```
"""SEQN: Sequence number (identifier)
RIAGENDR: Respondent's Gender (1=Male, 2=Female)
PAQ605: Physical activity questionnaire response: If the respondent engages in moderate or
BMXBMI: Body Mass Index
LBXGLU: Glucose level
DIQ010: Diabetes questionnaire response
LBXGLT: Glucose tolerance (Oral)
LBXIN: Insulin level"""
```

```
'SEQN: Sequence number (identifier)\nRIAGENDR: Respondent's Gender (1=Male, 2=Female)
\nPAQ605: Physical activity questionnaire response: If the respondent engages in moder
ate or vigorous-intensity sports, fitness, or recreational activities in the typical w
eek\nBMXBMI: Body Mass Index\nLBXGLU: Glucose level\nDIQ010: Diabetes questionnaire re
sponse\nLBXGLT: Glucose tolerance (Oral)\nLBXIN: Insulin level'
```

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')

from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn import tree
from IPython.display import SVG
from graphviz import Source
from IPython.display import display
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score, confusion_matrix, accuracy_score, roc_curve
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
df = pd.read_csv("Train_Data.csv")
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1966 entries, 0 to 1965
Data columns (total 9 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   SEQN       1954 non-null   float64
 1   RIAGENDR   1948 non-null   float64
 2   PAQ605     1953 non-null   float64
 3   BMXBMI     1948 non-null   float64
 4   LBXGLU     1953 non-null   float64
 5   DIQ010     1948 non-null   float64
 6   LBXGLT     1955 non-null   float64
 7   LBXIN      1957 non-null   float64
 8   age_group  1952 non-null   object 
dtypes: float64(8), object(1)
memory usage: 138.4+ KB
```

```
df.head()
```

	SEQN	RIAGENDR	PAQ605	BMXBMI	LBXGLU	DIQ010	LBXGLT	LBXIN	age_group
0	73564.0	2.0	2.0	35.7	110.0	2.0	150.0	14.91	Adult
1	73568.0	2.0	2.0	20.3	89.0	2.0	80.0	3.85	Adult
2	73576.0	1.0	2.0	23.2	89.0	2.0	68.0	6.14	Adult
3	73577.0	1.0	2.0	28.9	104.0	NaN	84.0	16.15	Adult
4	73580.0	2.0	1.0	35.9	103.0	2.0	81.0	10.92	Adult

Next steps: Generate code with df View recommended plots New interactive sheet

```
df.shape
```

```
(1966, 9)
```

```
df.isnull().sum()
```

	0
SEQN	12
RIAGENDR	18
PAQ605	13
BMXBMI	18
LBXGLU	13
DIQ010	18
LBXGLT	11
LBXIN	9
age_group	14

dtype: int64

```
df.describe(include='all').T
```

	count	unique	top	freq	mean	std	min	25%	5
SEQN	1954.0	NaN	NaN	NaN	78683.621801	2924.115709	73564.0	76194.0	7871
RIAGENDR	1948.0	NaN	NaN	NaN	1.510267	0.500023	1.0	1.0	2
PAQ605	1953.0	NaN	NaN	NaN	1.825397	0.399449	1.0	2.0	2
BMXBMI	1948.0	NaN	NaN	NaN	27.9654	7.327616	14.5	22.8	2
LBXGLU	1953.0	NaN	NaN	NaN	99.491039	16.774665	63.0	91.0	9
DIQ010	1948.0	NaN	NaN	NaN	2.015914	0.187579	1.0	2.0	2
LBXGLT	1955.0	NaN	NaN	NaN	115.150384	46.271615	40.0	87.0	10
LBXIN	1957.0	NaN	NaN	NaN	11.862892	9.756713	0.14	5.8	9.
age_group	1952	2	Adult	1638	NaN	NaN	NaN	NaN	N

```
#duplicate data
df.duplicated().sum()
```

```
np.int64(0)
```

```
df.drop("SEQN", axis = 1, inplace = True)
df.drop('PAQ605', axis = 1, inplace = True)
```

```
#Missing data handling
df_copy = df.copy(deep = True)
```

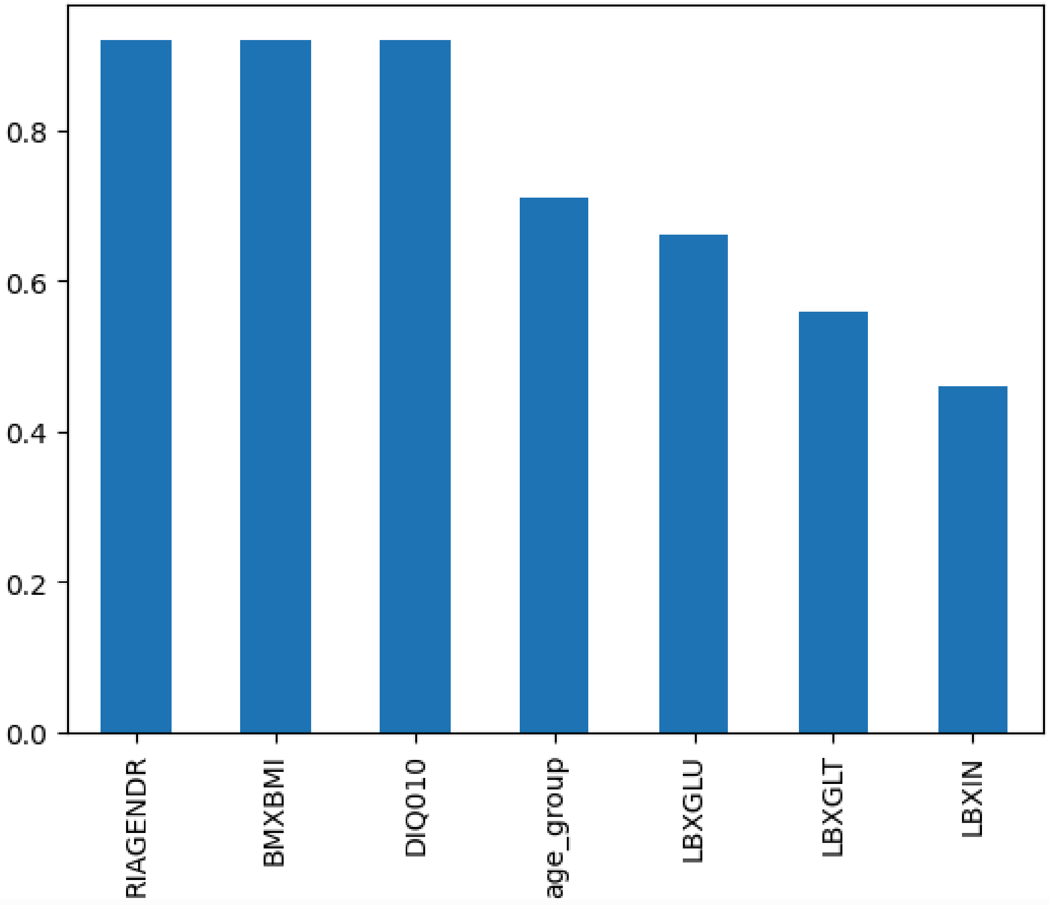
```
# Missing Value Count Function
def show_missing():
    missing = df_copy.columns[df_copy.isnull().any()].tolist()
    return missing

# Missing data counts and percentage
print('Missing Data Count')
print(df_copy[show_missing()].isnull().sum().sort_values(ascending = False))
print('--'*50)
print('Missing Data Percentage')
print(round(df_copy[show_missing()].isnull().sum().sort_values(ascending = False)/len(df_c
```

```
Missing Data Count
RIAGENDR     18
BMXBMI       18
DIQ010       18
age_group    14
LBXGLU       13
LBXGLT       11
LBXIN         9
dtype: int64
----------------------------------------------------------------------------------------------------
Missing Data Percentage
RIAGENDR     0.92
```

```
BMXBMI       0.92
DIQ010       0.92
age_group    0.71
LBXGLU       0.66
LBXGLT       0.56
LBXIN        0.46
dtype: float64
```

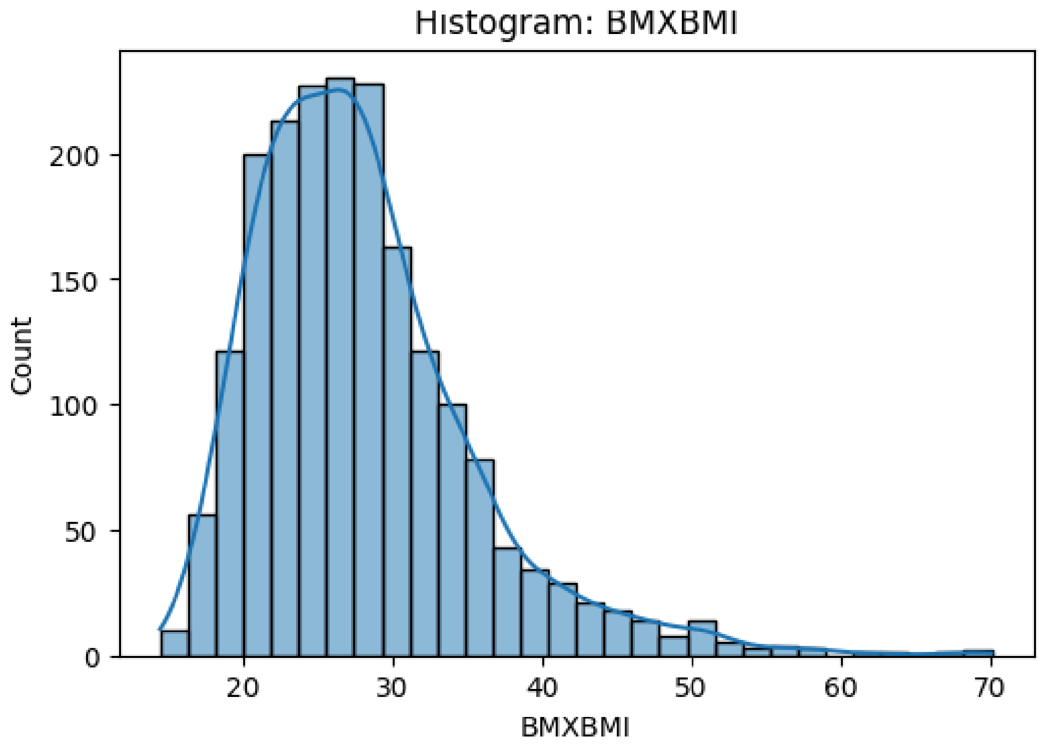
```
#histogram plot
round(df_copy[show_missing()].isnull().sum().sort_values(ascending = False)/len(df_copy)*1
```

```
<Axes: >
```

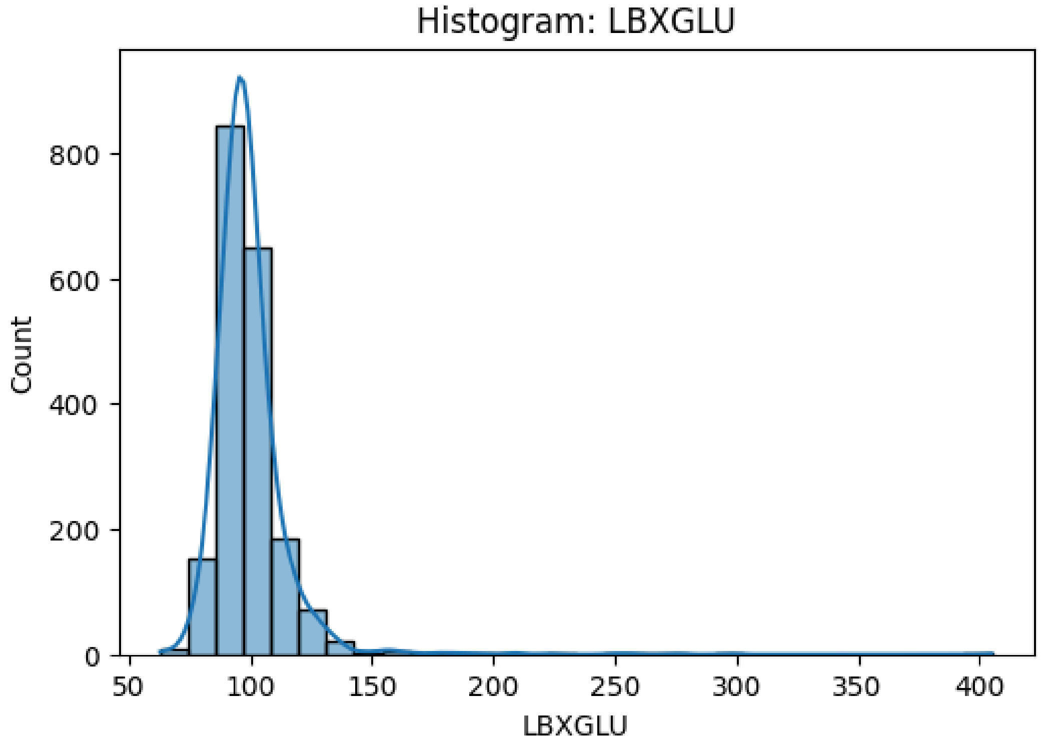


```
numeric_cols = ['BMXBMI', 'LBXGLU', 'LBXGLT', 'LBXIN']

# Histograms
for col in numeric_cols:
    plt.figure(figsize=(6,4))
    sns.histplot(df[col], kde=True, bins=30)
    plt.title(f'Histogram: {col}')
    plt.xlabel(col)
    plt.ylabel('Count')
    plt.show()

# Boxplots with respect to age_group
for col in numeric_cols:
    plt.figure(figsize=(6,4))
    sns.boxplot(data=df, x='age_group', y=col)
```

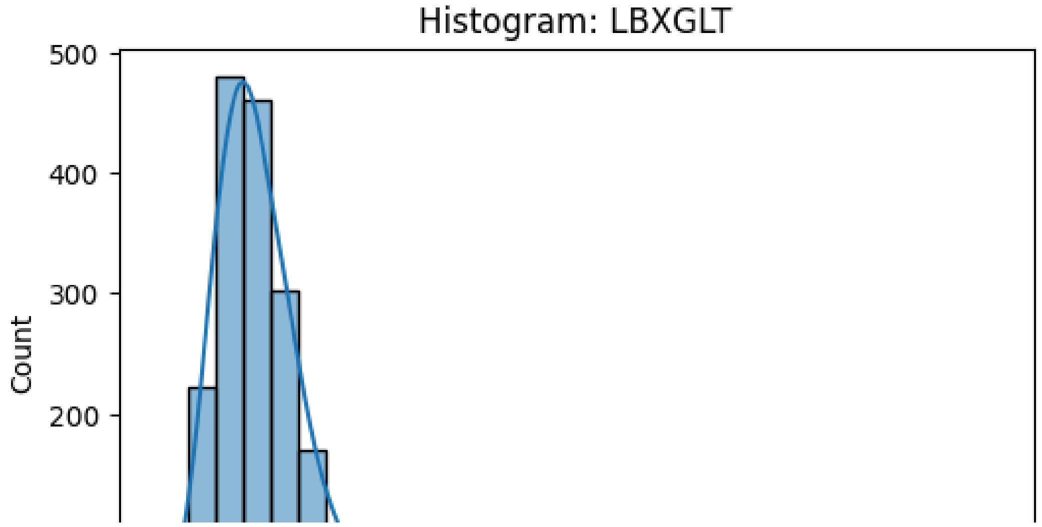
```
plt.title(f'{col} by Age Group')
plt.xlabel('Age Group (0=Adult, 1=Senior)')
plt.ylabel(col)
plt.show()
```



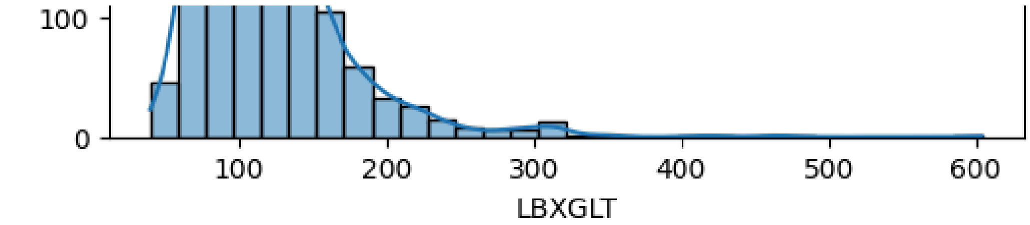
Histogram: BMXBMI



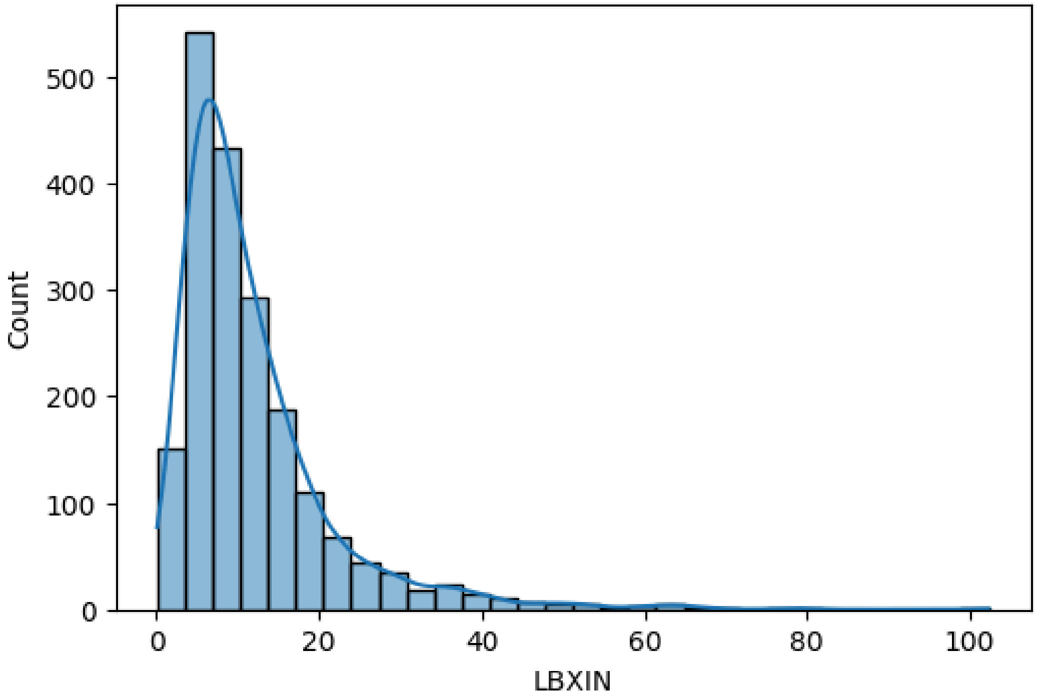
Histogram: LBXGLU



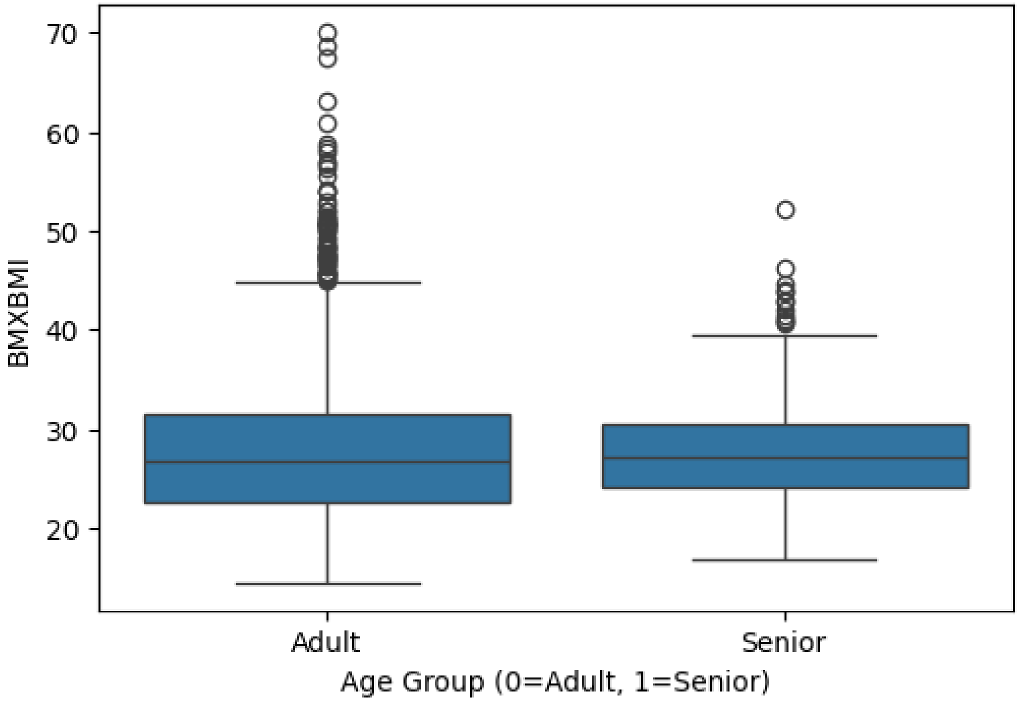
Histogram: LBXGLT



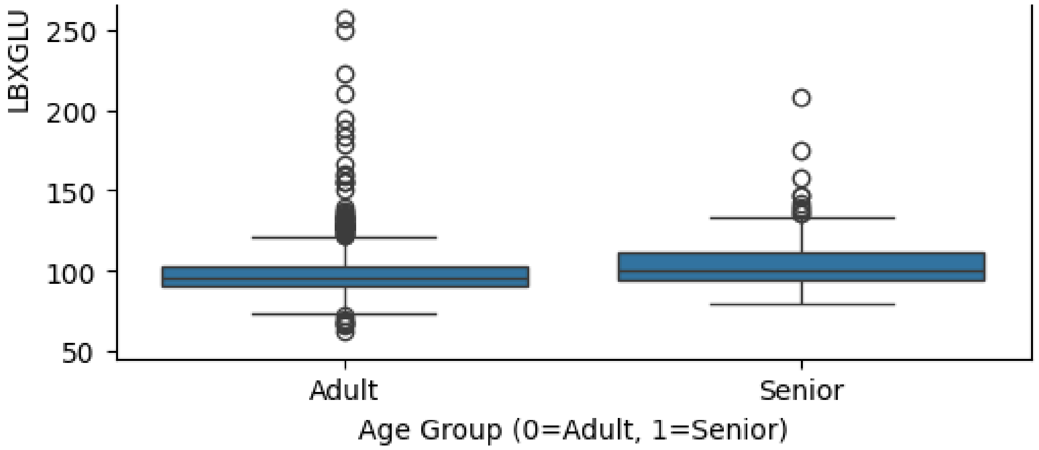
Histogram: LBXIN



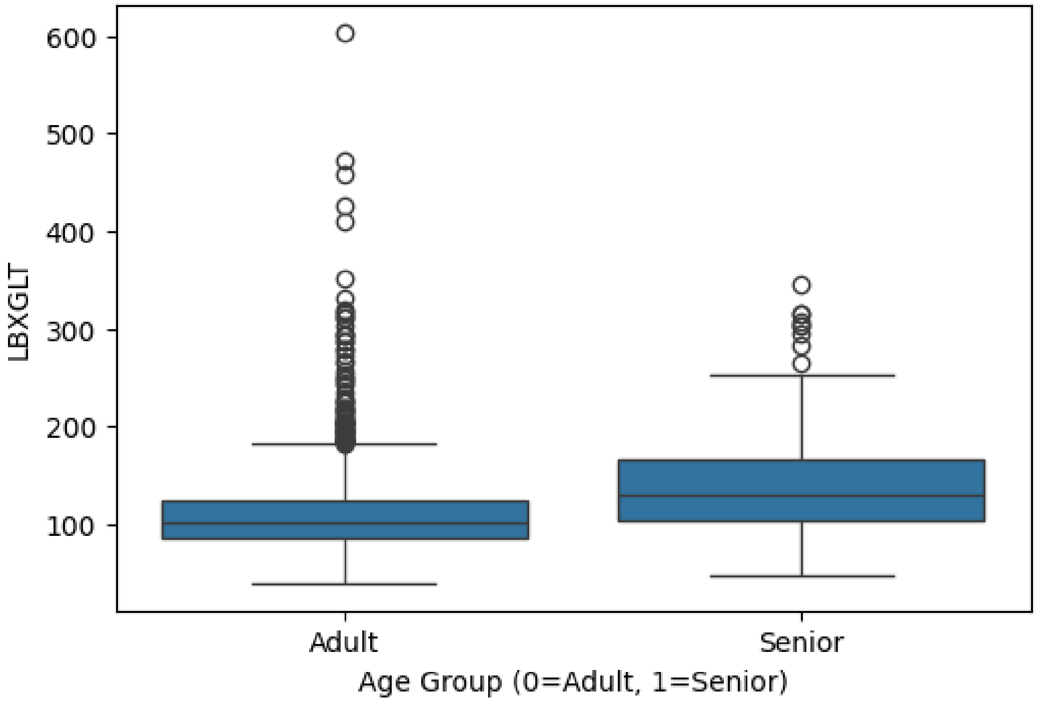
BMXBMI by Age Group



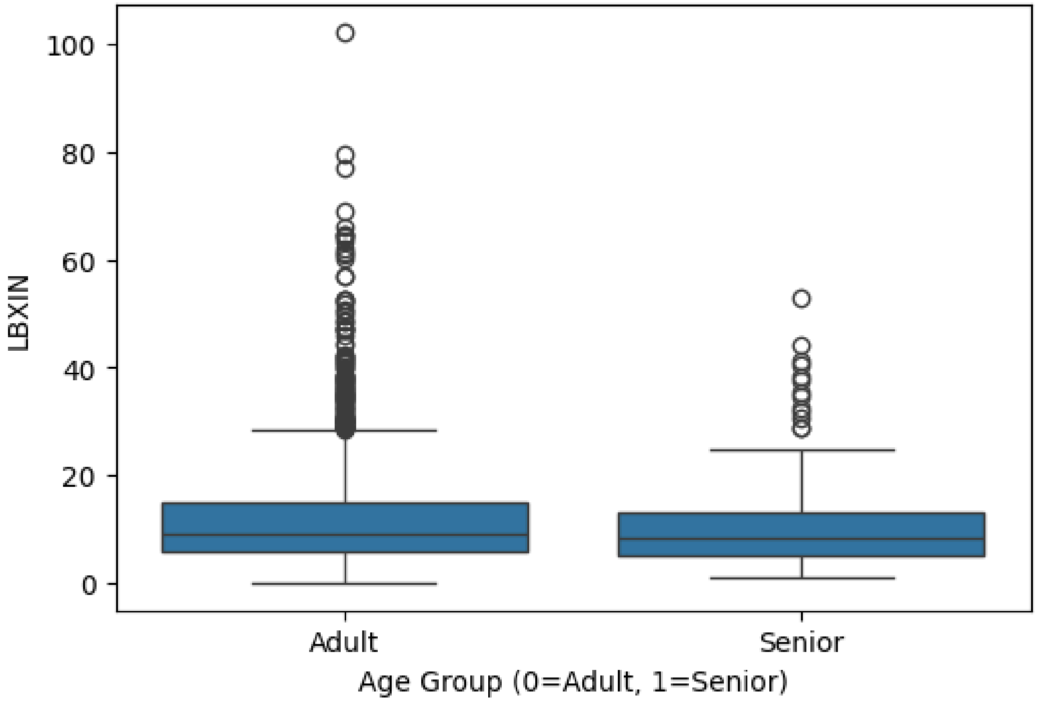
LBXGLU by Age Group



LBXGLT by Age Group



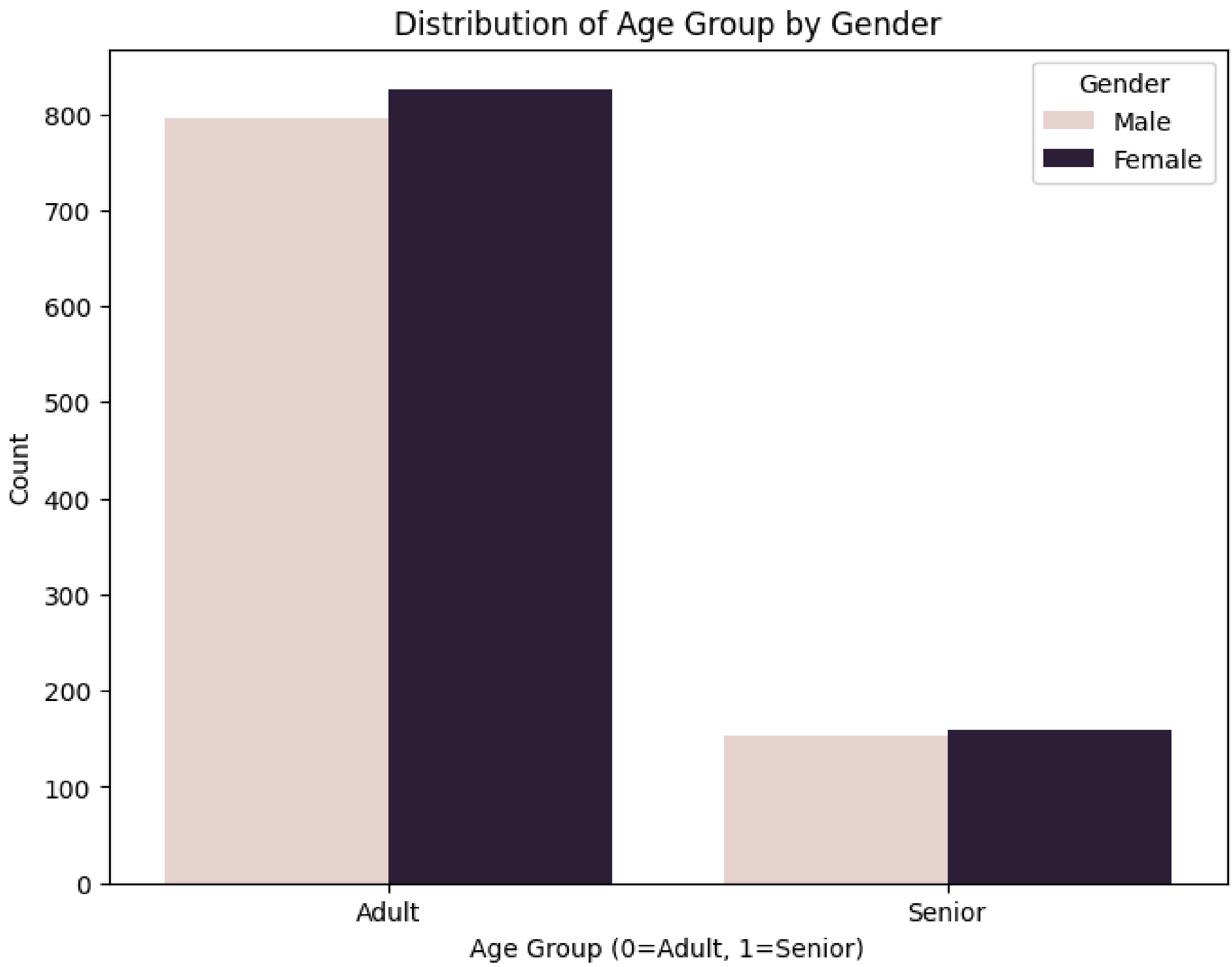
LBXIN by Age Group



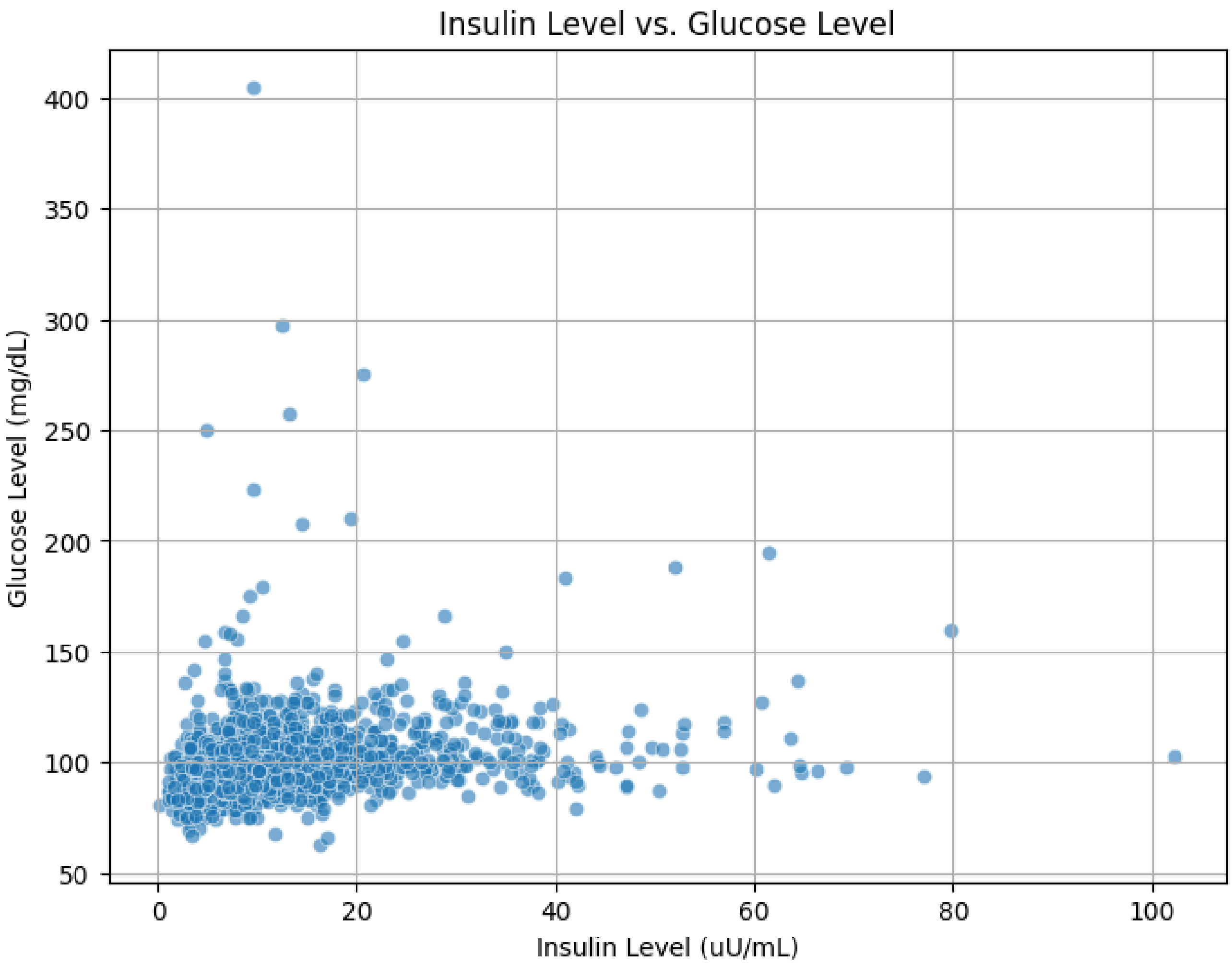
```
# Function to impute missing values
def impute_missing_values(df):
    for col in df.columns:
        if df[col].isnull().any():
            if df[col].dtype in ['int64', 'float64']:
                # Impute numeric columns with median
                df[col].fillna(df[col].median(), inplace=True)
            else:
                # Impute categorical columns with mode
                df[col].fillna(df[col].mode()[0], inplace=True)
    return df

# Impute missing values in the DataFrame copy
df_copy = impute_missing_values(df_copy)

# Verify that there are no more missing values
print('Missing Data Count after imputation')
print(df_copy[show_missing()].isnull().sum().sort_values(ascending = False))
```

```
Missing Data Count after imputation
Series([], dtype: float64)
```

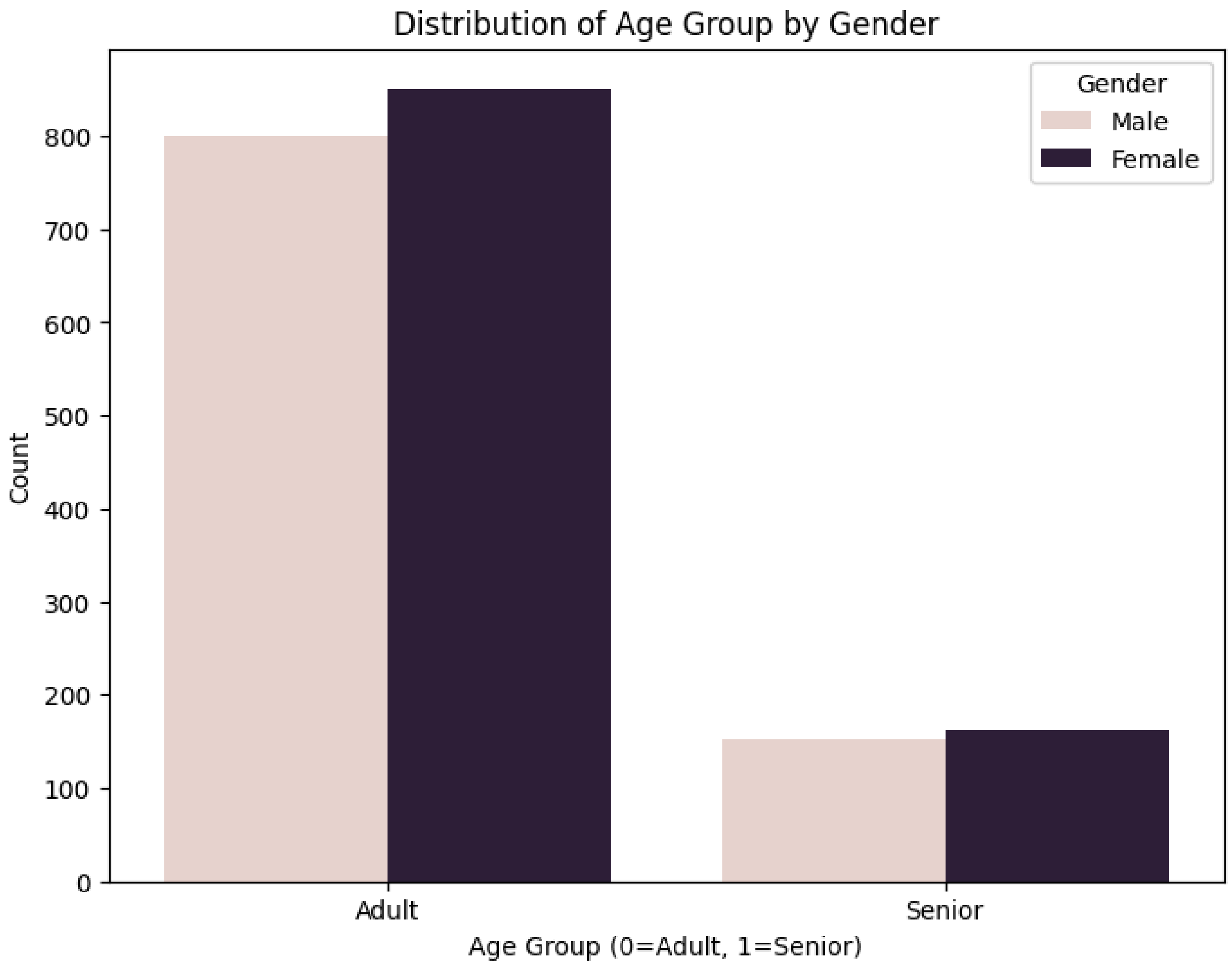
```
# Age vs. Gender
plt.figure(figsize=(8, 6))
sns.countplot(data=df, x='age_group', hue='RIAGENDR')
plt.title('Distribution of Age Group by Gender')
plt.xlabel('Age Group (0=Adult, 1=Senior)')
plt.ylabel('Count')
plt.xticks([0, 1], ['Adult', 'Senior'])
plt.legend(title='Gender', labels=['Male', 'Female'])
plt.show()
```



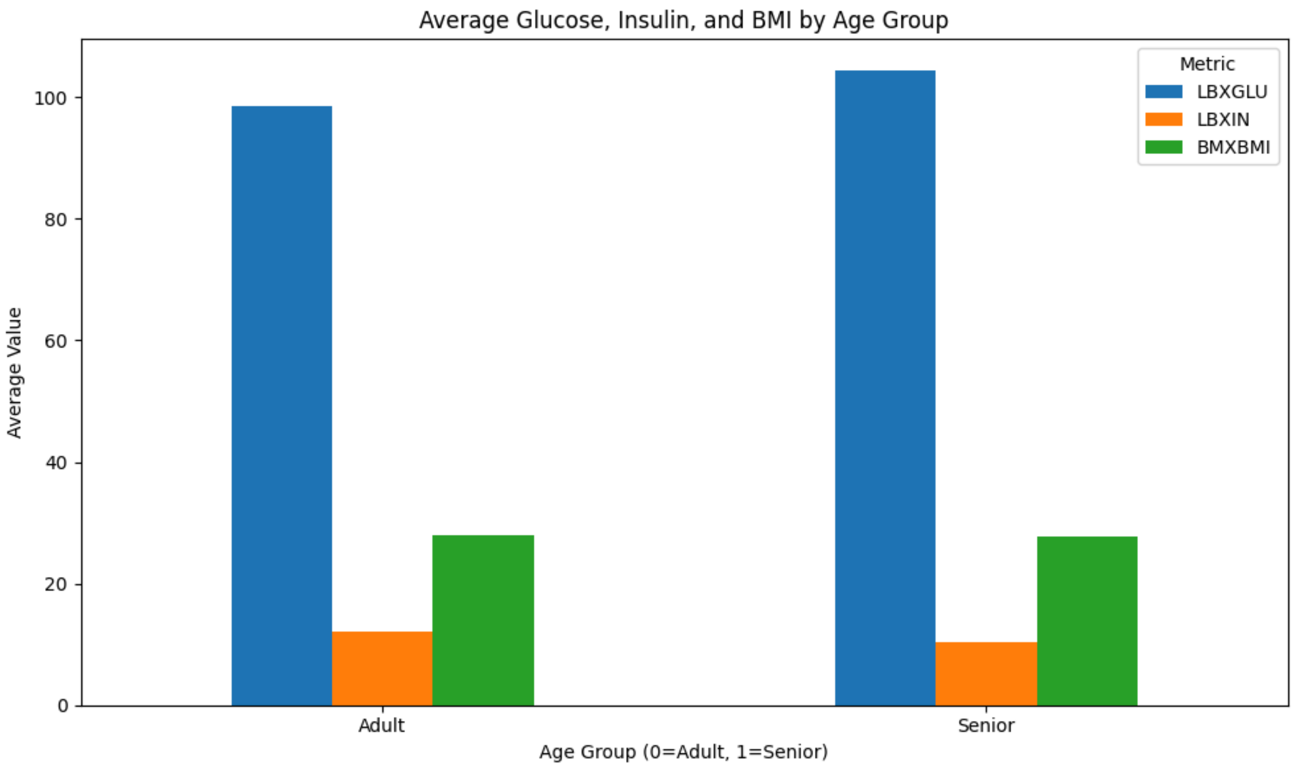
```
plt.figure(figsize=(8, 6))
sns.scatterplot(data=df_copy, x='LBXIN', y='LBXGLU', alpha=0.6)
plt.title('Insulin Level vs. Glucose Level')
plt.xlabel('Insulin Level (uU/mL)')
plt.ylabel('Glucose Level (mg/dL)')
plt.grid(True)
plt.show()
```



```
plt.figure(figsize=(8, 6))
sns.countplot(data=df_copy, x='age_group', hue='RIAGENDR')
plt.title('Distribution of Age Group by Gender')
plt.xlabel('Age Group (0=Adult, 1=Senior)')
plt.ylabel('Count')
plt.xticks([0, 1], ['Adult', 'Senior'])
plt.legend(title='Gender', labels=['Male', 'Female'])
plt.show()
```



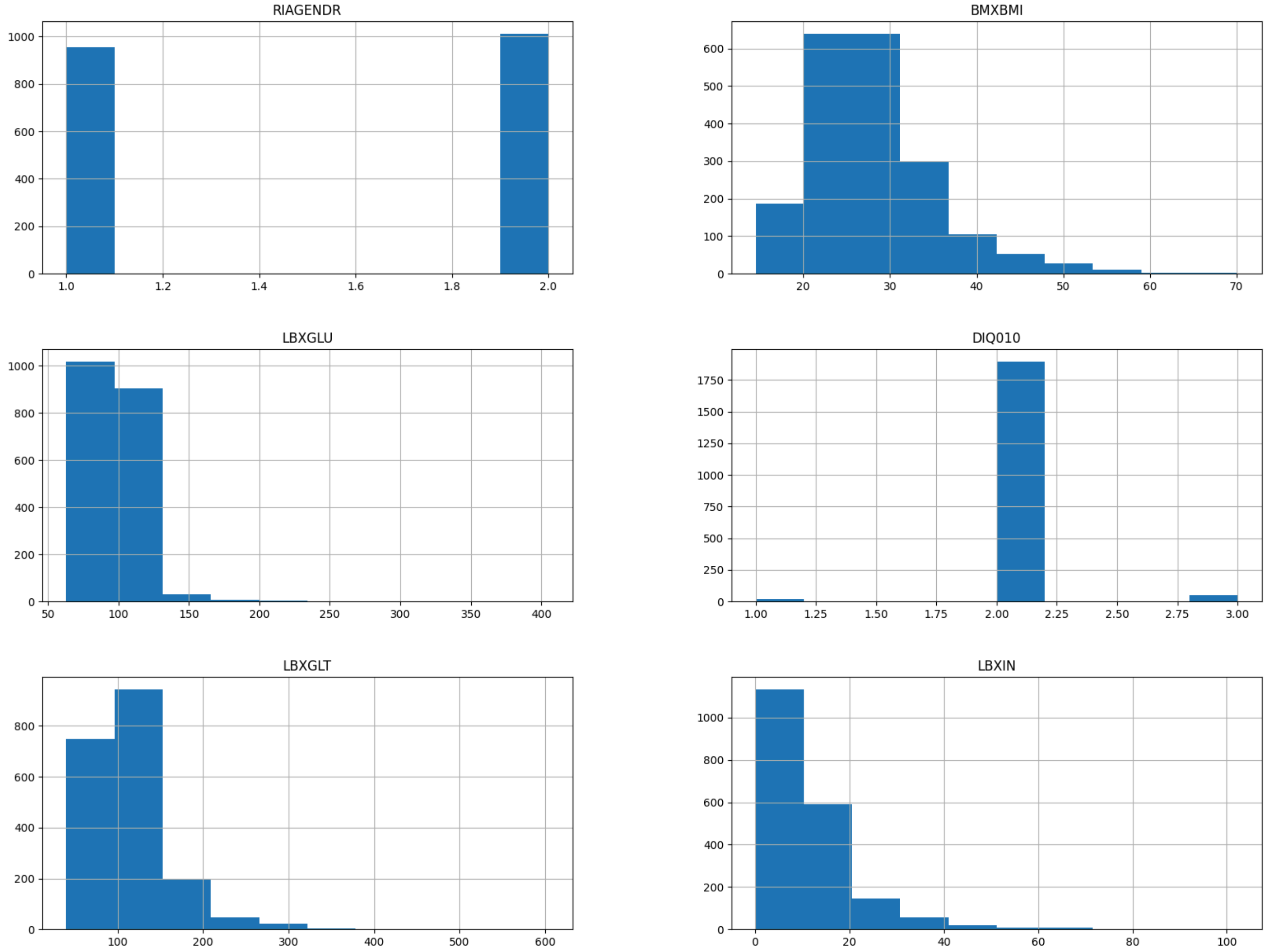
```
grouped_data = df_copy.groupby('age_group')[['LBXGLU', 'LBXIN', 'BMXBMI']].mean()

# Plotting the grouped bar chart
grouped_data.plot(kind='bar', figsize=(10, 6))
plt.title('Average Glucose, Insulin, and BMI by Age Group')
plt.xlabel('Age Group (0=Adult, 1=Senior)')
plt.ylabel('Average Value')
plt.xticks(ticks=[0, 1], labels=['Adult', 'Senior'], rotation=0)
plt.legend(title='Metric')
plt.tight_layout()
plt.show()

# Assuming 'Stroke' column exists in your DataFrame for stroke distribution
if 'Stroke' in df_copy.columns:
    plt.figure(figsize=(6, 4))
    sns.countplot(data=df_copy, x='Stroke')
    plt.title('Distribution of Stroke')
    plt.xlabel('Stroke (0=No, 1=Yes)')
    plt.ylabel('Count')
    plt.xticks([0, 1], ['No Stroke', 'Stroke'])
    plt.show()
else:
    print("The 'Stroke' column is not found in the DataFrame. Cannot plot stroke distributio
```



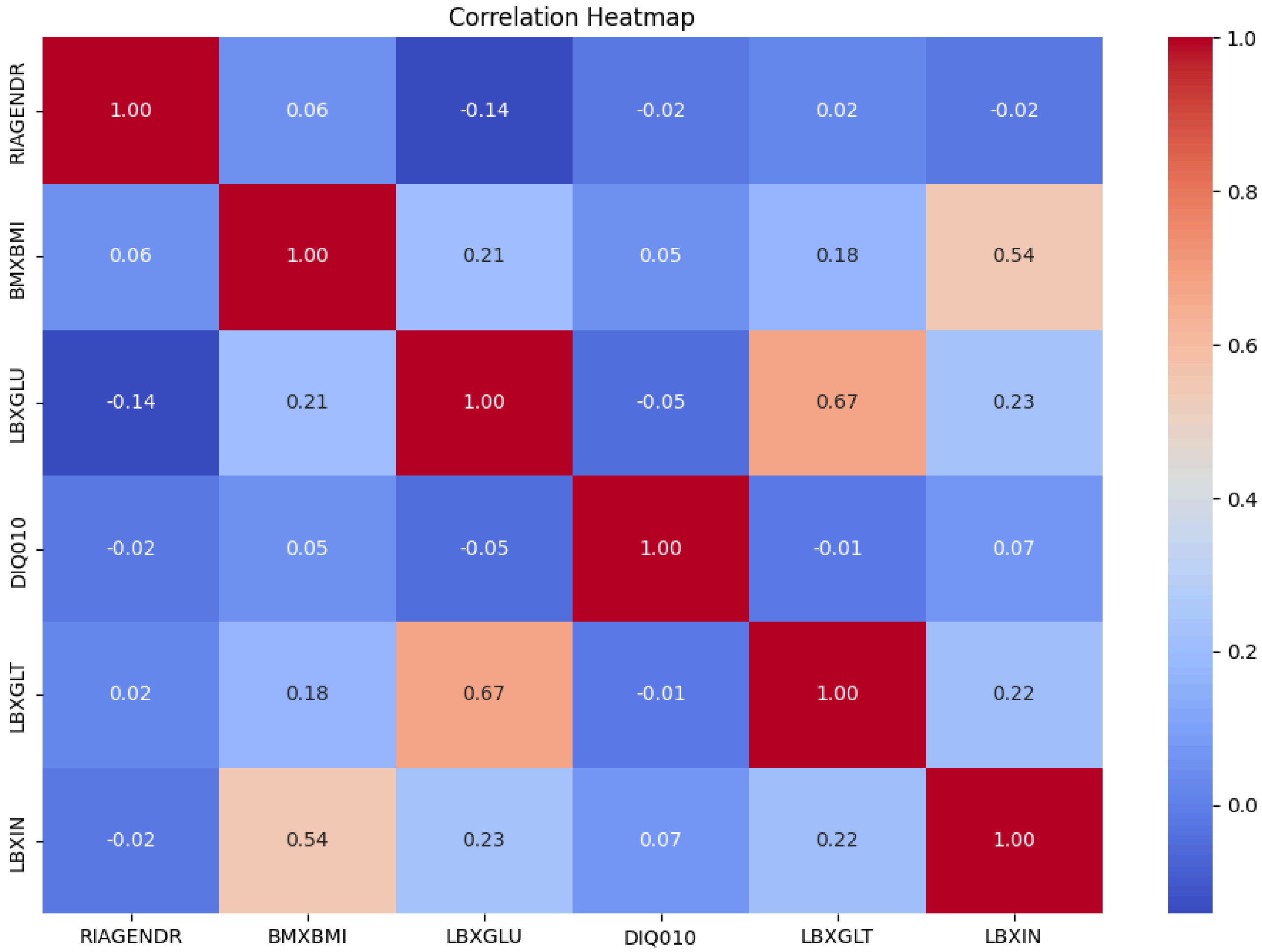
```
The 'Stroke' column is not found in the DataFrame. Cannot plot stroke distribution.
```

```
fig = plt.figure(figsize = (20,15))
ax = fig.gca()
df_copy.hist(ax = ax)
```

```
array([[<Axes: title={'center': 'RIAGENDR'}>,
        <Axes: title={'center': 'BMXBMI'}>],
       [<Axes: title={'center': 'LBXGLU'}>,
        <Axes: title={'center': 'DIQ010'}>],
       [<Axes: title={'center': 'LBXGLT'}>,
        <Axes: title={'center': 'LBXIN'}>]], dtype=object)
```



```
plt.figure(figsize=(12, 8))
# Exclude non-numeric columns before calculating correlation
numeric_df = df_copy.select_dtypes(include=np.number)
sns.heatmap(numeric_df.corr(), annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Heatmap')
plt.show()
```



```
from sklearn.model_selection import train_test_split

X = df_copy.drop('age_group', axis=1)
y = df_copy['age_group']

# Convert categorical features to numerical using one-hot encoding
X = pd.get_dummies(X)
test = pd.read_csv('Test_Data.csv')
test = pd.get_dummies(test)
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)

# Print the shapes of the resulting datasets
print("Shape of X_train:", X_train.shape)
print("Shape of X_test:", X_test.shape)
print("Shape of y_train:", y_train.shape)
print("Shape of y_test:", y_test.shape)
```

```
Shape of X_train: (1474, 6)
Shape of X_test: (492, 6)
Shape of y_train: (1474,)
Shape of y_test: (492,)
```