0.0
1.0
HvyAlcoholConsump
Age_5Groups
Age(15~34)
Age(35~49)
Age(50~64)
Age(65~79)
Age(>=80)

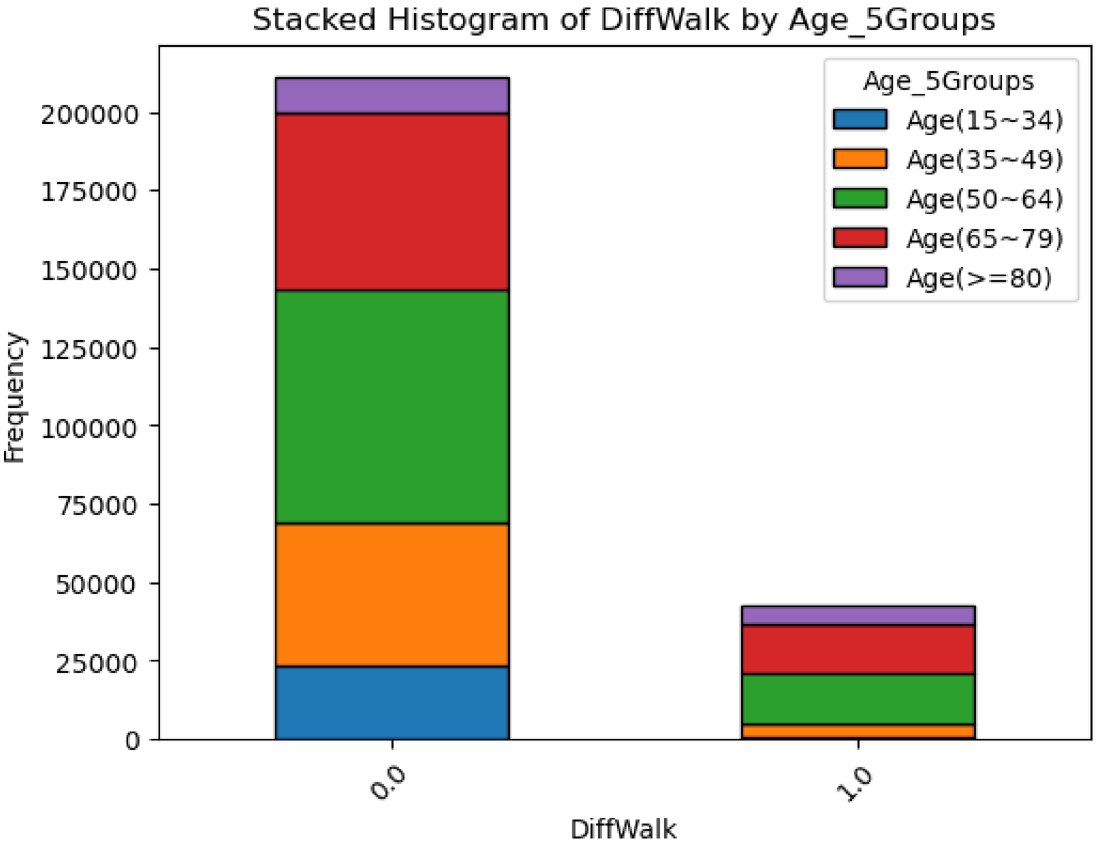

Stacked Histogram of AnyHealthcare by Age_5Groups
Frequency
250000
200000
150000
100000
50000
0
Age_5Groups
Age(15~34)
Age(35~49)
Age(50~64)
Age(65~79)
Age(>=80)
0.0
1.0
AnyHealthcare


Stacked Histogram of NoDocbcCost by Age_5Groups
Age_5Groups
Age(15~34)
Age(35~49)
Age(50~64)
Age(65~79)
Age(>=80)
Frequency
0
50000
100000
150000
200000
0.0
1.0
NoDocbcCost

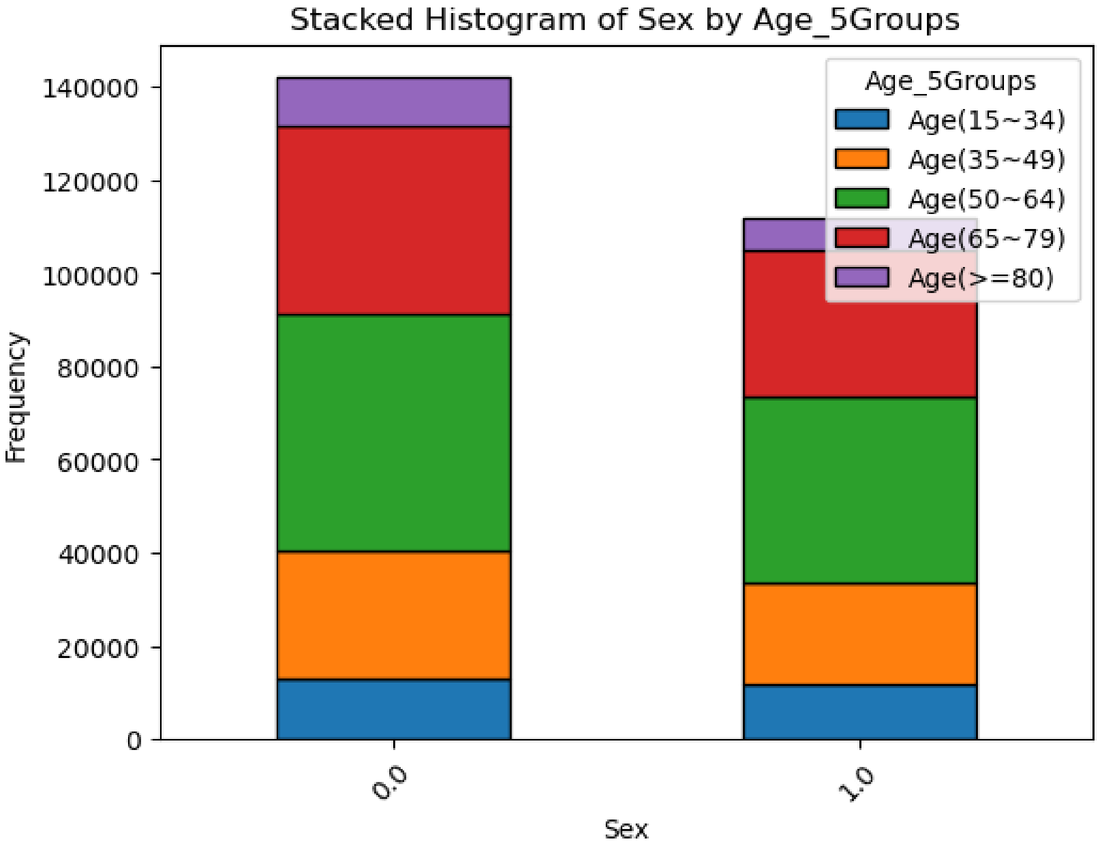

Stacked Histogram of GenHlth by Age_5Groups
Age_5Groups
Age(15~34)
Age(35~49)
Age(50~64)
Age(65~79)
Age(>=80)
Frequency
0
20000
40000
60000
80000
1.0
2.0
3.0
4.0
5.0
GenHlth

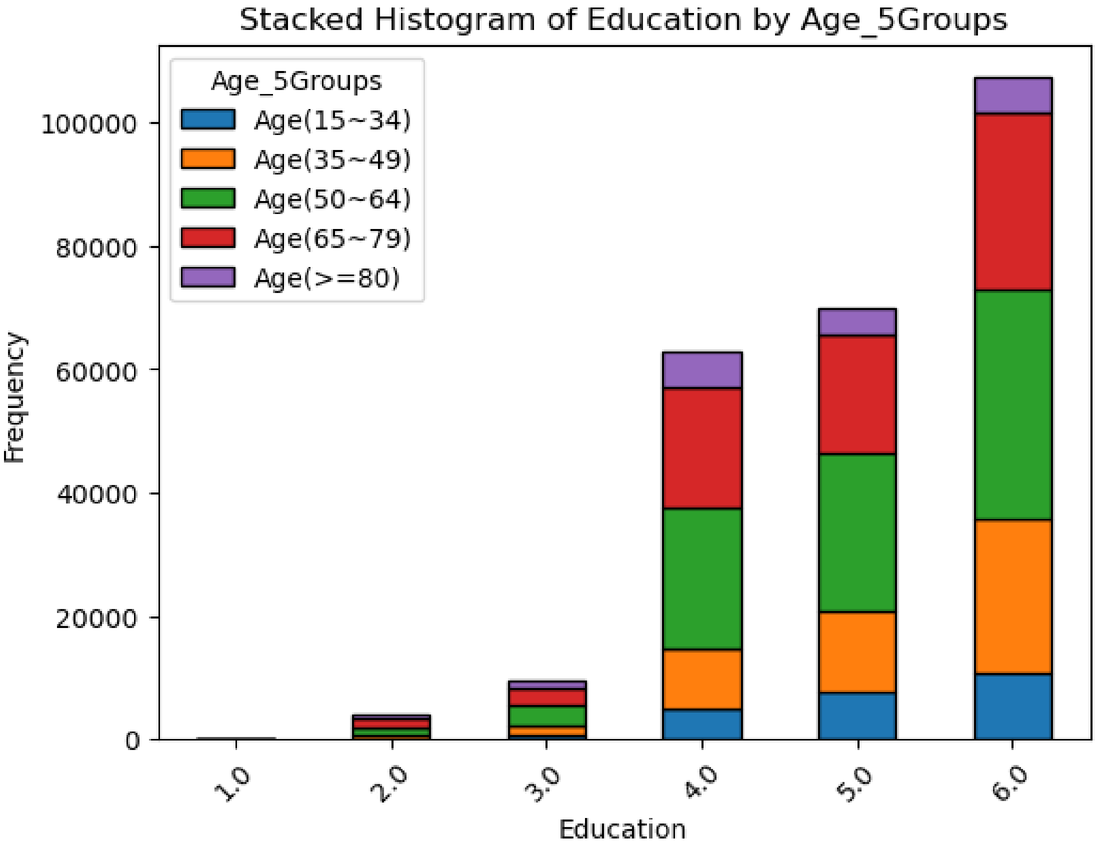

Stacked Histogram of MentHlth by Age_5Groups
Frequency
175000
150000
125000
100000
75000
50000
25000
0
0.0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0
9.0
10.0
11.0
12.0
13.0
14.0
15.0
16.0
17.0
18.0
19.0
20.0
21.0
22.0
23.0
24.0
25.0
26.0
27.0
28.0
29.0
30.0
MentHlth
Age_5Groups
Age(15~34)
Age(35~49)
Age(50~64)
Age(65~79)
Age(>=80)

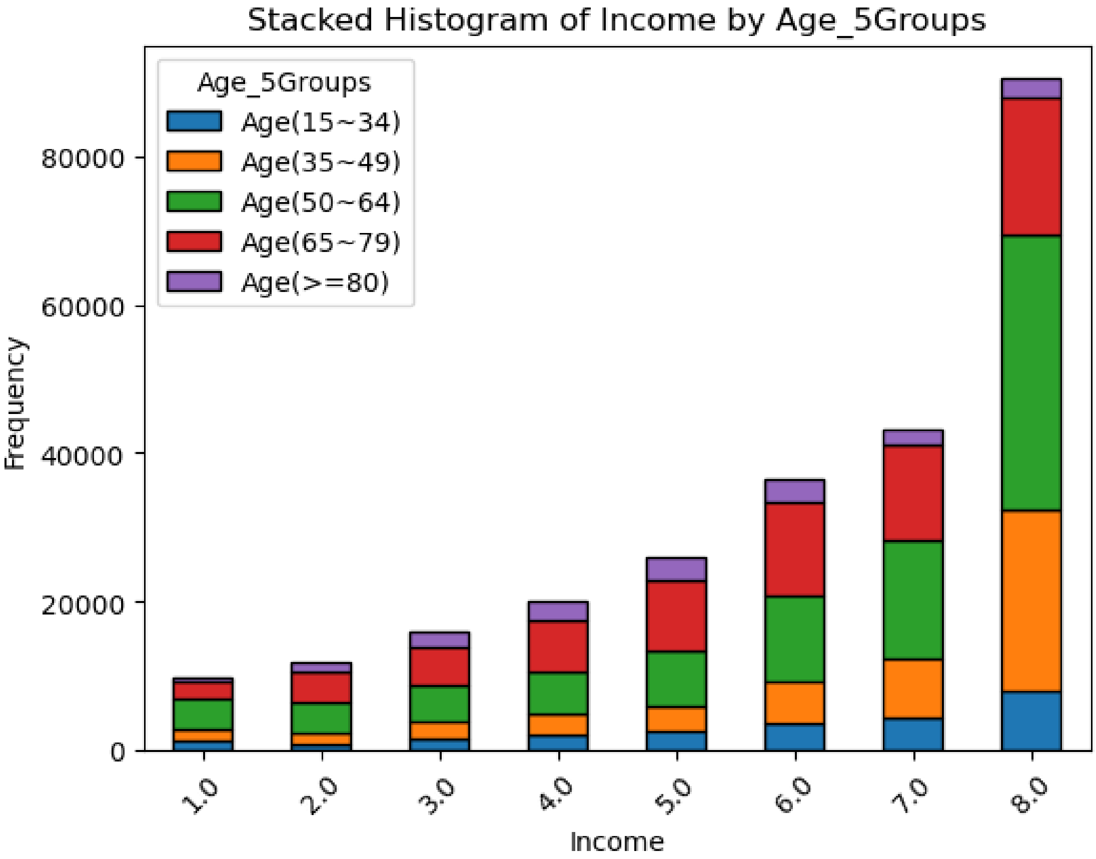

Stacked Histogram of PhysHlth by Age_5Groups
Frequency
0
20000
40000
60000
80000
100000
120000
140000
160000
PhysHlth
0.0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0
9.0
10.0
11.0
12.0
13.0
14.0
15.0
16.0
17.0
18.0
19.0
20.0
21.0
22.0
23.0
24.0
25.0
26.0
27.0
28.0
29.0
30.0
Age_5Groups
Age(15~34)
Age(35~49)
Age(50~64)
Age(65~79)
Age(>=80)

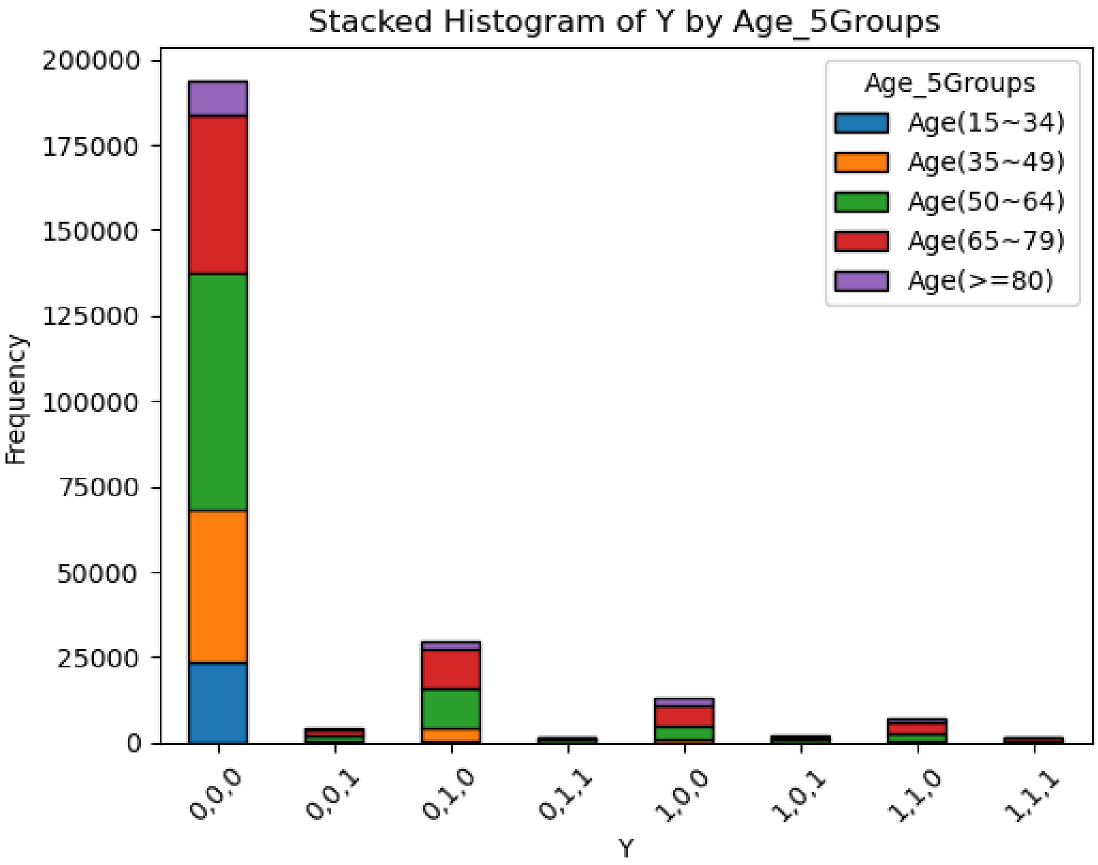


Stacked Histogram of DiffWalk by Age_5Groups
Frequency
0
25000
50000
75000
100000
125000
150000
175000
200000
DiffWalk
0.0
1.0
Age_5Groups
🟦 Age(15~34)
🟧 Age(35~49)
🟩 Age(50~64)
🟥 Age(65~79)
🟪 Age(>=80)


Stacked Histogram of Sex by Age_5Groups
Age_5Groups
Age(15~34)
Age(35~49)
Age(50~64)
Age(65~79)
Age(>=80)
Frequency
0
20000
40000
60000
80000
100000
120000
140000
0.0
1.0
Sex


Stacked Histogram of Education by Age_5Groups
Age_5Groups
Age(15~34)
Age(35~49)
Age(50~64)
Age(65~79)
Age(>=80)
Frequency
100000
80000
60000
40000
20000
0
1.0
2.0
3.0
4.0
5.0
6.0
Education


Stacked Histogram of Income by Age_5Groups
Age_5Groups
Age(15~34)
Age(35~49)
Age(50~64)
Age(65~79)
Age(>=80)
Frequency
0
20000
40000
60000
80000
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0
Income



5 Mutual Information

5.0.1 Function that calculates the mutual entropy given a column

```
[ ]: ##Entropy
     def entropy(Y):
         if isinstance(Y, pd.Series):
             Y = Y.values

         # Handle object dtype appropriately
         if Y.dtype == object:
             # Convert array elements to string if they are not, ensures unique␣
      ↪works correctly
             Y = np.array([str(y) for y in Y])

         unique, count = np.unique(Y, return_counts=True, axis=0)
         prob = count/len(Y)
         en = np.sum((-1)*prob*np.log(prob))
         return en
```

```
#Joint Entropy
def jEntropy(Y,X):
    YX = np.c_[Y,X]
    return entropy(YX)

#Conditional Entropy
def cEntropy(Y, X):
    return jEntropy(Y, X) - entropy(X)


#Information Gain
def get_mutual_information(df, Y_name, X_name, drop_Y000=False,␣
 ↪merge_Ynot000=False):
    '''
    Calculates the mutual entropy given the name of the column
    @df:              dataframe
    @columnName:      name of the column that we want to calculate the mutual␣
 ↪entropy for
    @drop_Y000:       drop the datapoints with Y = '0,0,0'
    @merge_Ynot000: convert all the values in Y that is not '0,0,0' to '1,1,1'
    '''
    df_copy = df.copy()

    if drop_Y000:
        df_copy = df_copy[df_copy['Y'] != '0,0,0']

    if merge_Ynot000:
        df_copy.loc[df_copy['Y'] != '0,0,0', 'Y'] = '1,1,1'

    return entropy(df_copy[Y_name]) - cEntropy(df_copy[Y_name], df_copy[X_name])
```

[ ]:

```
get_mutual_information(df, 'Y', 'BMI')
```

[ ]:

```
0.03152370498739021
```

[ ]:

```
# def get_mutual_information_original(df, columnName, drop_Y000=False,␣
 ↪merge_Ynot000=False):
#     '''
#     Calculates the mutual entropy given the name of the column
#     @df:              dataframe
#     @columnName:      name of the column that we want to calculate the mutual␣
 ↪entropy for
#     @drop_Y000:       drop the datapoints with Y = '0,0,0'
#     @merge_Ynot000: convert all the values in Y that is not '0,0,0' to '1,1,1'
#     '''
#     df_copy = df.copy()

#     if drop_Y000:
#         df_copy = df_copy[df_copy['Y'] != '0,0,0']

#     if merge_Ynot000:
#         df_copy.loc[df_copy['Y'] != '0,0,0', 'Y'] = '1,1,1'

#     contingency_table = pd.crosstab(df_copy[columnName], df_copy['Y'])
#     total = contingency_table.values.sum()
#     probs = contingency_table.values / total

#     conditional_entropy = sum(entropy(prob) * prob.sum() for prob in probs)

#     prob_Y = contingency_table.sum(axis=0) / total
#     entropy_Y = entropy(prob_Y)

#     mutual_information = entropy_Y - conditional_entropy
#     return mutual_information
```

[ ]:
```
columns_to_drop = ['HeartDiseaseorAttack', 'Diabetes', 'Stroke', 'Y', 'Age',␣
 ↪'Age_5Groups']
```

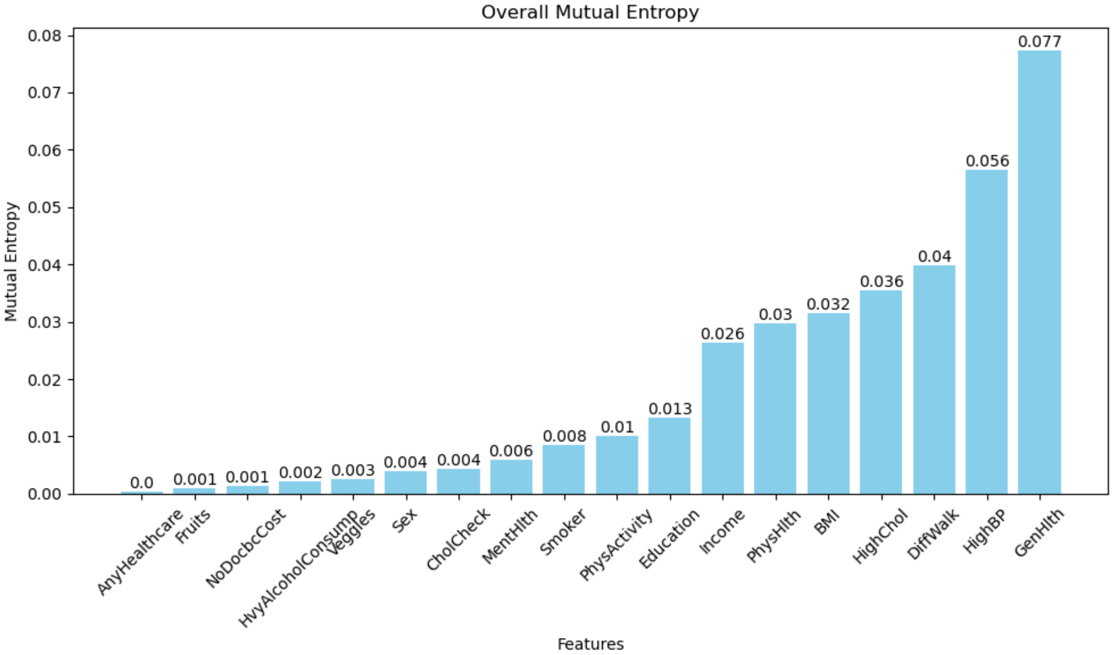
Function that draws the Mutual Entropy graph

[ ]:
```
def draw_mutual_information(df, columns_to_drop, byGroup=None,␣
 ↪targetColumn='Y'):
    '''
    Draws the Mutual Entropy graph(s)
    @df:                        dataframe
    @columns_to_drop:           a list of columns that we don't want to be included␣
 ↪in the graph
    @byGroup:                   the name of the column to group the data by before␣
 ↪calculating mutual entropy.
                                Each unique value in this column will result in a␣
 ↪separate graph. If None (default),
                                a single graph for all features is produced.
    '''
    if not byGroup:
        feature_names = df.drop(columns=columns_to_drop).columns
        mutual_information_list = []
        for columnName in feature_names:
            mutual_information = get_mutual_information(df, columnName,␣
 ↪targetColumn)
            mutual_information_list.append(mutual_information)
        features_with_entropy = list(zip(feature_names,␣
↪mutual_information_list))
        sorted_features_with_entropy = sorted(features_with_entropy, key=lambda␣
↪x: x[1])
        sorted_feature_names, sorted_entropy_list =␣
↪zip(*sorted_features_with_entropy)

        plt.figure(figsize=(10, 6))
        plt.bar(range(len(sorted_entropy_list)), sorted_entropy_list,␣
↪color='skyblue')
        plt.xlabel('Features')
        plt.ylabel('Mutual Entropy')

        plt.title('Overall Mutual Entropy')

        plt.xticks(range(len(sorted_entropy_list)), sorted_feature_names,␣
↪rotation=45)
        plt.tight_layout()

        for i, entropy_value in enumerate(sorted_entropy_list):
            plt.text(i, entropy_value, round(entropy_value, 3), ha='center',␣
↪va='bottom')

        plt.show()

    else:
        value_names = df[byGroup].unique()
        value_names = sorted(value_names)

        for value in value_names:
            df_group = df[df[byGroup] == value]

            feature_names = df_group.drop(columns=columns_to_drop).columns

            mutual_information_list = []
            for columnName in feature_names:
                mutual_information = get_mutual_information(df_group,␣
↪columnName, targetColumn)
                mutual_information_list.append(mutual_information)

            features_with_entropy = list(zip(feature_names,␣
↪mutual_information_list))
            sorted_features_with_entropy = sorted(features_with_entropy,␣
↪key=lambda x: x[1])
            sorted_feature_names, sorted_entropy_list =␣
↪zip(*sorted_features_with_entropy)
```

```
            plt.figure(figsize=(10, 6))
            plt.bar(range(len(sorted_entropy_list)), sorted_entropy_list,␣
↪color='skyblue')
            plt.xlabel('Features')
            plt.ylabel('Mutual information')
            plt.title(f'Mutual information between {targetColumn} and other␣
↪features where the df is grouped by {byGroup} and the value is {value}')
            plt.xticks(range(len(sorted_entropy_list)), sorted_feature_names,␣
↪rotation=45)
            plt.tight_layout()

            for i, entropy_value in enumerate(sorted_entropy_list):
                plt.text(i, entropy_value, round(entropy_value, 3),␣
↪ha='center', va='bottom')

            plt.show()
```

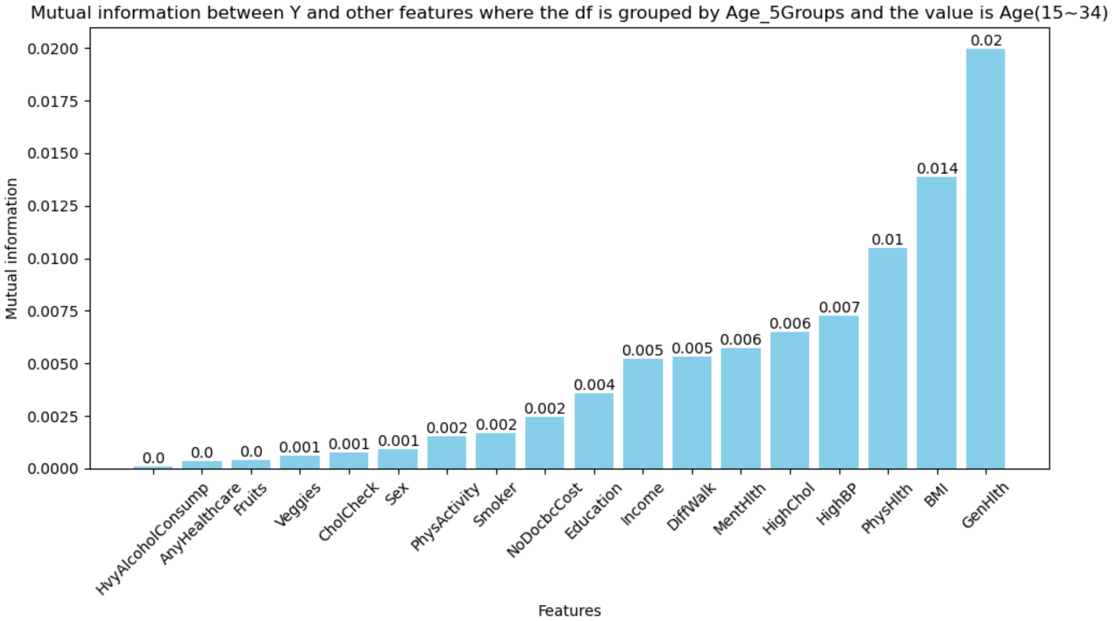
5.0.2 Mutual Entropy

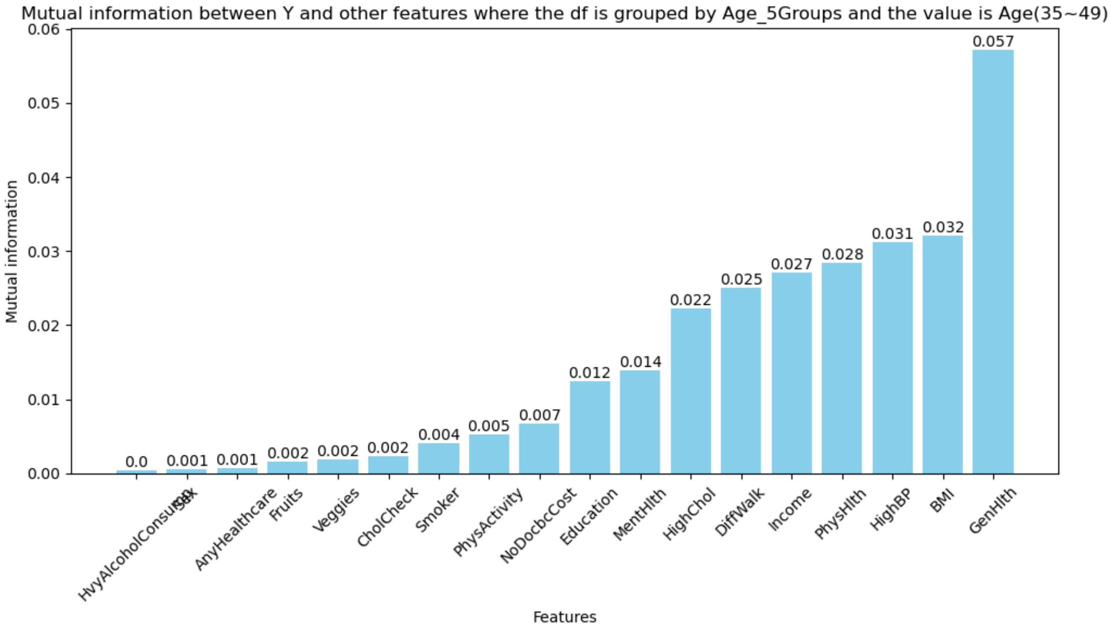
[ ]:
```
draw_mutual_information(df, columns_to_drop)
```

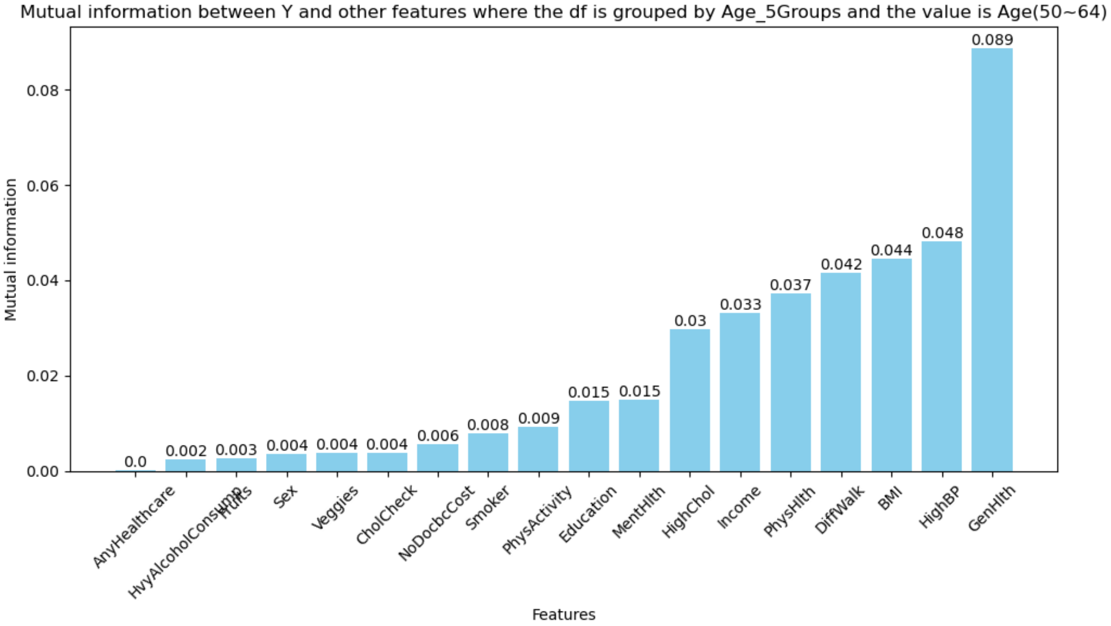


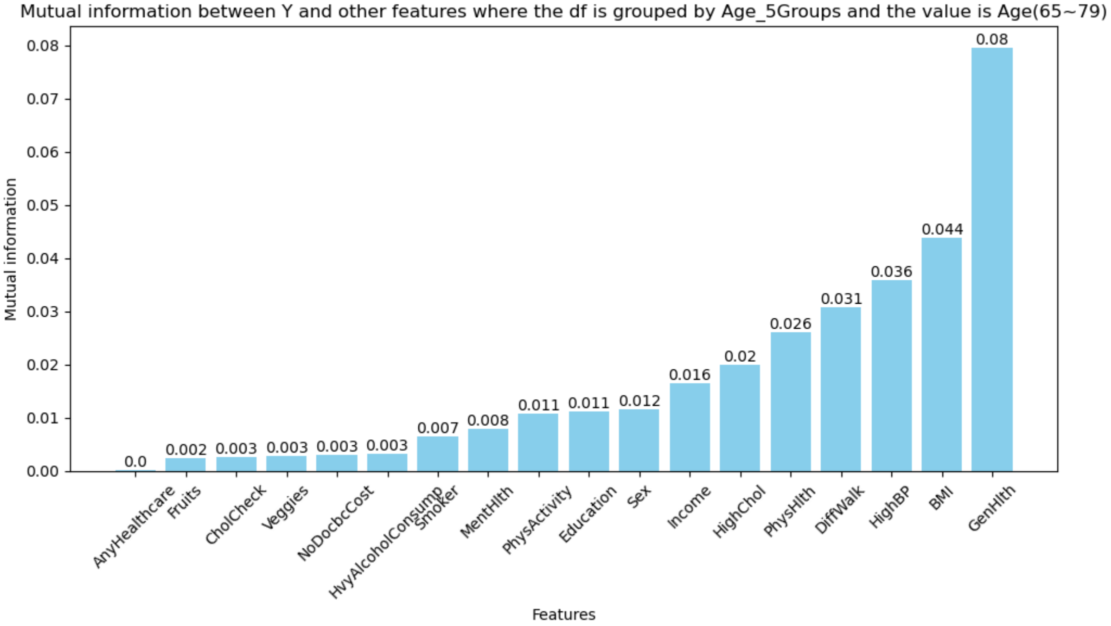
5.0.3 Mutual Entropy from each Age group

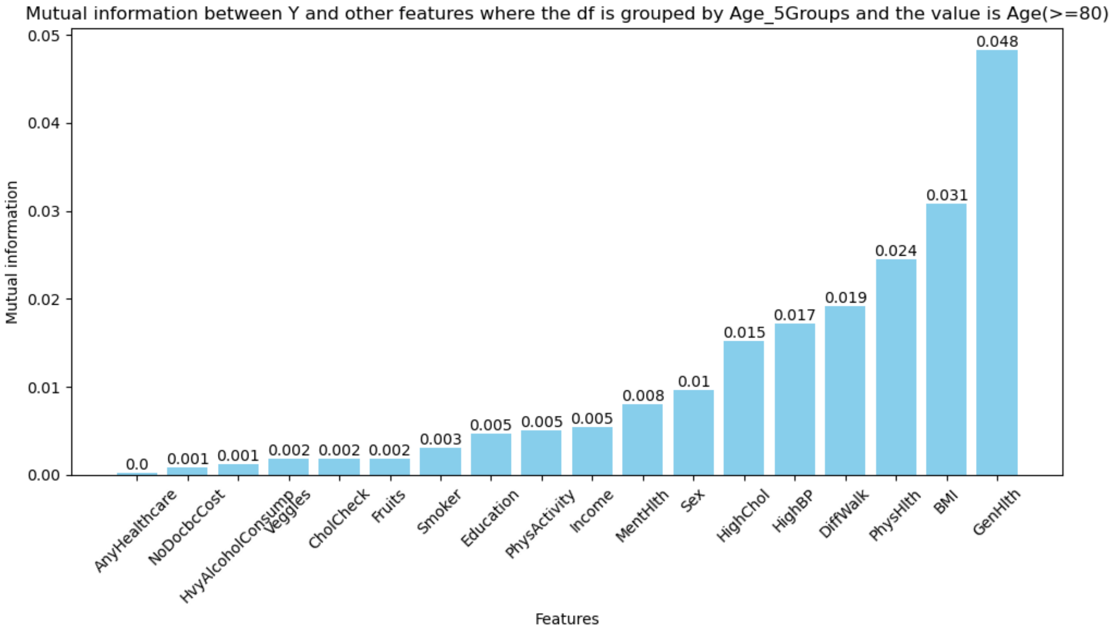
```
[ ]: draw_mutual_information(df, columns_to_drop, 'Age_5Groups')
```






Mutual information between Y and other features where the df is grouped by Age_5Groups and the value is Age(50~64)


Mutual information between Y and other features where the df is grouped by Age_5Groups and the value is Age(65~79)

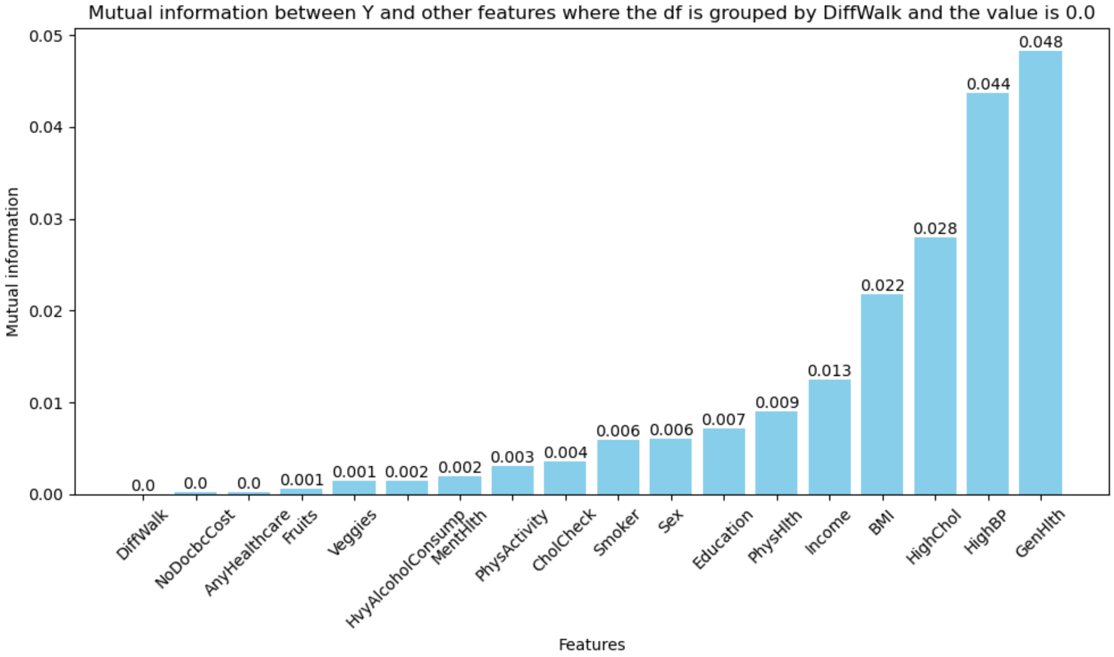


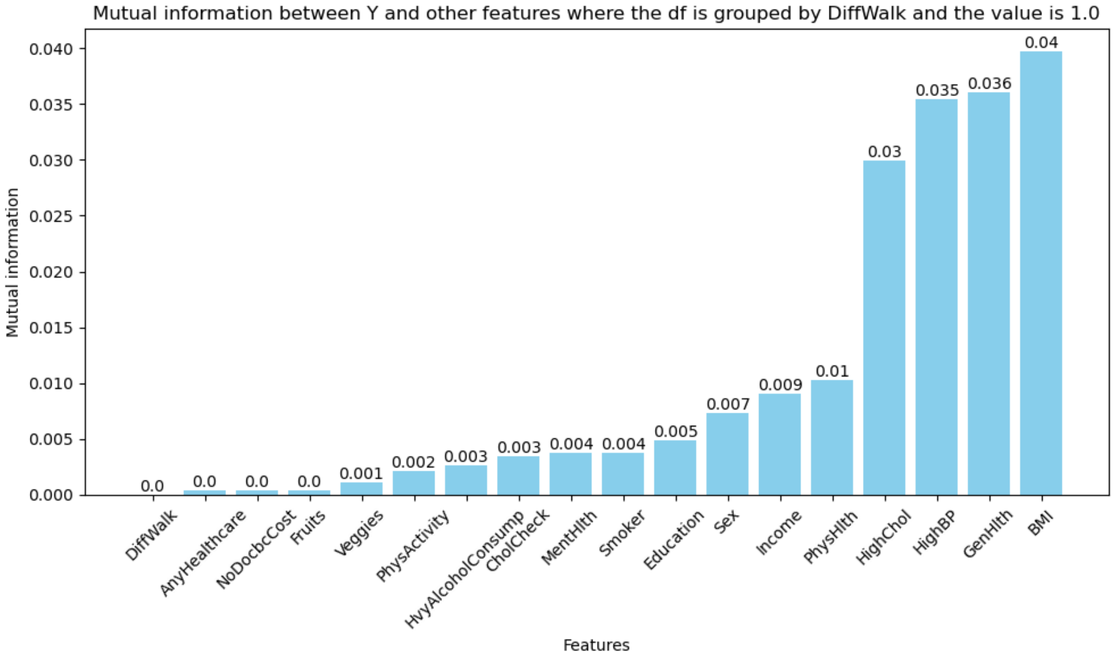
can also change the third parameter to do the graph with df grouped by different feature

Note that the mutual entropy of DiffWalk from the following 2 graphs are 0 because in each df this feature has only 1 value (either '1' or '0')

we could append the DiffWalk to columns_to_drop to ignore this feature

```
[ ]: draw_mutual_information(df, columns_to_drop, 'DiffWalk')
```





5.0.4 Implement Contigency Table or something else below

6 Contingency Table

[ ]:
```
def con_table(columnName, prob=False):
    contingency_table = pd.crosstab(df[columnName], df['Y'])
    print("Contingency Table:")
    print(contingency_table)

    if prob:
        probability_table = contingency_table.div(contingency_table.
 ↪sum(axis=1), axis=0)
        print("\nProbability Table:")
        print(probability_table)
```

[ ]:
```
contingency_table = pd.crosstab(df['Sex'], df['Y'])
print("Contingency Table:")
contingency_table
```

```
Contingency Table:
```

```
[ ]: Y      0,0,0  0,0,1  0,1,0  0,1,1  1,0,0  1,0,1  1,1,0  1,1,1
     Sex
     0.0  111685   2724  16329   1031   5470   1080   2804    851
     1.0   82108   1835  13310    765   7681   1120   4001    886
```