for-440-model

April 12, 2025

[12]:
```python
# imports
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt
```

Read The **Dataset**

[13]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Read the dataset
df = pd.read_csv('/Dataset - Updated.csv')

# Display basic info
print(df.info())
print(df.head())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1205 entries, 0 to 1204
Data columns (total 12 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Age                      1205 non-null   int64
 1   Systolic BP              1200 non-null   float64
 2   Diastolic                1201 non-null   float64
 3   BS                       1203 non-null   float64
 4   Body Temp                1205 non-null   int64
 5   BMI                      1187 non-null   float64
 6   Previous Complications   1203 non-null   float64
 7   Preexisting Diabetes     1203 non-null   float64
 8   Gestational Diabetes     1205 non-null   int64
 9   Mental Health            1205 non-null   int64
 10  Heart Rate               1203 non-null   float64
 11  Risk Level               1187 non-null   object
dtypes: float64(7), int64(4), object(1)
memory usage: 113.1+ KB
```

```
None
   Age  Systolic BP  Diastolic   BS  Body Temp   BMI  Previous Complications  \
0   22         90.0       60.0  9.0        100  18.0                     1.0
1   22        110.0       70.0  7.1         98  20.4                     0.0
2   27        110.0       70.0  7.5         98  23.0                     1.0
3   20        100.0       70.0  7.2         98  21.2                     0.0
4   20         90.0       60.0  7.5         98  19.7                     0.0

   Preexisting Diabetes  Gestational Diabetes  Mental Health  Heart Rate  \
0                   1.0                     0              1        80.0
1                   0.0                     0              0        74.0
2                   0.0                     0              0        72.0
3                   0.0                     0              0        74.0
4                   0.0                     0              0        74.0

  Risk Level
0       High
1        Low
2        Low
3        Low
4        Low
```

[14]:
```
# #tells the number of rows and columns of a given DataFrame
print("Shape of the dataset", df.shape)
df.shape

#printing number of rows and number of columns
print('Number of rows: ', df.shape[0])
print('Number of columns: ', df.shape[1])
```

```
Shape of the dataset (1205, 12)
Number of rows:  1205
Number of columns:  12
```

[15]:
```
df.head() #Shows us the first 5 rows of the dataset
```

[15]:
```
   Age  Systolic BP  Diastolic   BS  Body Temp   BMI  Previous Complications  \
0   22         90.0       60.0  9.0        100  18.0                     1.0
1   22        110.0       70.0  7.1         98  20.4                     0.0
2   27        110.0       70.0  7.5         98  23.0                     1.0
3   20        100.0       70.0  7.2         98  21.2                     0.0
4   20         90.0       60.0  7.5         98  19.7                     0.0

   Preexisting Diabetes  Gestational Diabetes  Mental Health  Heart Rate  \
0                   1.0                     0              1        80.0
1                   0.0                     0              0        74.0
2                   0.0                     0              0        72.0
```

```
3                   0.0                        0              0        74.0
4                   0.0                        0              0        74.0

  Risk Level
0       High
1        Low
2        Low
3        Low
4        Low
```

[16]:
```
df.size # number of total elements in the dataset; 1205 \times 12
```

[16]:
```
14460
```

[17]:
```
#printing the column names in the form of a list
col_list = [] # this is an empty list
for x in df.columns:
    col_list.append(x)

col_list
```

[17]:
```
['Age',
 'Systolic BP',
 'Diastolic',
 'BS',
 'Body Temp',
 'BMI',
 'Previous Complications',
 'Preexisting Diabetes',
 'Gestational Diabetes',
 'Mental Health',
 'Heart Rate',
 'Risk Level']
```

[18]:
```
# Convert column names to lowercase and replace spaces with underscores
df.columns = df.columns.str.lower().str.replace(' ', '_')
```

[19]:
```
# printing Basic infos (Non-null count and data type) per feature of the dataset
print(df.info());
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1205 entries, 0 to 1204
Data columns (total 12 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   age                     1205 non-null   int64
 1   systolic_bp             1200 non-null   float64
```

```
 2   diastolic                1201 non-null   float64
 3   bs                       1203 non-null   float64
 4   body_temp                1205 non-null   int64
 5   bmi                      1187 non-null   float64
 6   previous_complications   1203 non-null   float64
 7   preexisting_diabetes     1203 non-null   float64
 8   gestational_diabetes     1205 non-null   int64
 9   mental_health            1205 non-null   int64
 10  heart_rate               1203 non-null   float64
 11  risk_level               1187 non-null   object
dtypes: float64(7), int64(4), object(1)
memory usage: 113.1+ KB
None
```

[20]:

```
#printing basic statistical infos per feature of the dataset
df.describe()
```

[20]:

```
               age  systolic_bp    diastolic           bs    body_temp  \
count  1205.000000  1200.000000  1201.000000  1203.000000  1205.000000
mean     27.731950   116.819167    77.166528     7.501064    98.395851
std      12.571074    18.715502    14.305148     3.049522     1.088363
min      10.000000    70.000000    40.000000     3.000000    97.000000
25%      21.000000   100.000000    65.000000     6.000000    98.000000
50%      25.000000   120.000000    80.000000     6.900000    98.000000
75%      32.000000   130.000000    90.000000     7.900000    98.000000
max     325.000000   200.000000   140.000000    19.000000   103.000000

               bmi  previous_complications  preexisting_diabetes  \
count  1187.000000             1203.000000           1203.000000
mean     23.315080                0.175395              0.288446
std       3.875682                0.380463              0.453228
min       0.000000                0.000000              0.000000
25%      20.450000                0.000000              0.000000
50%      23.000000                0.000000              0.000000
75%      25.000000                0.000000              1.000000
max      37.000000                1.000000              1.000000

       gestational_diabetes  mental_health   heart_rate
count           1205.000000     1205.00000  1203.000000
mean               0.117842        0.33444    75.817124
std                0.322555        0.47199     7.227338
min                0.000000        0.00000    58.000000
25%                0.000000        0.00000    70.000000
50%                0.000000        0.00000    76.000000
75%                0.000000        1.00000    80.000000
max                1.000000        1.00000    92.000000
```

1 New Section

[21]:
```
for col in df:
    print(f'{col}: {df[col].unique()}')
```

```
age: [ 22  27  20  23  26  25  19  18  24  21  44  17  28  40  37  29  34  35
  36  32  38  48  30  39  31  33  41 325  42  15  50  63  55  49  16  12
  60  65  43  13  54  10  45]
systolic_bp: [ 90. 110. 100. 120. 140. 130. 150. 160. 180. 170. 200. 115.  80.
105.
 125. 135.  nan  85.  95.  76. 129.  99.  70.  78.  75.]
diastolic: [ 60.  70.  80. 100.  90. 110.  96.  65.  85. 120. 140. 130.  95.
79.
  75.  nan  49.  63.  50.  45.  55.  40.]
bs: [ 9.    7.1   7.5   7.2   7.01  7.    6.4  12.    9.9   6.    6.5   6.6
 14.    8.    6.2   7.3   6.9   7.9   6.7  11.    6.1  18.   13.   15.
 17.   19.    7.7   7.6   7.8   5.3   5.7   7.4   5.2   5.5   6.3  10.
 16.    8.6   8.4   8.3   8.1   8.2   8.7   8.9   8.8   9.8   9.7   5.6
  4.5   5.8   4.9   6.8   3.9   4.7   3.6   4.1   4.4   4.6   5.01  6.02
  5.9   4.    4.3   4.8   4.2   5.    3.4   4.03  6.04  3.7   3.5   3.
  3.8   3.3   3.01  4.01  5.07  6.09  4.07  8.01   nan  5.1   9.3  11.1
 11.5 ]
body_temp: [100  98 102 101  97  99 103]
bmi: [18.  20.4 23.  21.2 19.7 24.  17.6 21.3 22.  30.2 24.5 30.  21.  21.5
  nan 18.6 19.  20.  25.3 23.8 20.2 23.7 24.2 25.  19.3 18.8 23.4 22.8
 18.5 30.1 26.6 23.2 22.9 22.5 19.4 33.  26.  25.2 25.7 27.  25.5 18.9
 23.1 24.4 17.  30.3 31.  24.7 24.8 20.5 32.  26.7 28.  24.9 25.9 27.2
 32.2 29.9 26.2 31.4 25.8 32.4 31.3 27.5 18.4 18.2 25.4 18.3 31.6 17.8
 29.  26.5 26.4 27.4 32.9 29.6 30.5 28.4 35.  29.7 30.6 29.3 28.5 25.6
 33.3 29.8 26.9 24.1 22.4 23.5 22.1 24.3 23.9 21.1 21.6 22.2 26.3 18.7
 19.6 19.2 30.7 25.1 27.3 34.  16.  26.8 27.6 28.7 31.5 28.9 20.9 27.9
 17.3 23.3 22.3 27.7 27.8 23.6 16.8 32.3 17.2 28.1 29.5 33.4 31.9 17.5
 18.1 30.4 29.1 19.1 26.1 19.5 19.8 19.9  0.  20.1 21.8 22.7 21.9 28.2
 21.7 37.  27.1 35.1 17.9 24.6 36.  15.  17.7 15.6 15.5 16.6 15.9 34.5
 16.9 31.1 28.3 17.1]
previous_complications: [ 1.  0. nan]
preexisting_diabetes: [ 1.  0. nan]
gestational_diabetes: [0 1]
mental_health: [1 0]
heart_rate: [80. 74. 72. 76. 78. 84. 66. 70. 88. 68. 77. 60. 90. 86. 62. 82. 85.
92.
 79. 87. 64. 89. 83. nan 67. 65. 75. 69. 71. 73. 81. 58.]
risk_level: ['High' 'Low' nan]
```

[22]:
```
#prints maximum values of Age features
for col in df.iloc[:, :-1]:  # Exclude the Y label column
    print(f'{col}: {df[col].max()}')
```

```
age: 325
systolic_bp: 200.0
diastolic: 140.0
bs: 19.0
body_temp: 103
bmi: 37.0
previous_complications: 1.0
preexisting_diabetes: 1.0
gestational_diabetes: 1
mental_health: 1
heart_rate: 92.0
```

[23]:
```
df.isnull().sum() # check how many missing values in the data
```

[23]:
```
age                        0
systolic_bp                5
diastolic                  4
bs                         2
body_temp                  0
bmi                       18
previous_complications     2
preexisting_diabetes       2
gestational_diabetes       0
mental_health              0
heart_rate                 2
risk_level                18
dtype: int64
```

[24]:
```
df.describe()
```

[24]:
```
               age  systolic_bp    diastolic           bs    body_temp  \
count  1205.000000  1200.000000  1201.000000  1203.000000  1205.000000
mean     27.731950   116.819167    77.166528     7.501064    98.395851
std      12.571074    18.715502    14.305148     3.049522     1.088363
min      10.000000    70.000000    40.000000     3.000000    97.000000
25%      21.000000   100.000000    65.000000     6.000000    98.000000
50%      25.000000   120.000000    80.000000     6.900000    98.000000
75%      32.000000   130.000000    90.000000     7.900000    98.000000
max     325.000000   200.000000   140.000000    19.000000   103.000000

               bmi  previous_complications  preexisting_diabetes  \
count  1187.000000             1203.000000           1203.000000
mean     23.315080                0.175395              0.288446
std       3.875682                0.380463              0.453228
min       0.000000                0.000000              0.000000
25%      20.450000                0.000000              0.000000
50%      23.000000                0.000000              0.000000
```

```
75%      25.000000                0.000000               1.000000
max      37.000000                1.000000               1.000000

       gestational_diabetes  mental_health   heart_rate
count           1205.000000     1205.00000  1203.000000
mean               0.117842        0.33444    75.817124
std                0.322555        0.47199     7.227338
min                0.000000        0.00000    58.000000
25%                0.000000        0.00000    70.000000
50%                0.000000        0.00000    76.000000
75%                0.000000        1.00000    80.000000
max                1.000000        1.00000    92.000000
```

[25]:
```
df.drop_duplicates(inplace = True) #Remove Duplicate values
```

[26]:
```
df.describe()
```

[26]:
```
               age  systolic_bp    diastolic           bs    body_temp  \
count  1187.000000  1182.000000  1183.000000  1185.000000  1187.000000
mean     27.796125   116.804569    77.192730     7.507831    98.401853
std      12.616094    18.814254    14.384438     3.069674     1.095490
min      10.000000    70.000000    40.000000     3.000000    97.000000
25%      21.000000   100.000000    65.000000     6.000000    98.000000
50%      25.000000   120.000000    80.000000     6.900000    98.000000
75%      32.000000   130.000000    90.000000     8.000000    98.000000
max     325.000000   200.000000   140.000000    19.000000   103.000000

               bmi  previous_complications  preexisting_diabetes  \
count  1169.000000             1185.000000           1185.000000
mean     23.334559                0.177215              0.291983
std       3.894783                0.382012              0.454867
min       0.000000                0.000000              0.000000
25%      20.500000                0.000000              0.000000
50%      23.000000                0.000000              0.000000
75%      25.100000                0.000000              1.000000
max      37.000000                1.000000              1.000000

       gestational_diabetes  mental_health   heart_rate
count           1187.000000    1187.000000  1185.000000
mean               0.119629       0.339511    75.856540
std                0.324664       0.473743     7.251142
min                0.000000       0.000000    58.000000
25%                0.000000       0.000000    70.000000
50%                0.000000       0.000000    76.000000
75%                0.000000       1.000000    80.000000
max                1.000000       1.000000    92.000000
```

[27]:
```
#Since Y label has null values we directly remove them instead of imputation
df = df.dropna(subset=['risk_level'])
```

[28]:
```
df.isnull().sum()
```

[28]:
```
age                        0
systolic_bp                4
diastolic                  2
bs                         1
body_temp                  0
bmi                       14
previous_complications     1
preexisting_diabetes       1
gestational_diabetes       0
mental_health              0
heart_rate                 1
risk_level                 0
dtype: int64
```

[29]:
```
#Replace the missing values for numerical columns with mean: Mean imputation
num_cols = ['systolic_bp', 'diastolic',␣
 ↪'bs','bmi','previous_complications','preexisting_diabetes','heart_rate']
for col in num_cols:
    df[col] = df[col].fillna(df[col].mean())
```

[30]:
```
df.isnull().sum()
```

[30]:
```
age                       0
systolic_bp               0
diastolic                 0
bs                        0
body_temp                 0
bmi                       0
previous_complications    0
preexisting_diabetes      0
gestational_diabetes      0
mental_health             0
heart_rate                0
risk_level                0
dtype: int64
```

2 Label Encoding the Categorical Value

[31]:
```
#Converting the Y label categorical value to Numerical Value using label␣
↪encoding

from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
df['risk_level'] = le.fit_transform(df['risk_level']) #Label Encoding risk␣
↪level feature because it is categorical value

# Inverse transform to label high_risk as 1 and low_risk as 0
df['risk_level'] = df['risk_level'].apply(lambda x: 1 if x == 0 else 0)
```

[32]:
```
df.shape
```

[32]:
```
(1169, 12)
```

3 *Visualization*

[33]:
```
print(df.columns.tolist()) # convert a given array to an ordinary list
```

```
['age', 'systolic_bp', 'diastolic', 'bs', 'body_temp', 'bmi',
'previous_complications', 'preexisting_diabetes', 'gestational_diabetes',
'mental_health', 'heart_rate', 'risk_level']
```

[34]:
```
#make a copy of the dataset after dropping the columns having only two classes
df_cp = df.
↪drop(columns=['previous_complications','preexisting_diabetes','gestational_diabetes','menta
```

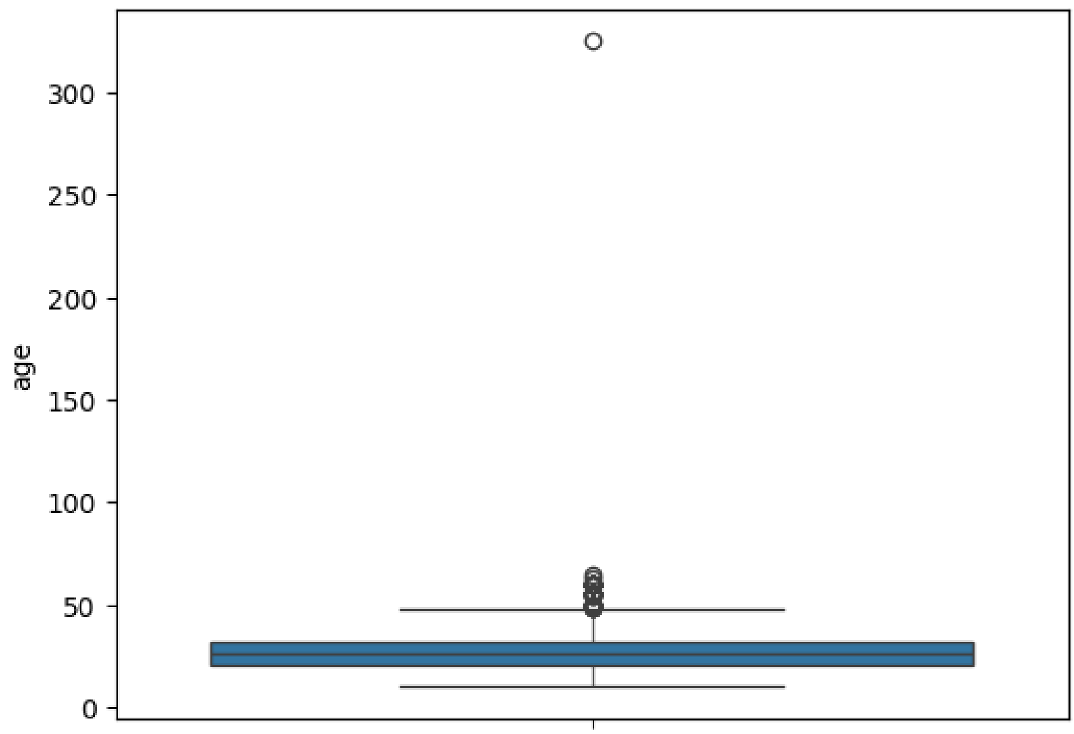
[35]:
```
df.head(10)
```

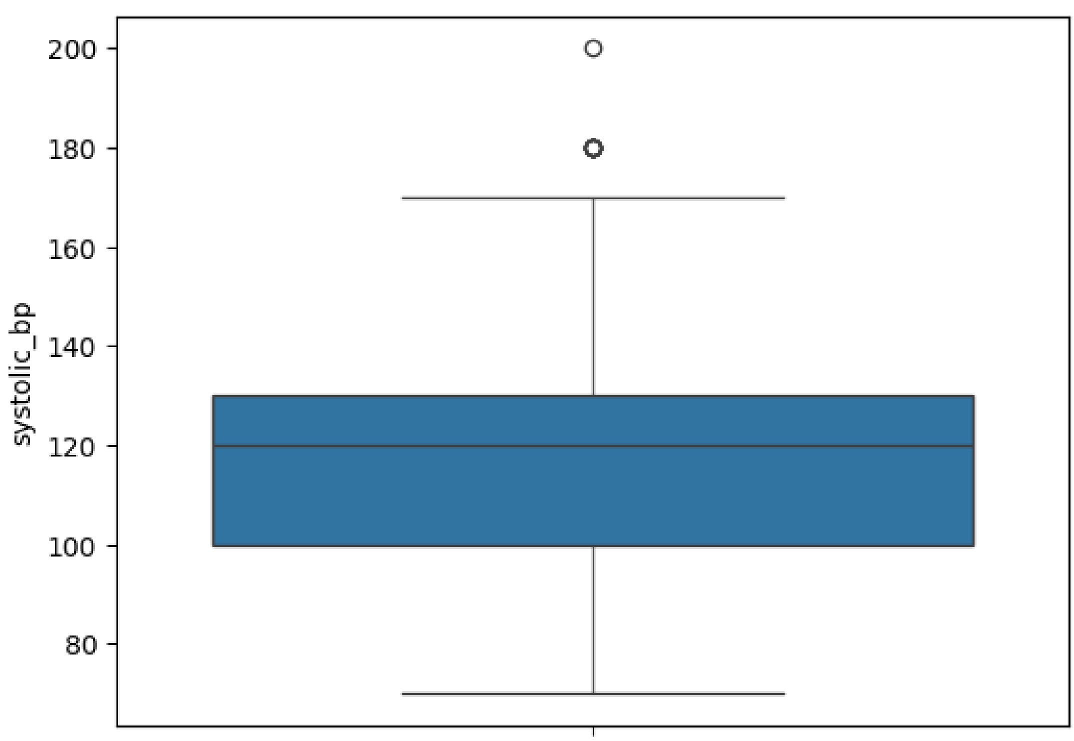
[35]:
```
   age  systolic_bp  diastolic     bs  body_temp   bmi  \
0   22         90.0       60.0   9.00        100  18.0
1   22        110.0       70.0   7.10         98  20.4
2   27        110.0       70.0   7.50         98  23.0
3   20        100.0       70.0   7.20         98  21.2
4   20         90.0       60.0   7.50         98  19.7
5   22        120.0       70.0   7.01         98  24.0
6   20        110.0       70.0   9.00        102  17.6
7   23        110.0       80.0   7.00         98  21.3
8   22         90.0       60.0   6.40         98  22.0
9   26        110.0       70.0  12.00        100  30.2

   previous_complications  preexisting_diabetes  gestational_diabetes  \
0                     1.0                   1.0                     0
```

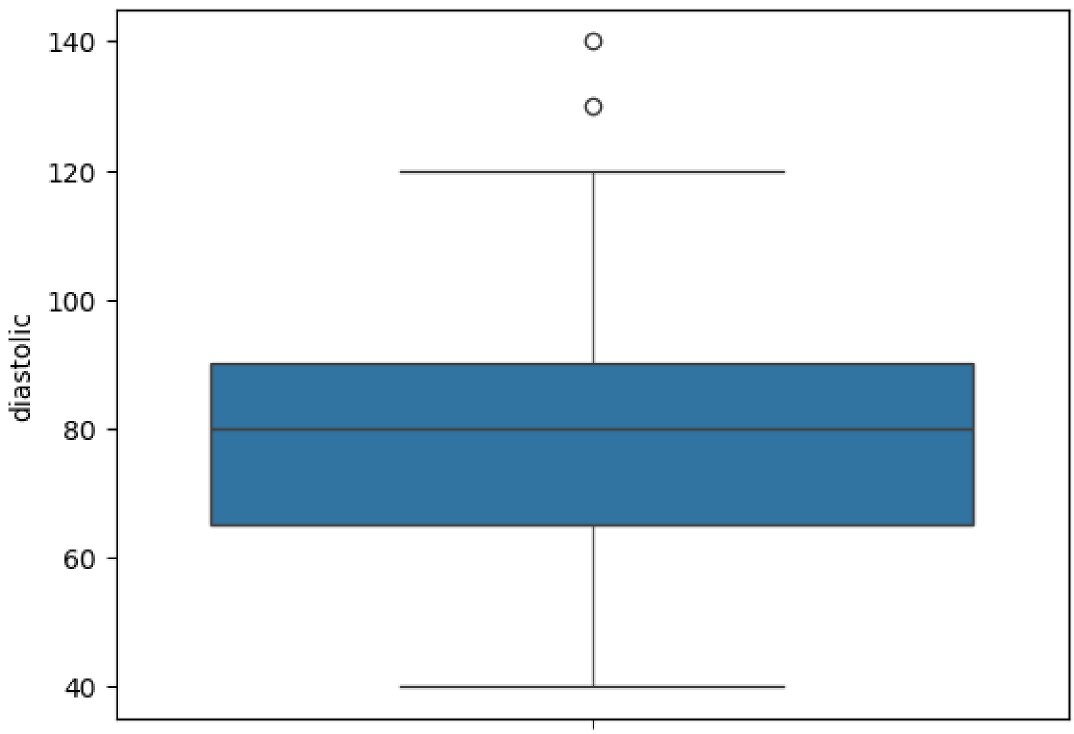
```
1                  0.0                0.0                   0
2                  1.0                0.0                   0
3                  0.0                0.0                   0
4                  0.0                0.0                   0
5                  0.0                0.0                   0
6                  0.0                1.0                   0
7                  0.0                0.0                   0
8                  0.0                0.0                   0
9                  1.0                1.0                   1

   mental_health  heart_rate  risk_level
0              1        80.0           1
1              0        74.0           0
2              0        72.0           0
3              0        74.0           0
4              0        74.0           0
5              0        76.0           0
6              0        78.0           1
7              0        74.0           0
8              0        72.0           0
9              1        80.0           1
```

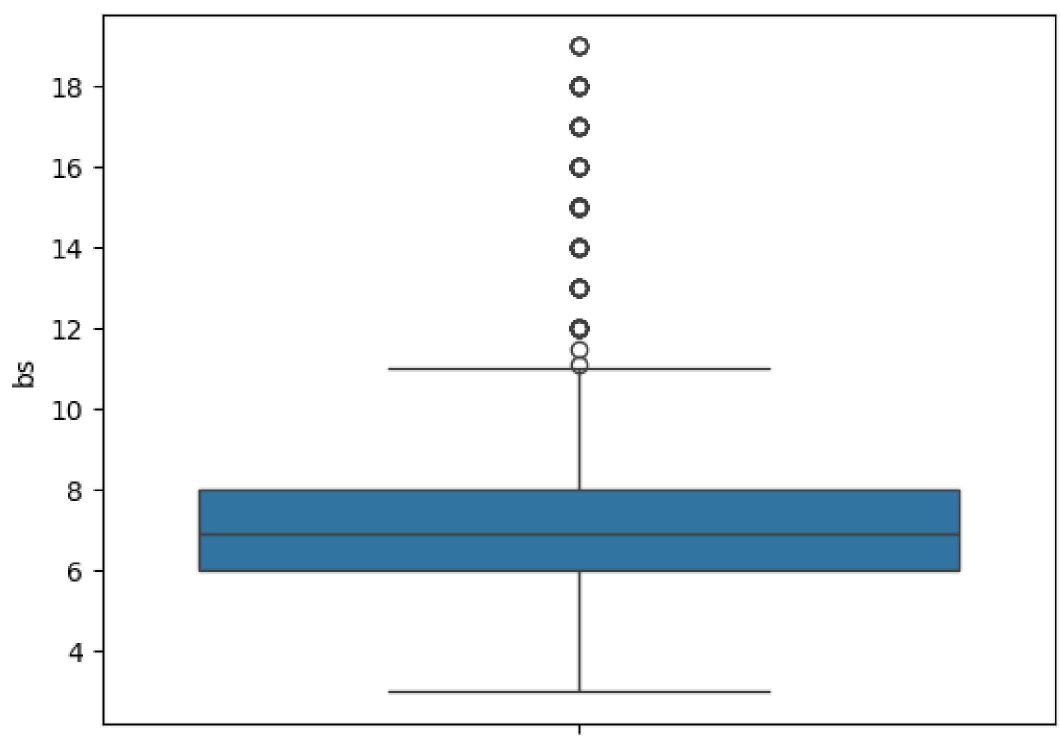
[36]:

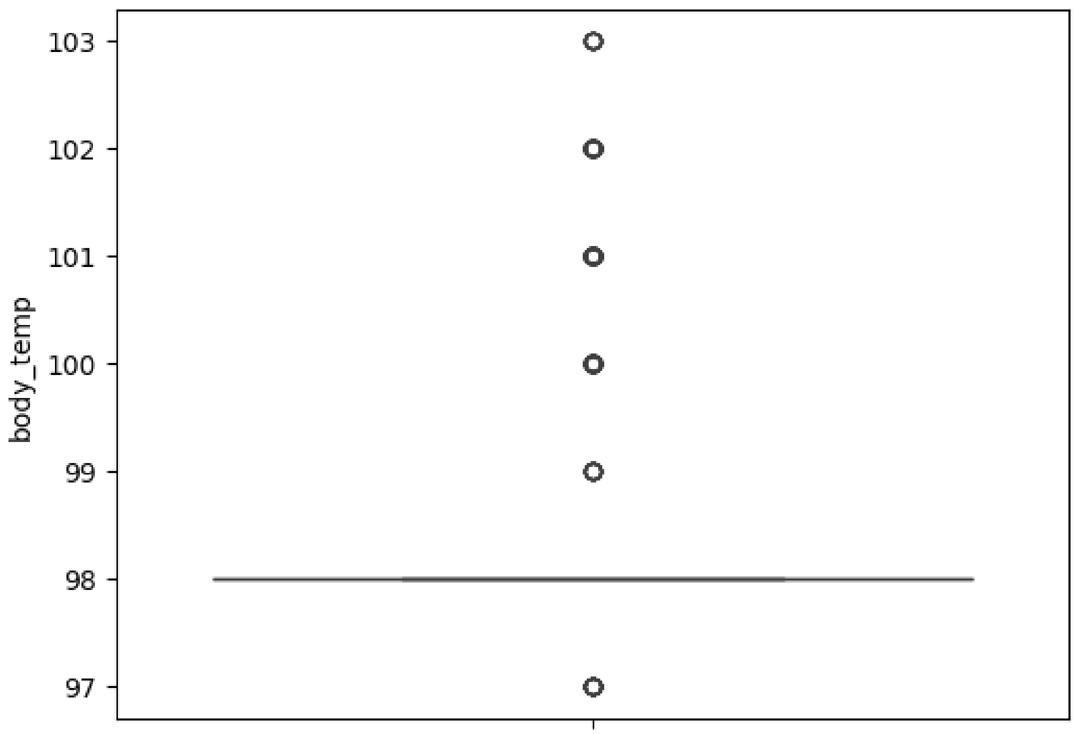
```
#Vertical Boxplot of all columns
for i in df_cp.columns:
    plt.figure()
    sns.boxplot(y=i, data = df_cp)
```

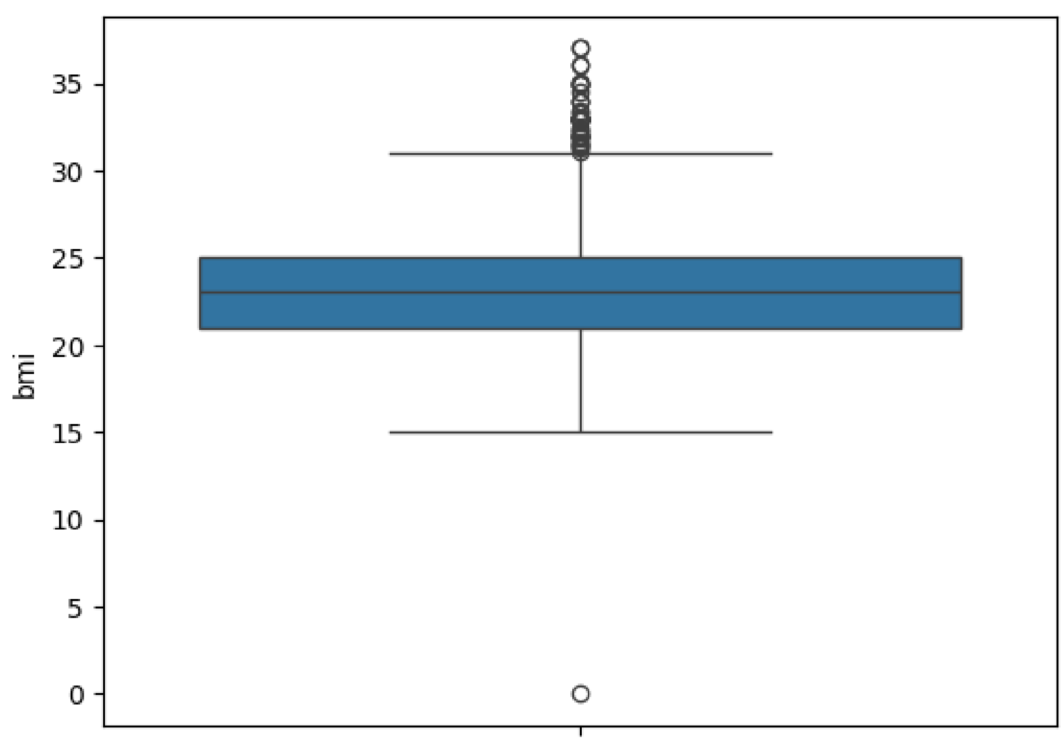

age
0
50
100
150
200
250
300

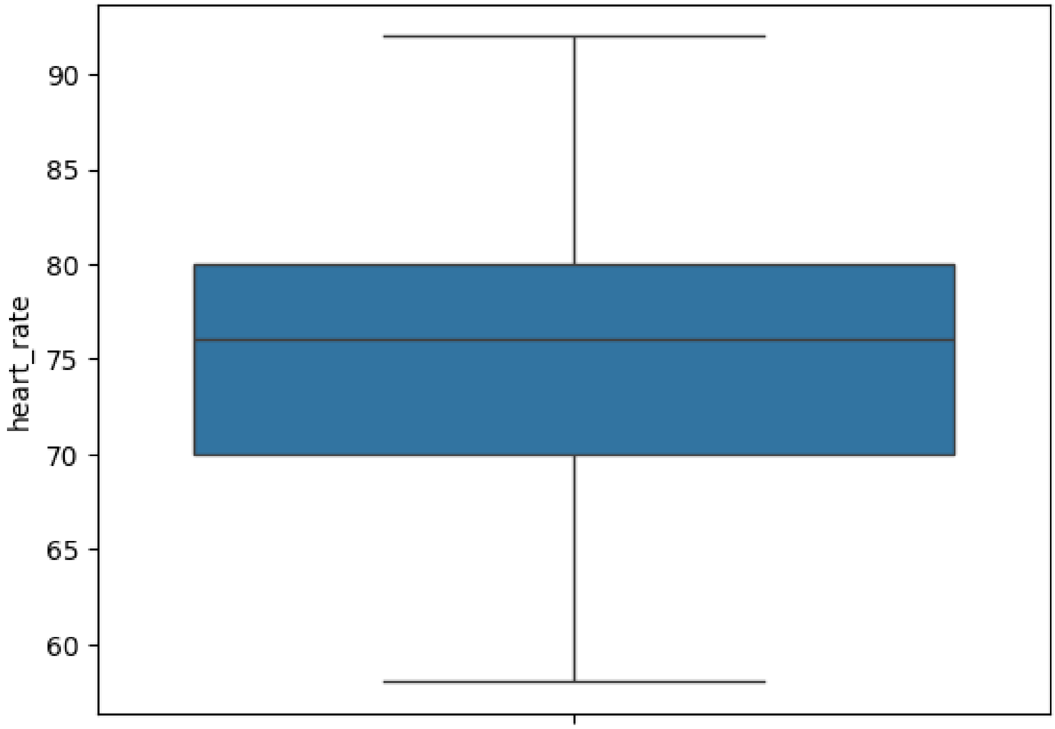

systolic_bp
80
100
120
140
160
180
200


diastolic
140
120
100
80
60
40

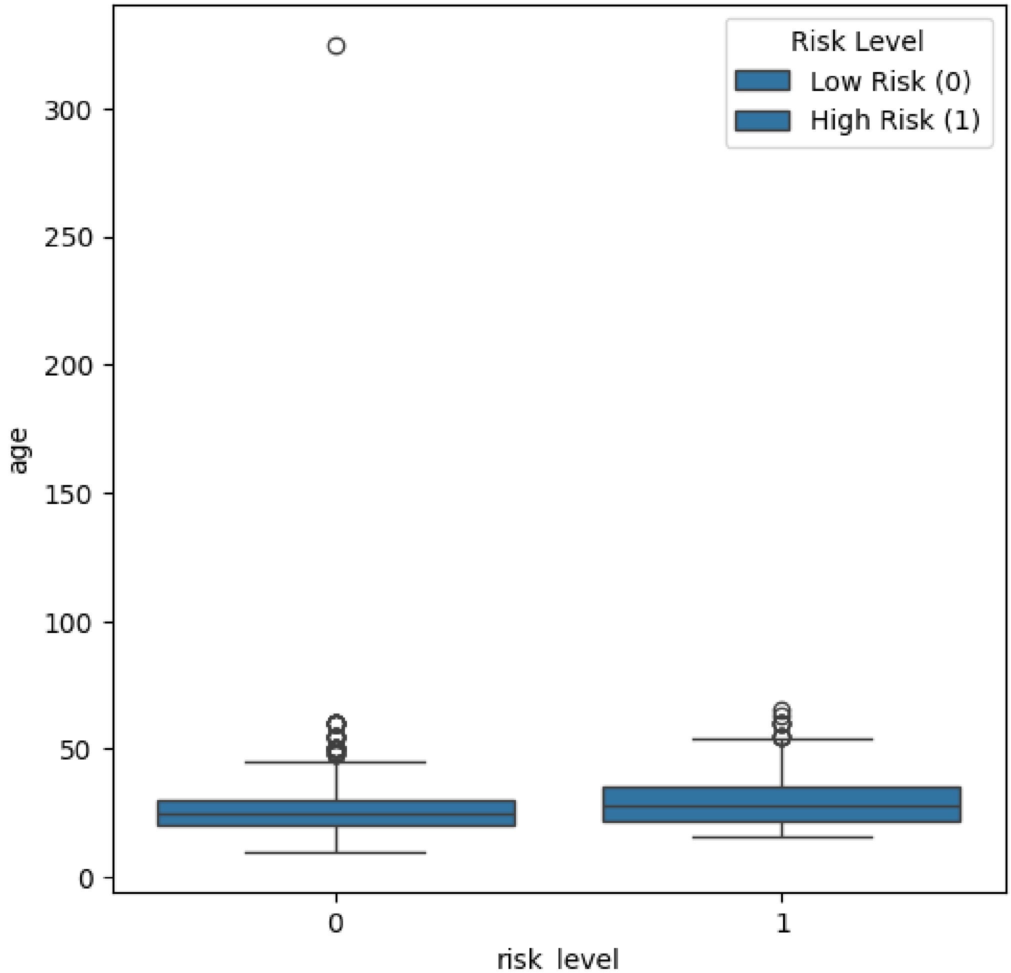

bs
18
16
14
12
10
8
6
4


103
102
101
100
99
98
97
body_temp

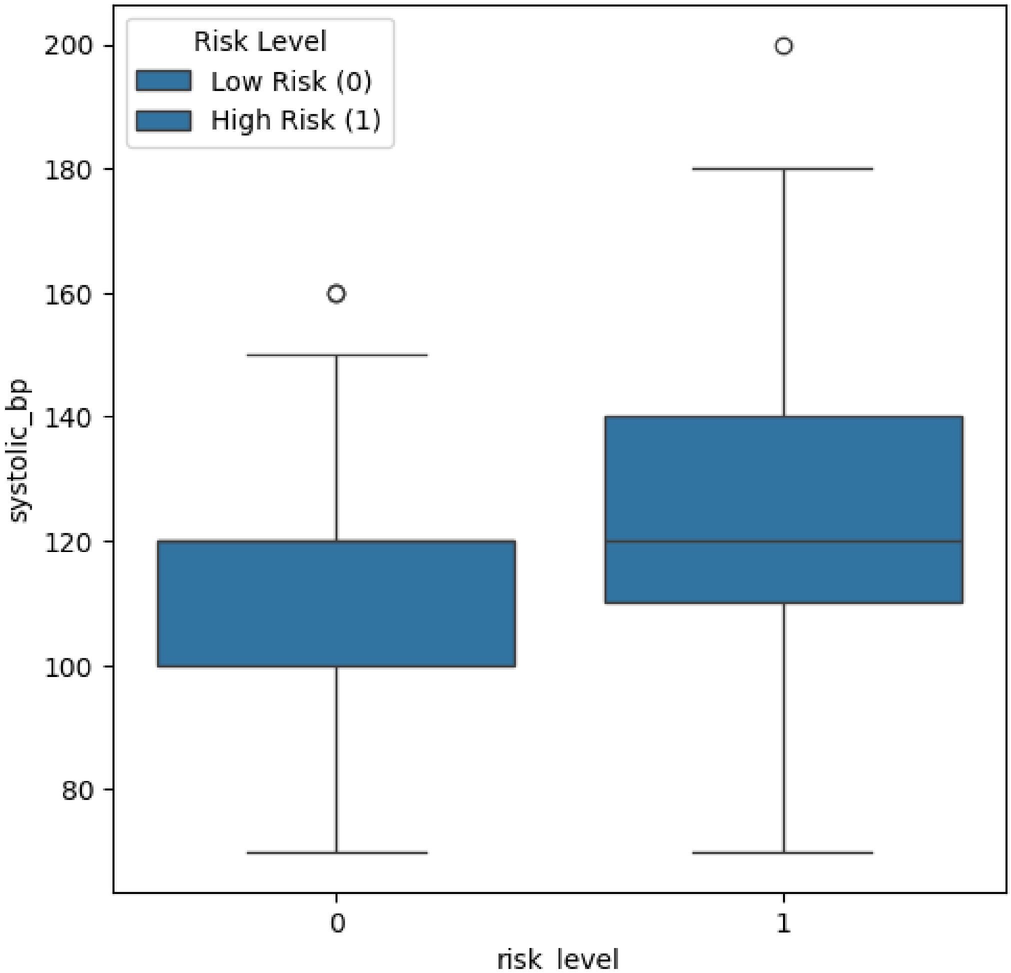

35
30
25
20
15
10
5
0
bmi



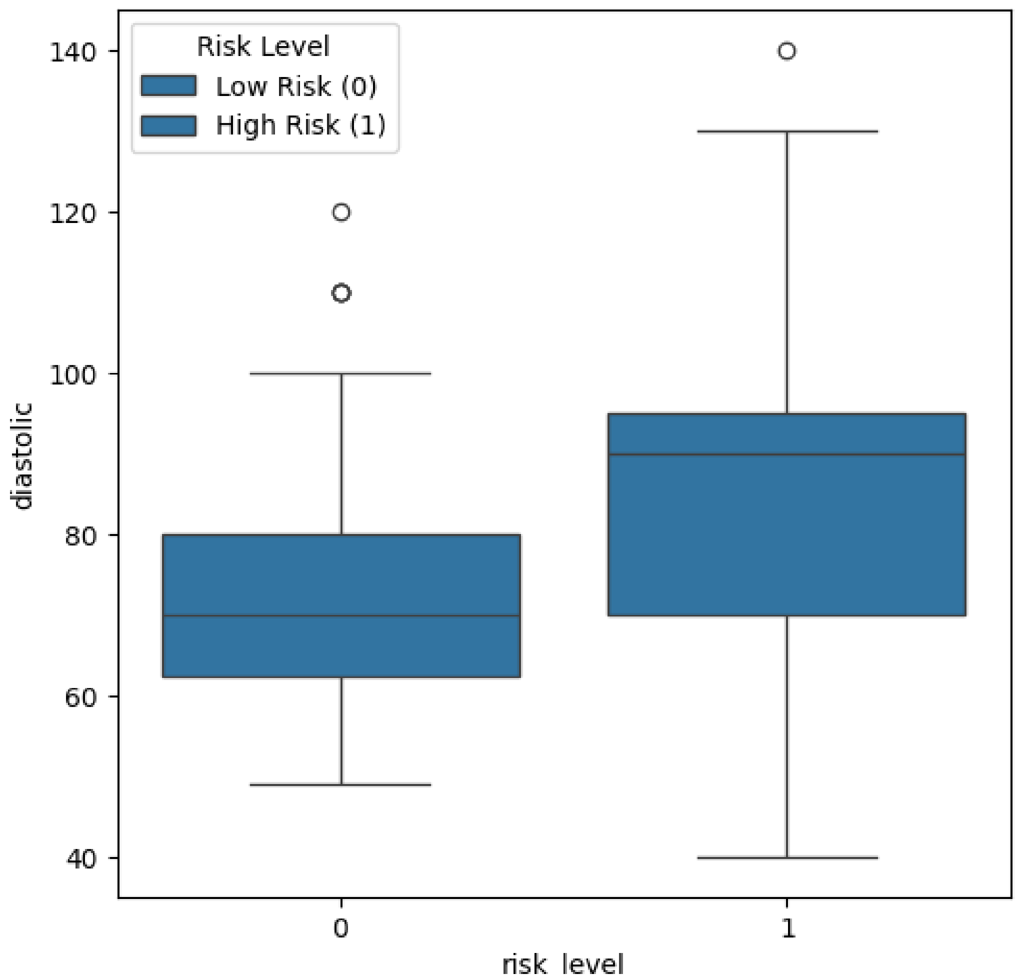

```
[37]: exclude_cols =␣
      ↪['previous_complications','preexisting_diabetes','gestational_diabetes','mental_health','ri
      for i in df.columns:
          if i not in exclude_cols:
              plt.figure(figsize=(6,6))
              sns.boxplot(y=i, x = 'risk_level', data = df)
              plt.legend(['Low Risk (0)', 'High Risk (1)'], title="Risk Level")
```

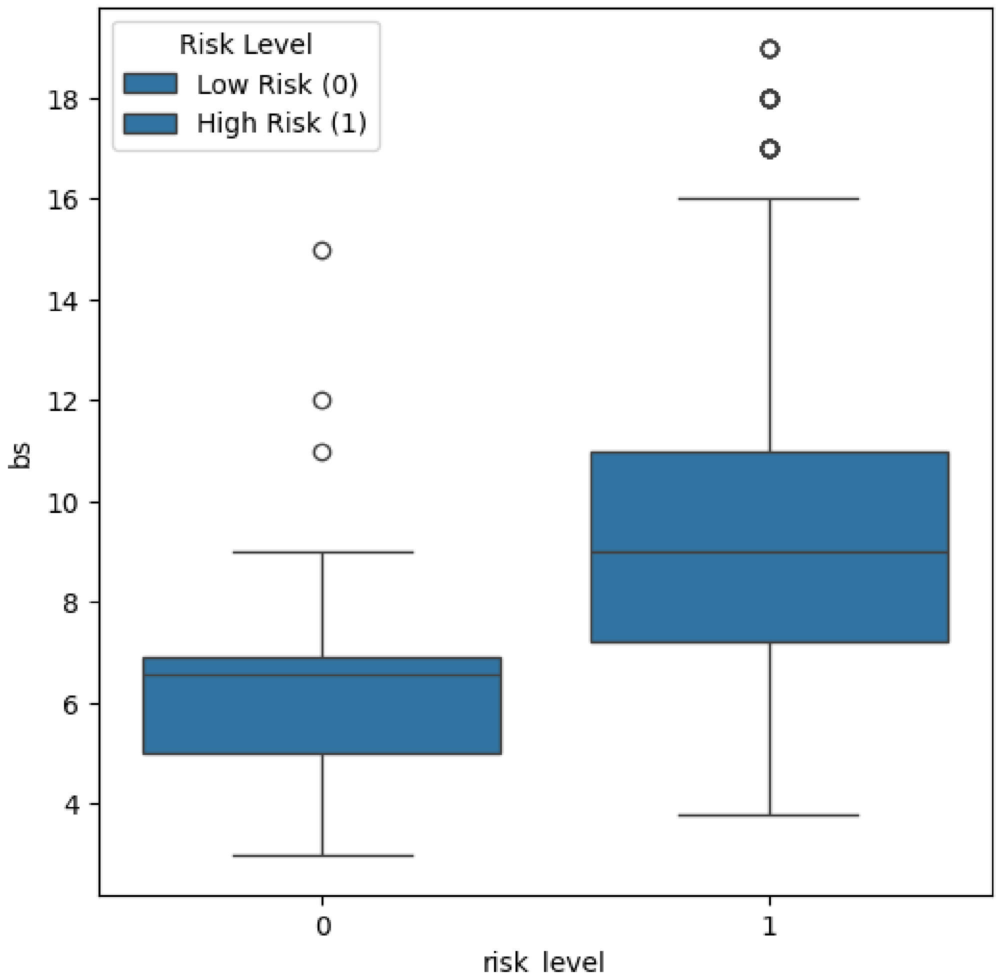


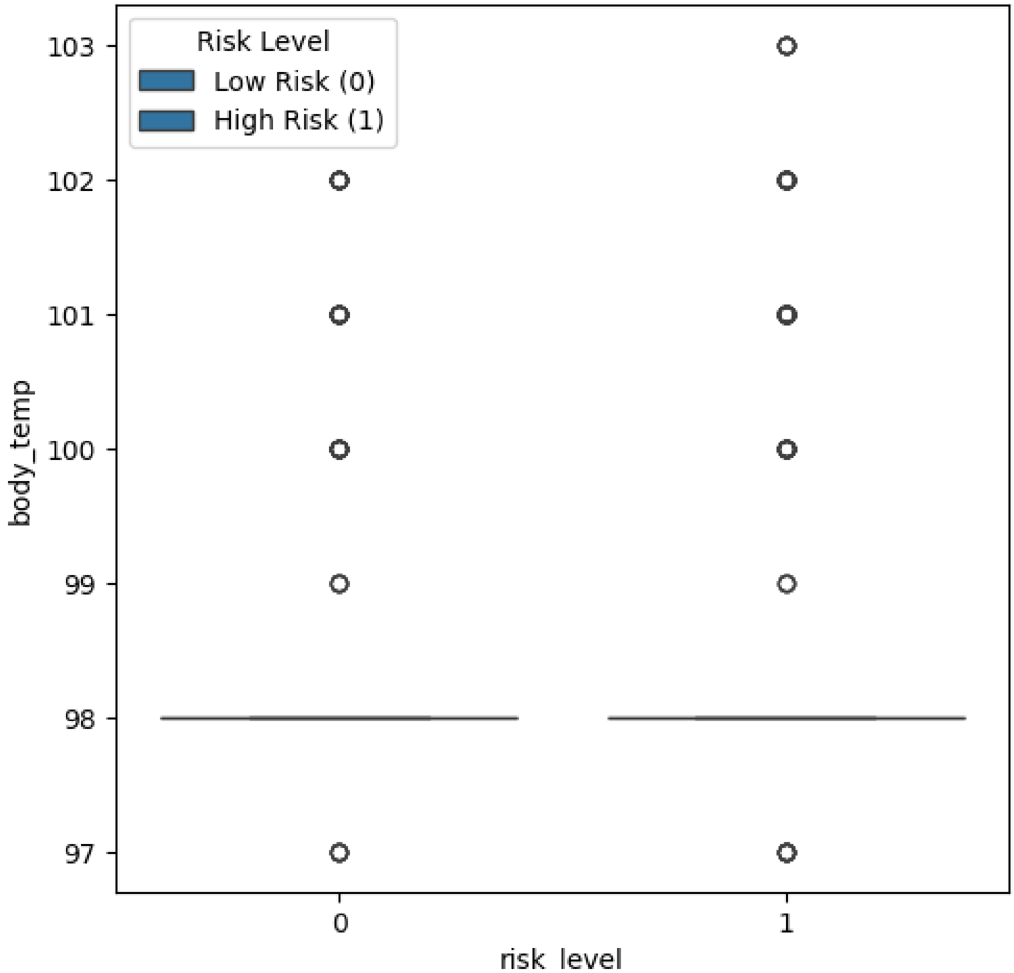
Risk Level
Low Risk (0)
High Risk (1)
300
250
200
150
100
50
0
age
0
1
risk_level

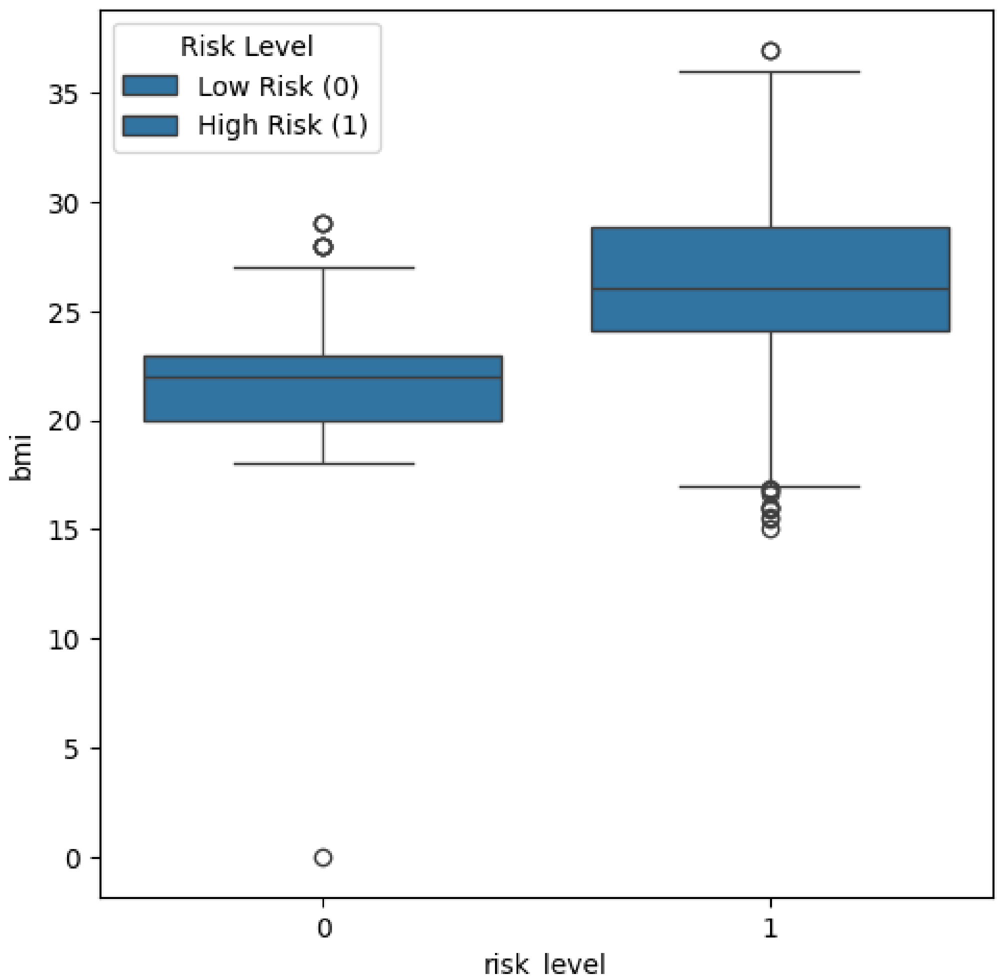


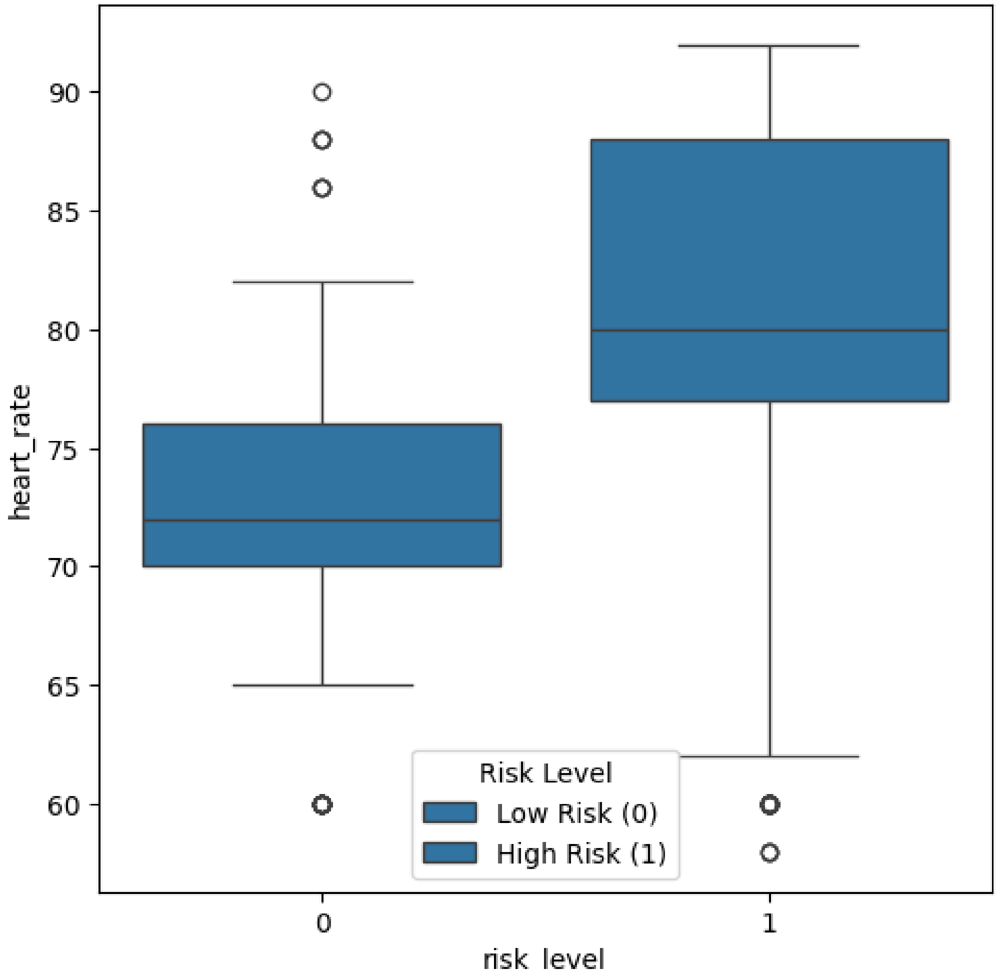
Risk Level
Low Risk (0)
High Risk (1)
200
180
160
140
120
100
80
systolic_bp
0
1
risk_level


Risk Level
Low Risk (0)
High Risk (1)
140
120
100
80
60
40
diastolic
0
1
risk_level


Risk Level
Low Risk (0)
High Risk (1)
18
16
14
12
10
8
6
4
bs
0
1
risk_level


Risk Level
Low Risk (0)
High Risk (1)
103
102
101
100
99
98
97
body_temp
0
1
risk_level


Risk Level
Low Risk (0)
High Risk (1)
35
30
25
20
15
10
5
0
bmi
0
1
risk_level



[38]:
```
# Sample: identify non-binary numeric columns
non_binary_cols = [
    col for col in df.select_dtypes(include=['number']).columns
    if df[col].nunique() > 2
]

# Loop through each non-binary column and calculate IQR stats
for col in non_binary_cols:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    lb = Q1 - 1.5 * IQR
    ub = Q3 + 1.5 * IQR
    print(f"\nFeature: {col}")
    print(f"  Q1: {Q1}")
    print(f"  Q3: {Q3}")
```

```python
    print(f"  IQR: {IQR}")
    print(f"  Lower Bound: {lb}")
    print(f"  Upper Bound: {ub}")
```

```
Feature: age
  Q1: 21.0
  Q3: 32.0
  IQR: 11.0
  Lower Bound: 4.5
  Upper Bound: 48.5

Feature: systolic_bp
  Q1: 100.0
  Q3: 130.0
  IQR: 30.0
  Lower Bound: 55.0
  Upper Bound: 175.0

Feature: diastolic
  Q1: 65.0
  Q3: 90.0
  IQR: 25.0
  Lower Bound: 27.5
  Upper Bound: 127.5

Feature: bs
  Q1: 6.0
  Q3: 8.0
  IQR: 2.0
  Lower Bound: 3.0
  Upper Bound: 11.0

Feature: body_temp
  Q1: 98.0
  Q3: 98.0
  IQR: 0.0
  Lower Bound: 98.0
  Upper Bound: 98.0

Feature: bmi
  Q1: 21.0
  Q3: 25.0
  IQR: 4.0
  Lower Bound: 15.0
  Upper Bound: 31.0
```

```
Feature: previous_complications
  Q1: 0.0
  Q3: 0.0
  IQR: 0.0
  Lower Bound: 0.0
  Upper Bound: 0.0

Feature: preexisting_diabetes
  Q1: 0.0
  Q3: 1.0
  IQR: 1.0
  Lower Bound: -1.5
  Upper Bound: 2.5

Feature: heart_rate
  Q1: 70.0
  Q3: 80.0
  IQR: 10.0
  Lower Bound: 55.0
  Upper Bound: 95.0
```

[39]:

```python
# Calculate the IQR
Q1 = df['age'].quantile(0.25)  # First quartile (25th percentile)
Q3 = df['age'].quantile(0.75)  # Third quartile (75th percentile)
IQR = Q3 - Q1                  # Interquartile range

# Define lower and upper bounds for outliers
lb_age = Q1 - 1.5 * IQR
ub_age = Q3 + 1.5 * IQR

print(lb_age)
print(ub_age)
```

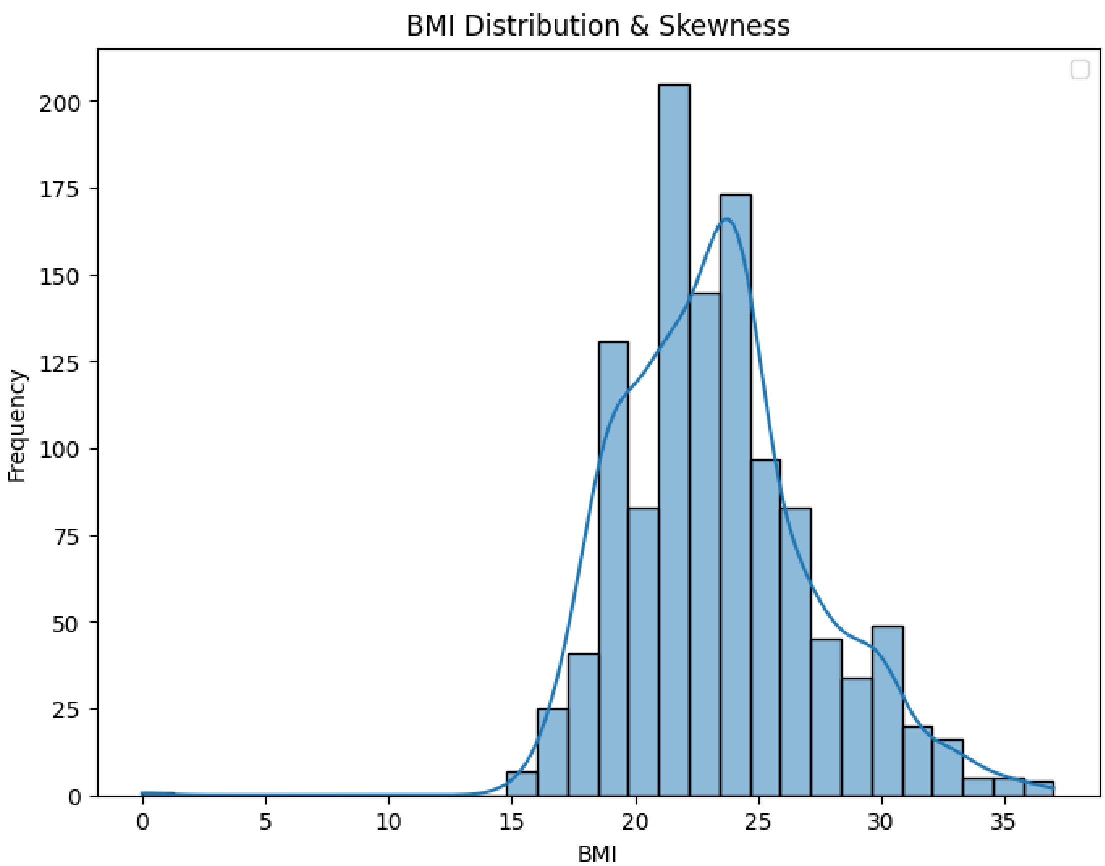
```
4.5
48.5
```

[40]:

```python
# Calculate the IQR
Q1 = df['bmi'].quantile(0.25)  # First quartile (25th percentile)
Q3 = df['bmi'].quantile(0.75)  # Third quartile (75th percentile)
IQR = Q3 - Q1                  # Interquartile range

# Define lower and upper bounds for outliers
lb_bmi = Q1 - 1.5 * IQR
ub_bmi = Q3 + 1.5 * IQR

print(lb_bmi)
print(ub_bmi)
```

```
15.0
31.0
```

[41]:
```python
# Plot histogram with KDE
plt.figure(figsize=(8,6))
sns.histplot(df['bmi'], bins=30, kde=True)  # kde=True adds the density curve
plt.legend()
plt.title("BMI Distribution & Skewness")
plt.xlabel("BMI")
plt.ylabel("Frequency")
plt.show()
```



[42]:
```python
df = df[df['bmi'] != 0]
```

4 Remove the Outliers in BMI and Age column

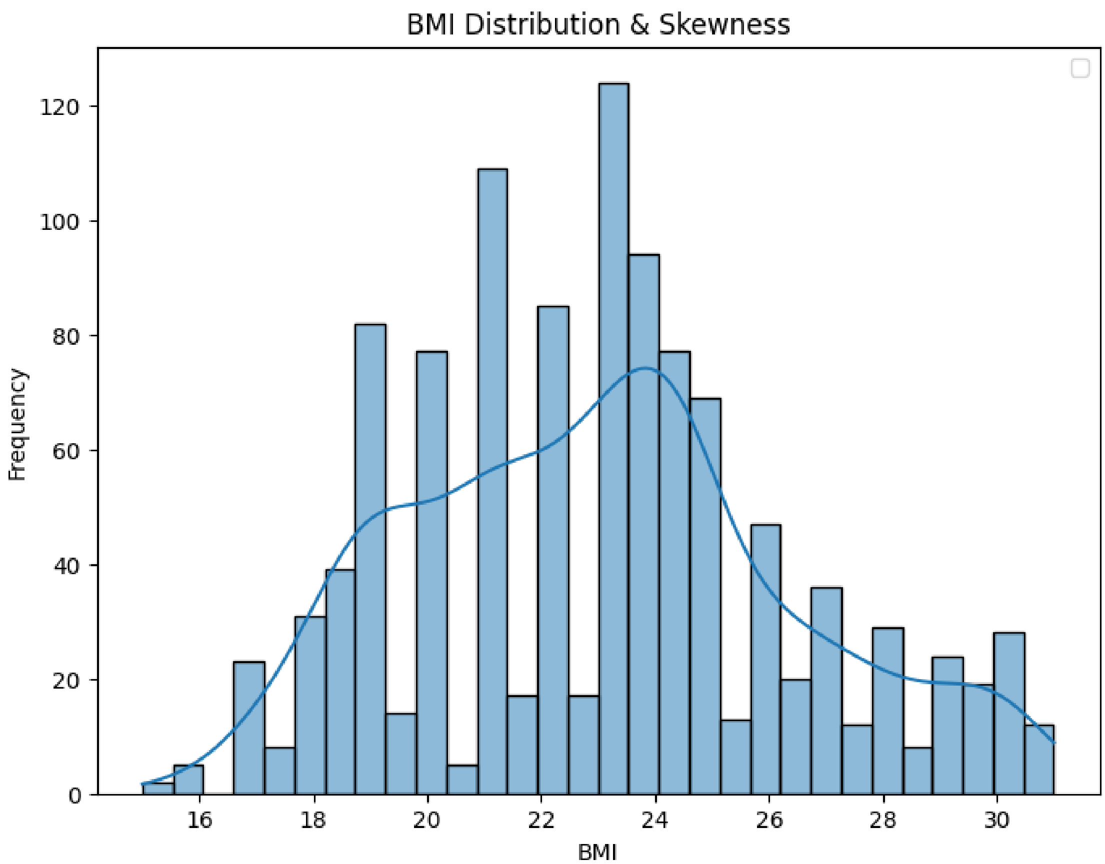
[43]:
```
# Filter the dataset to remove outliers in BMI column
df_cln = df[(df['bmi'] >= lb_bmi) & (df['bmi'] <= ub_bmi)]

# Updated dataset size
print(df_cln.shape)
```

```
(1126, 12)
```

[44]:
```
df_cln.head()
```

[44]:
```
   age  systolic_bp  diastolic   bs  body_temp   bmi  previous_complications  \
0   22         90.0       60.0  9.0        100  18.0                     1.0
1   22        110.0       70.0  7.1         98  20.4                     0.0
2   27        110.0       70.0  7.5         98  23.0                     1.0
3   20        100.0       70.0  7.2         98  21.2                     0.0
4   20         90.0       60.0  7.5         98  19.7                     0.0

   preexisting_diabetes  gestational_diabetes  mental_health  heart_rate  \
0                   1.0                     0              1        80.0
1                   0.0                     0              0        74.0
2                   0.0                     0              0        72.0
3                   0.0                     0              0        74.0
4                   0.0                     0              0        74.0

   risk_level
0           1
1           0
2           0
3           0
4           0
```

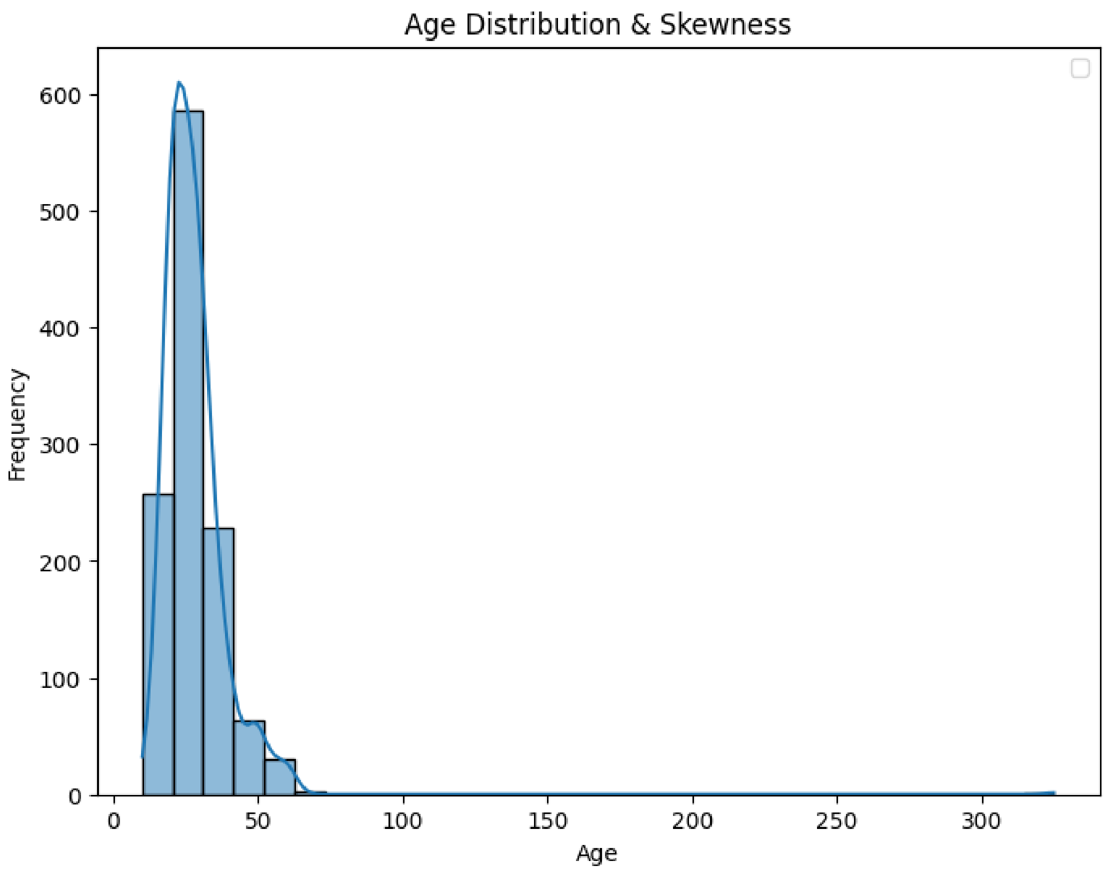
[45]:
```
# Plot histogram with KDE
plt.figure(figsize=(8,6))
sns.histplot(df_cln['bmi'], bins=30, kde=True)  # kde=True adds the density␣
 ↪curve
plt.legend()
plt.title("BMI Distribution & Skewness")
plt.xlabel("BMI")
plt.ylabel("Frequency")
plt.show()
print(df_cln['age'].describe())
```



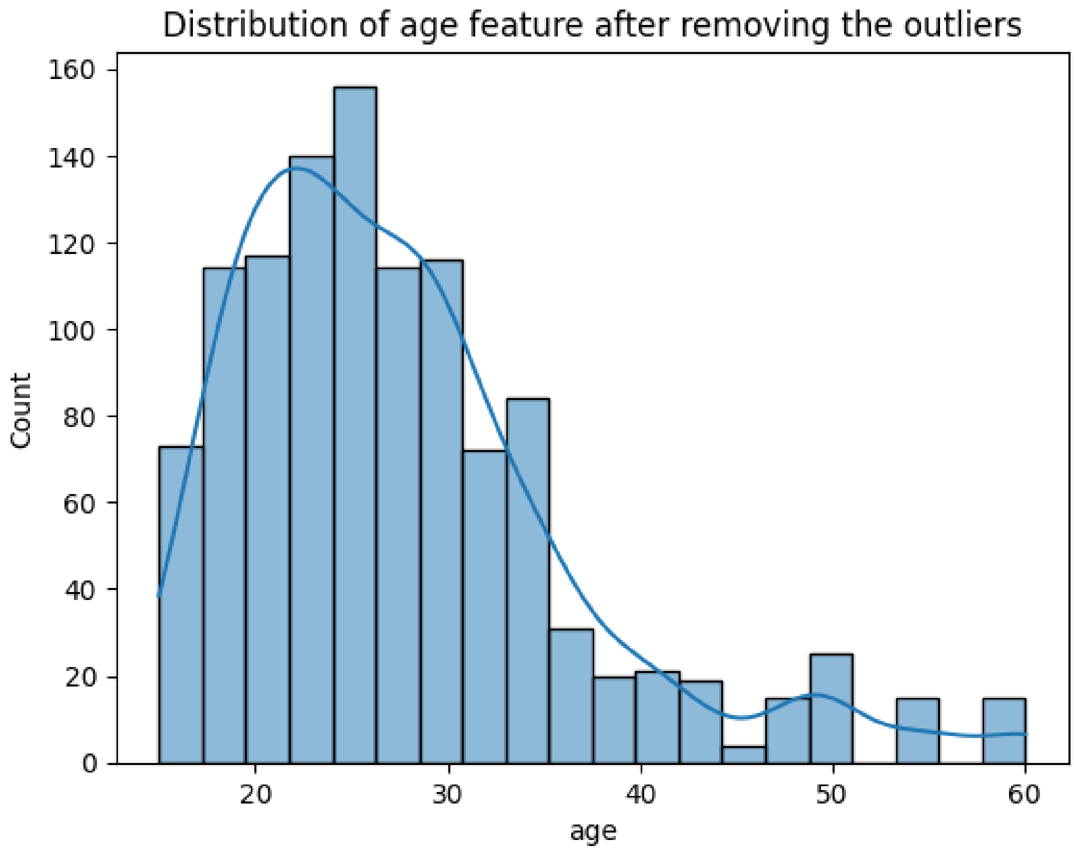

```
count    1126.000000
mean       27.658970
std        12.791839
min        10.000000
25%        21.000000
50%        25.000000
75%        31.000000
max       325.000000
Name: age, dtype: float64
```

[46]:

```
# Plot histogram with KDE
plt.figure(figsize=(8,6))
sns.histplot(df['age'], bins=30, kde=True)  # kde=True adds the density curve
plt.legend()
plt.title("Age Distribution & Skewness")
plt.xlabel("Age")
plt.ylabel("Frequency")
plt.show()
```



[47]:
```python
# Filter according to realistic age range for pregnancy in rural Bangladesh
df_cln = df[df['age'].between(15, 60)]

sns.histplot(df_cln['age'], bins=20, kde=True)
plt.title("Distribution of age feature after removing the outliers")
plt.show()

# Check updated plot
print(df_cln['age'].describe())
```




```
count    1151.000000
mean       27.702867
std         9.027802
min        15.000000
25%        21.000000
50%        26.000000
75%        32.000000
max        60.000000
Name: age, dtype: float64
```

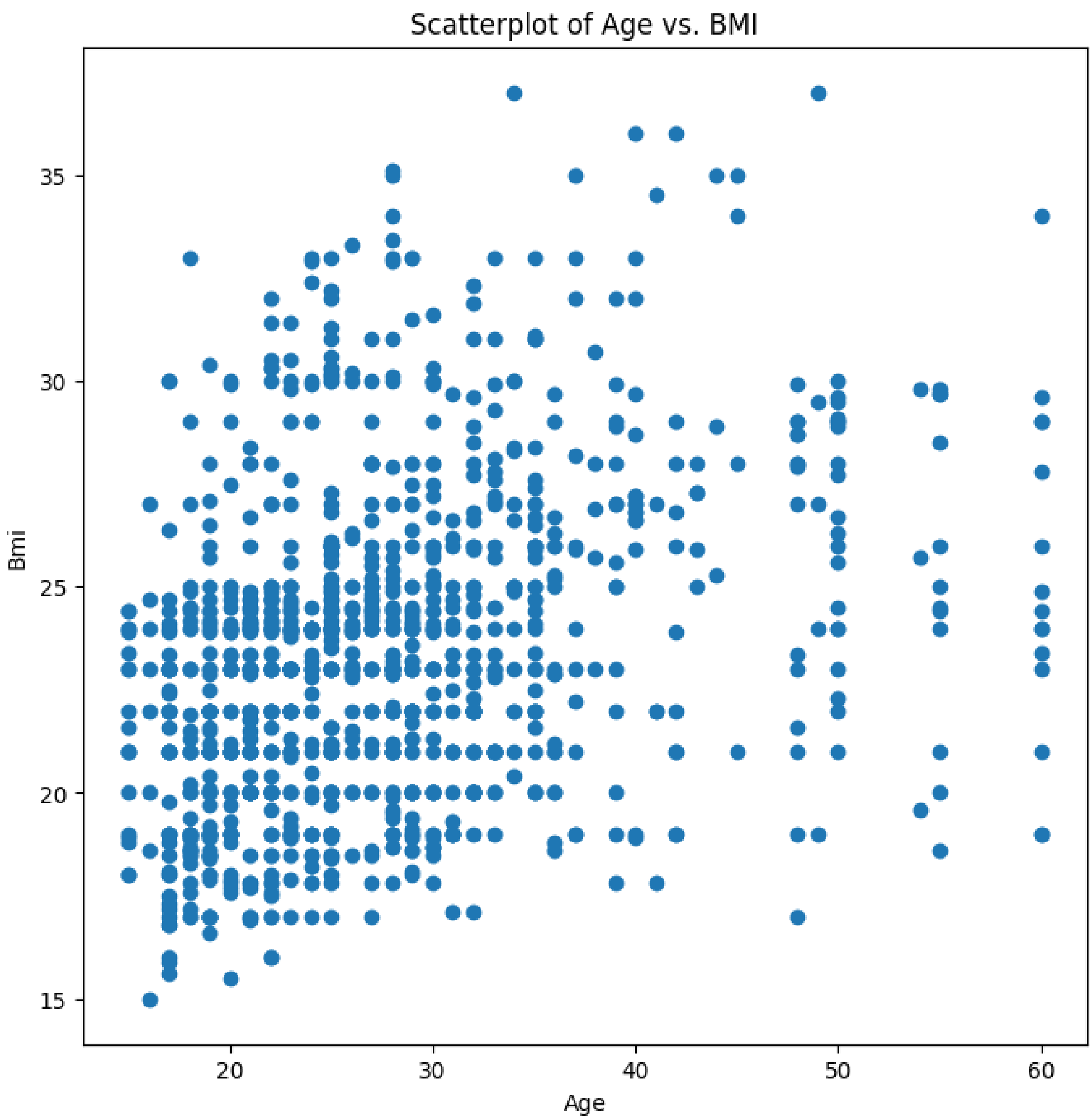
[48]:
```
df_cln.describe()
```

[48]:
```
               age  systolic_bp    diastolic           bs    body_temp  \
count  1151.000000  1151.000000  1151.000000  1151.000000  1151.000000
mean     27.702867   117.033281    77.370522     7.526849    98.388358
std       9.027802    18.736151    14.296552     3.073142     1.080018
min      15.000000    70.000000    40.000000     3.000000    97.000000
25%      21.000000   100.000000    65.000000     6.000000    98.000000
50%      26.000000   120.000000    80.000000     6.900000    98.000000
75%      32.000000   130.000000    90.000000     8.000000    98.000000
```

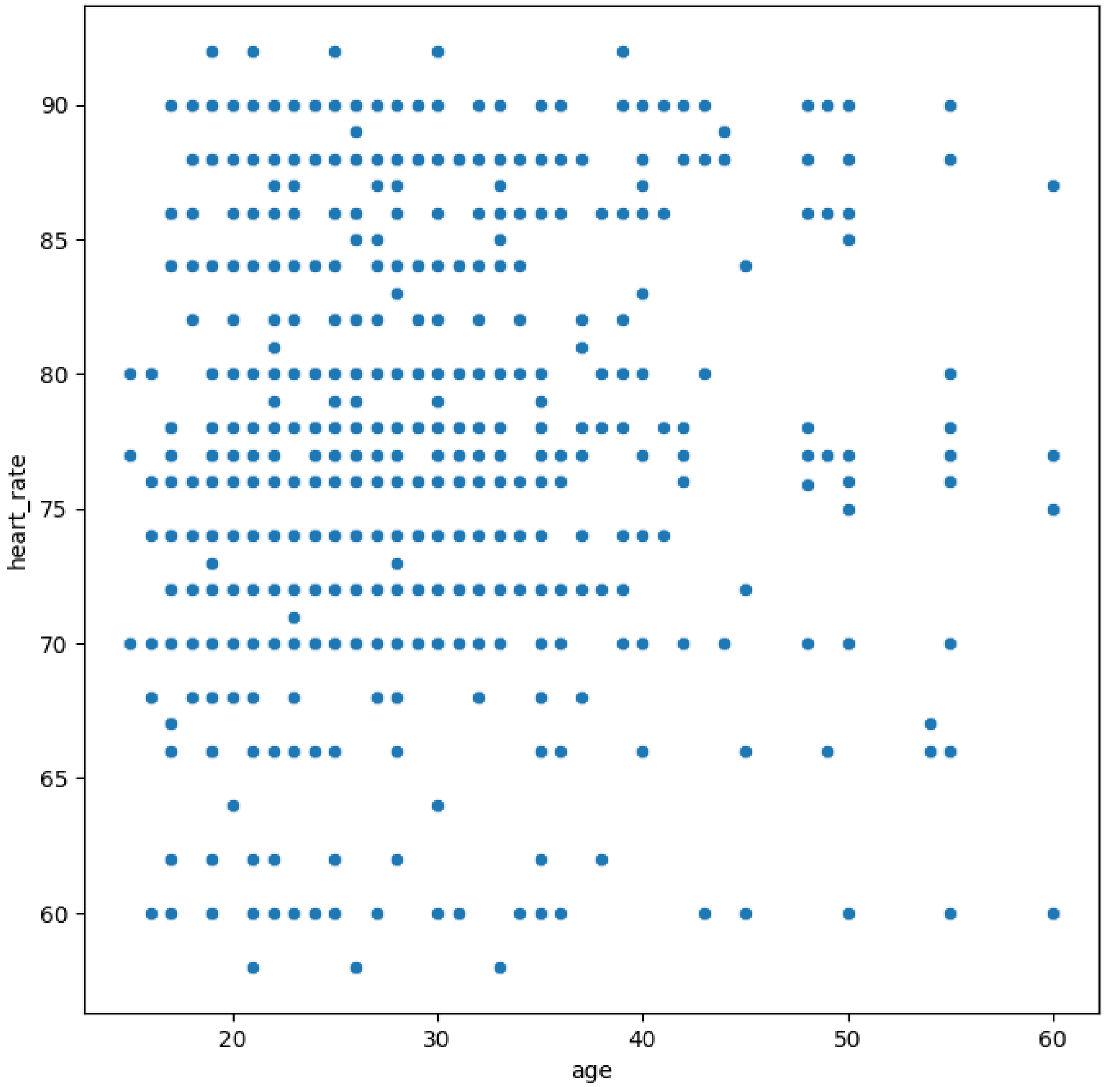
```
max      60.000000   200.000000   140.000000    19.000000   103.000000

               bmi  previous_complications  preexisting_diabetes  \
count  1151.000000             1151.000000           1151.000000
mean     23.390824                0.181737              0.294782
std       3.834108                0.385629              0.455945
min      15.000000                0.000000              0.000000
25%      21.000000                0.000000              0.000000
50%      23.000000                0.000000              0.000000
75%      25.150000                0.000000              1.000000
max      37.000000                1.000000              1.000000

       gestational_diabetes  mental_health   heart_rate   risk_level
count           1151.000000    1151.000000  1151.000000  1151.000000
mean               0.121633       0.344049    75.927788     0.409209
std                0.327004       0.475264     7.212610     0.491902
min                0.000000       0.000000    58.000000     0.000000
25%                0.000000       0.000000    70.000000     0.000000
50%                0.000000       0.000000    76.000000     0.000000
75%                0.000000       1.000000    80.000000     1.000000
max                1.000000       1.000000    92.000000     1.000000
```

5 Scatter Plot

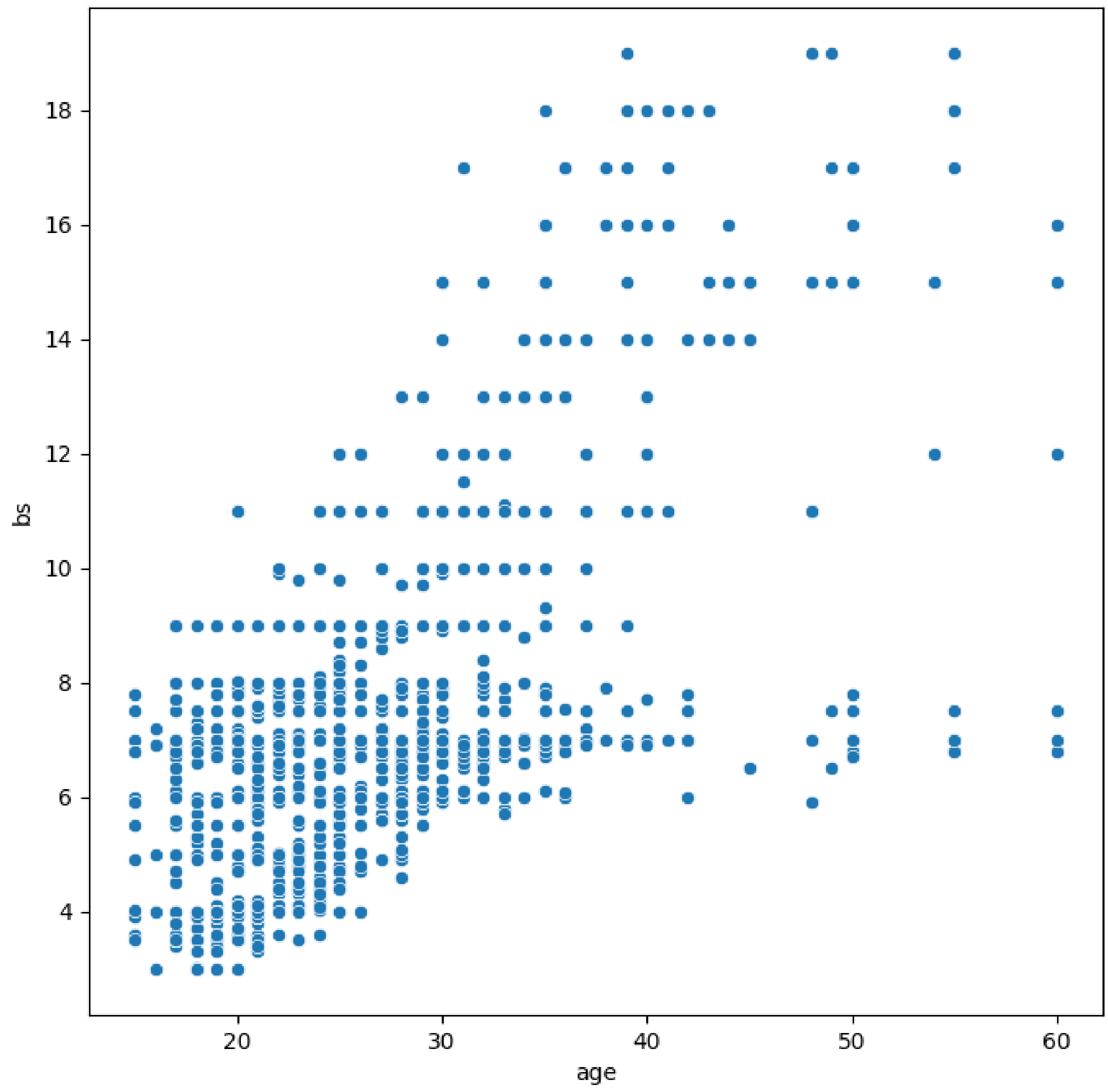
[49]:
```
#Let's see how the BMI varies according to age in pregnancy
plt.figure(figsize = (8,8))
plt.scatter(x = 'age', y = 'bmi', data = df_cln)
# scatter plot with pyplot
plt.xlabel('Age')
plt.ylabel('Bmi')
plt.title('Scatterplot of Age vs. BMI');
```



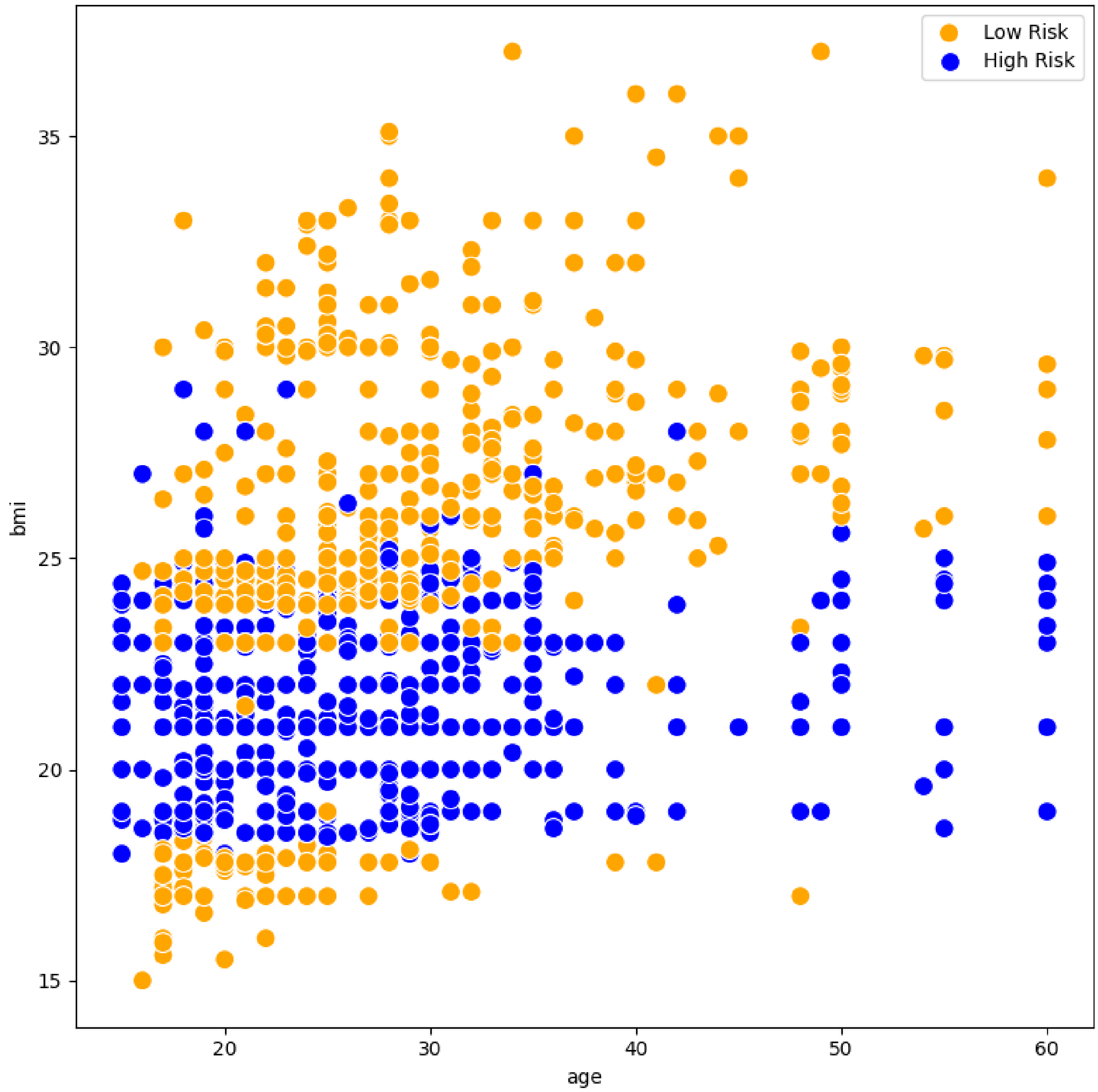
[50]:
```
plt.figure(figsize = (8,8))
sns.scatterplot(x = 'age', y = 'heart_rate', data = df_cln);
# scatter plot with Seaborn
```



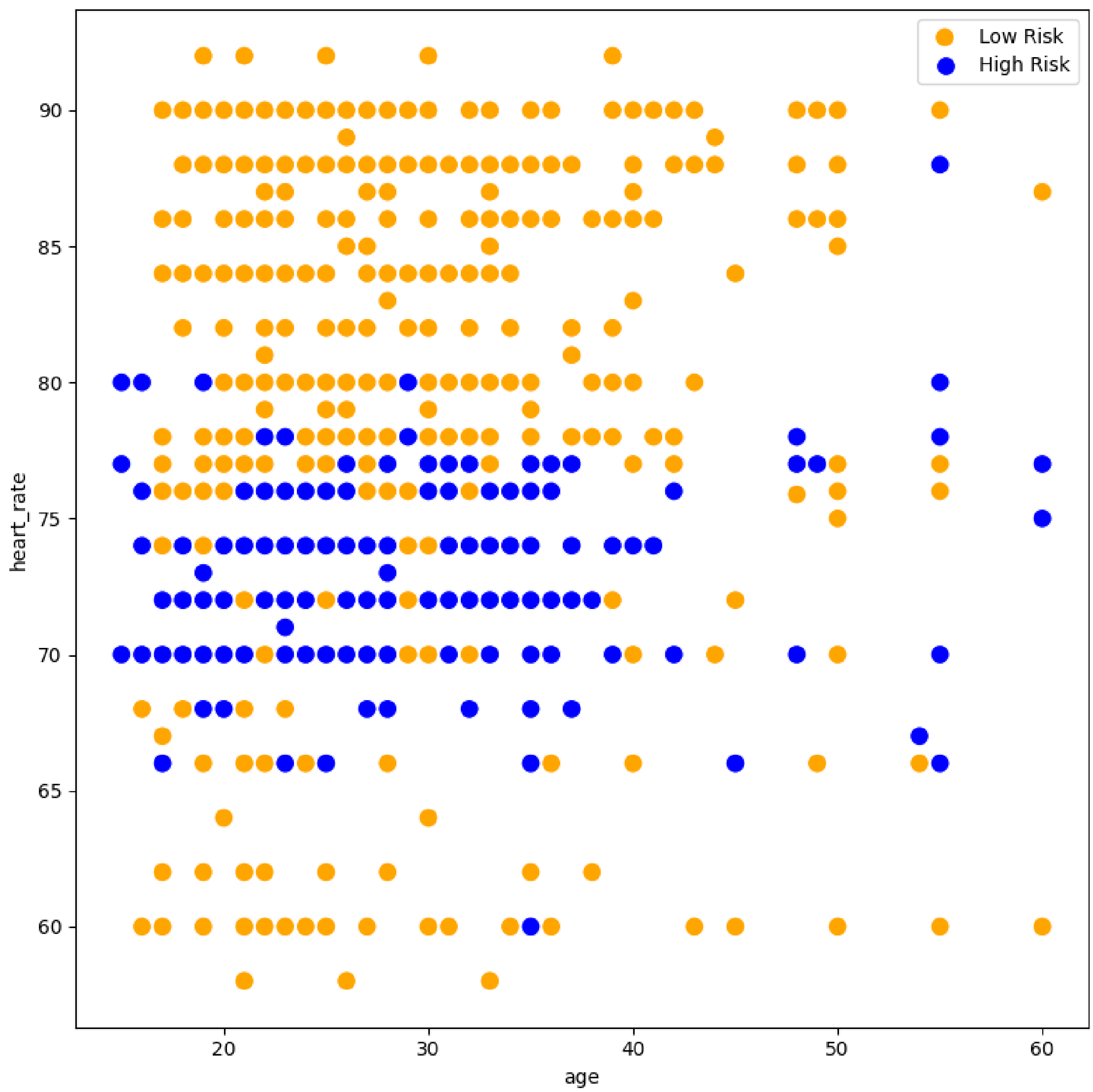
[51]:
```
plt.figure(figsize = (8,8))
sns.scatterplot(x = 'age', y = 'bs', data = df_cln);
# scatter plot with Seaborn
```



[52]:
```
plt.figure(figsize = (8,8))
sns.scatterplot(x = 'age', y = 'bmi', hue='risk_level', palette=['blue',␣
↪'orange'], s=100, data = df_cln);
plt.legend(labels=['Low Risk', 'High Risk'], loc='upper right')
plt.tight_layout()
# scatter plot with Seaborn
```

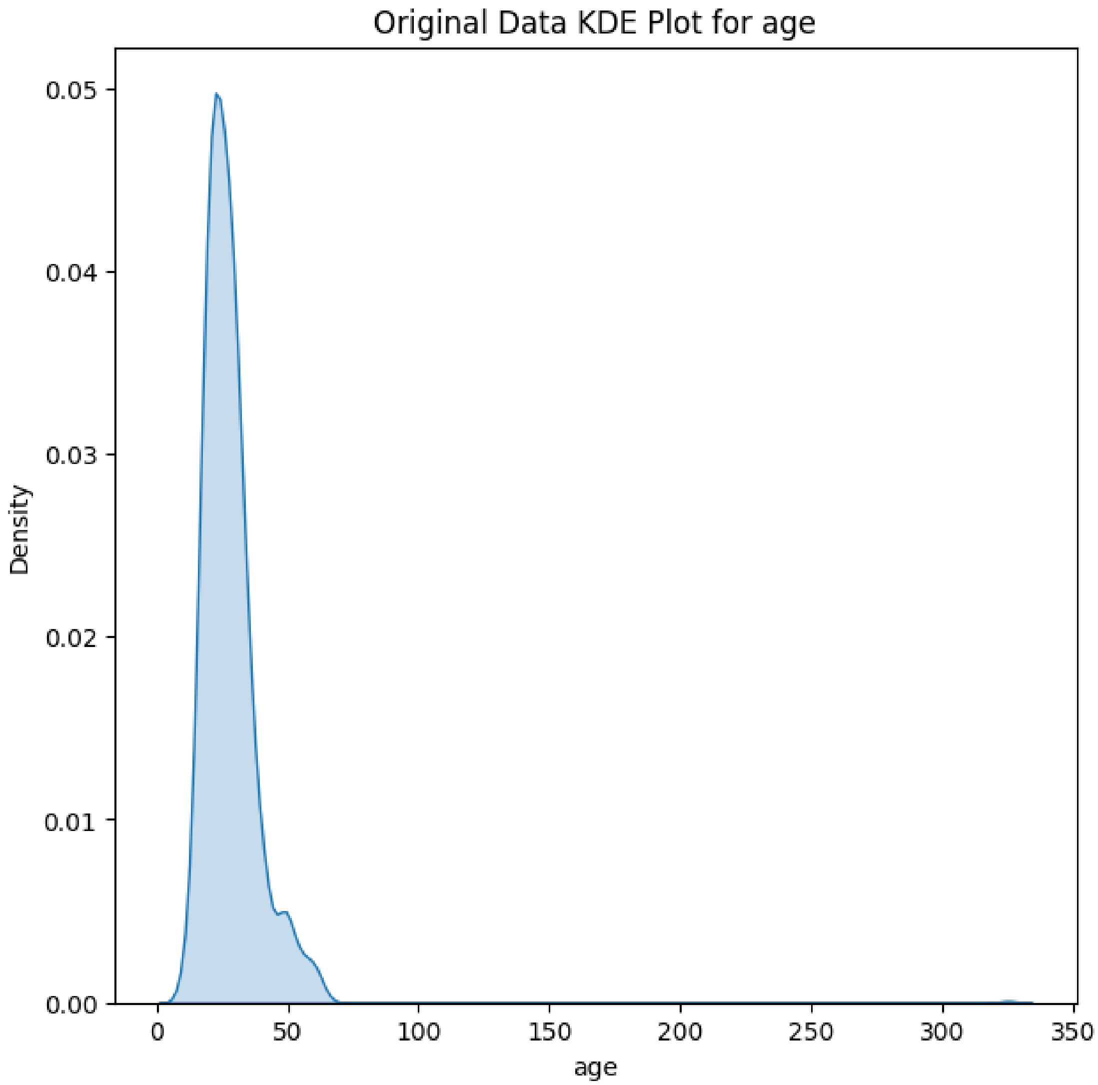


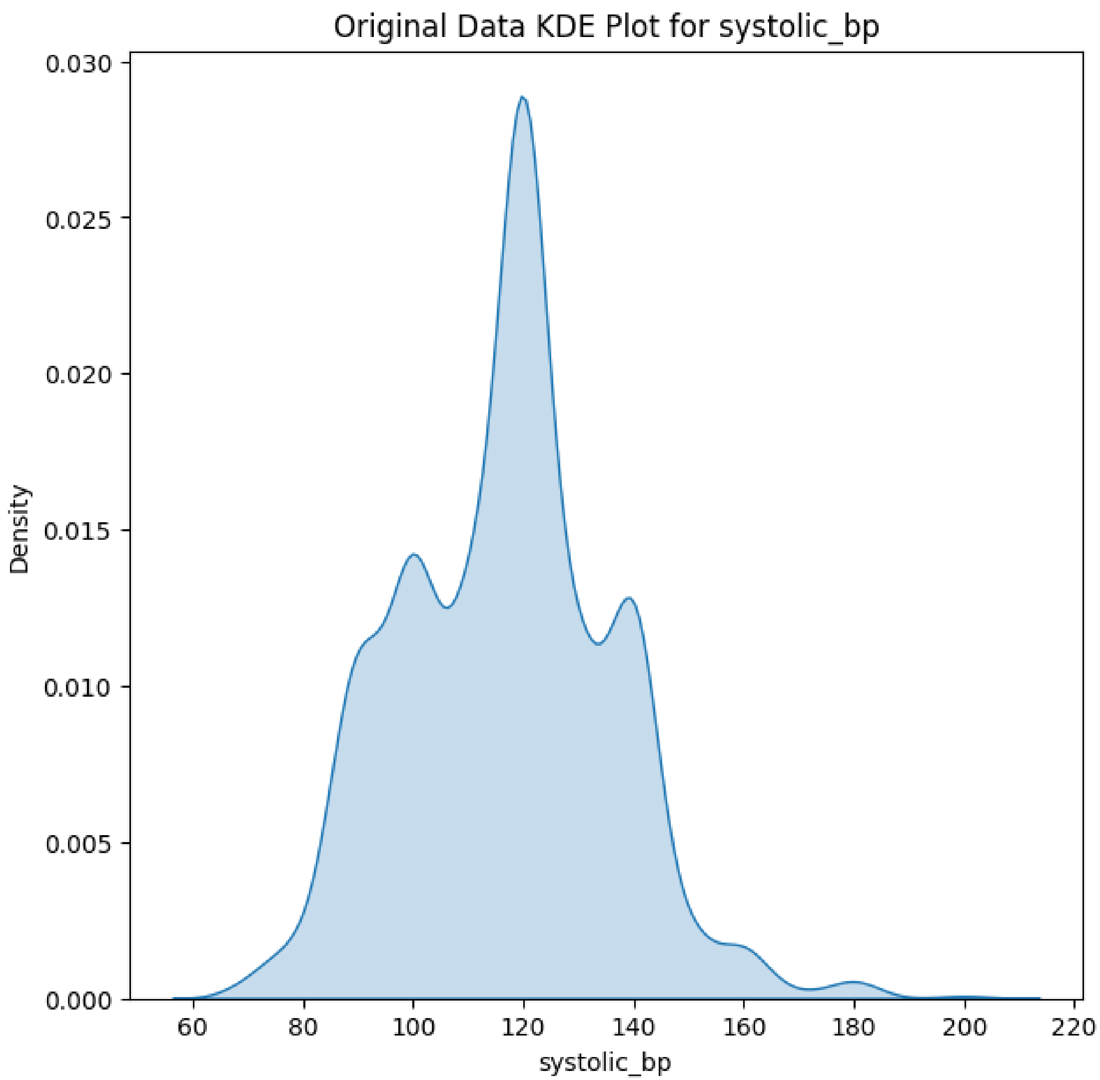
[53]:
```
plt.figure(figsize = (8,8))
sns.scatterplot(x = 'age', y = 'heart_rate', hue='risk_level', palette=['blue',
    'orange'], s=100, data = df_cln);
plt.legend(labels=['Low Risk', 'High Risk'], loc='upper right')
plt.tight_layout()
# scatter plot with Seaborn
```

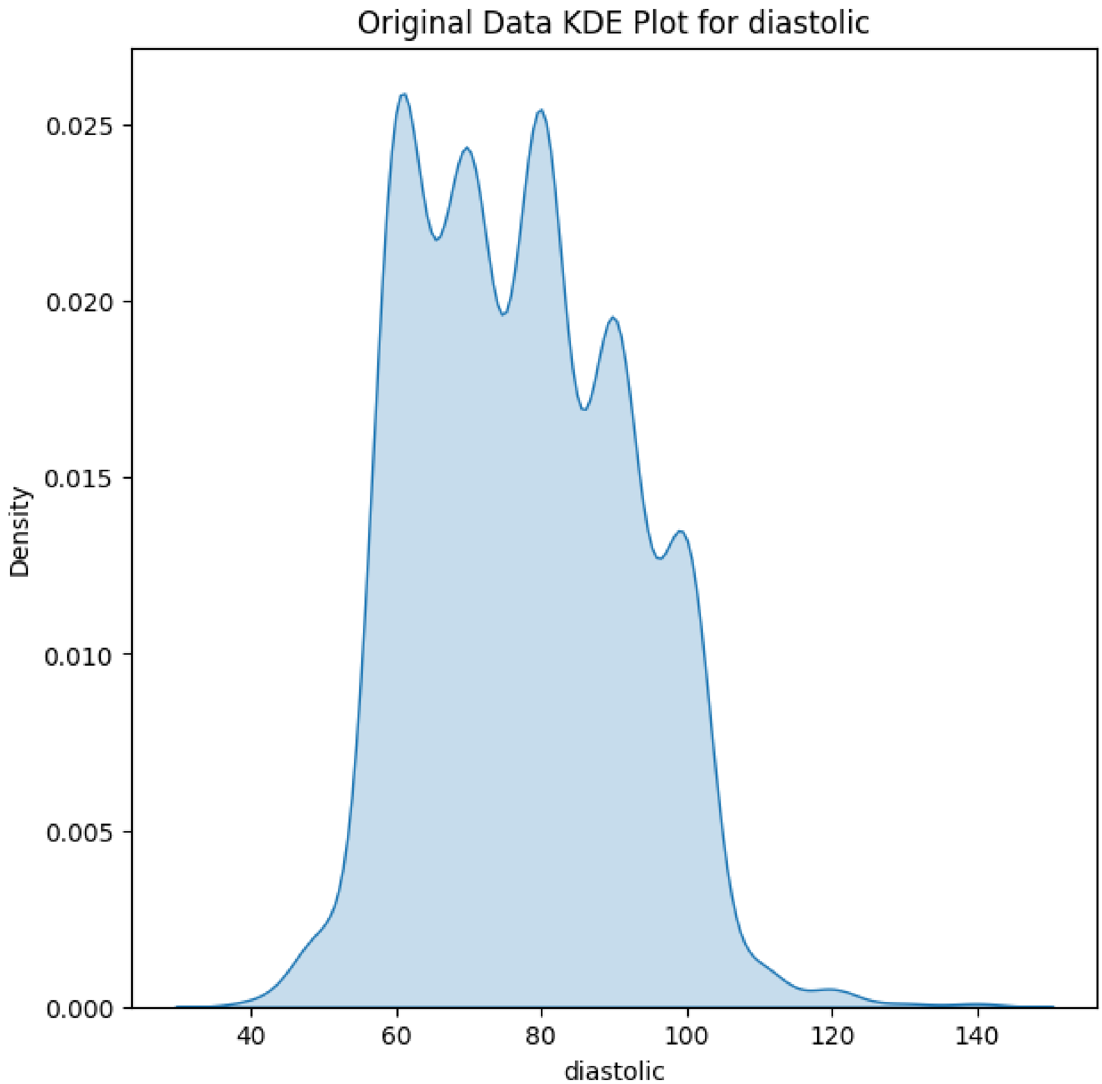


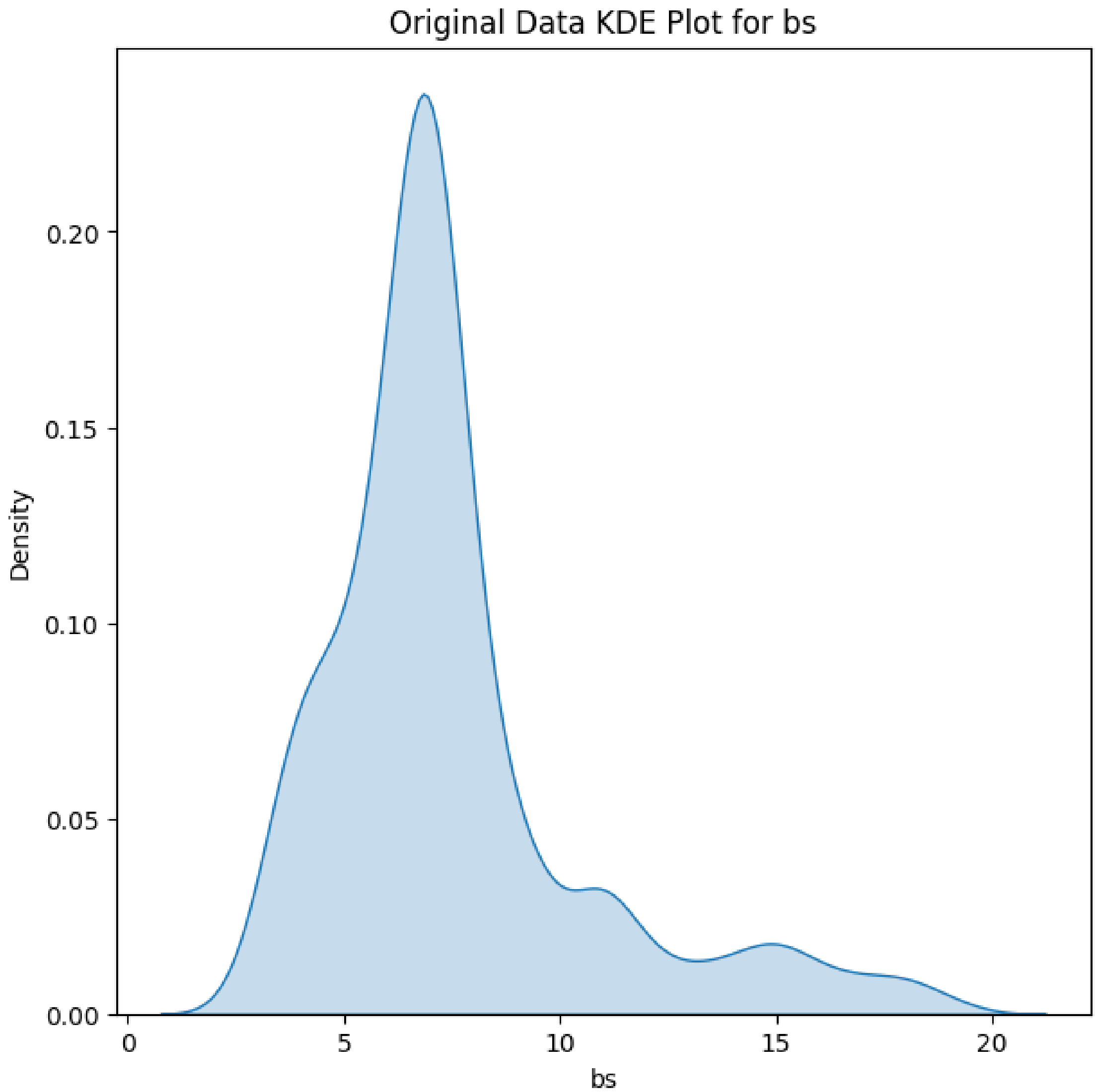
6 KDE Plot

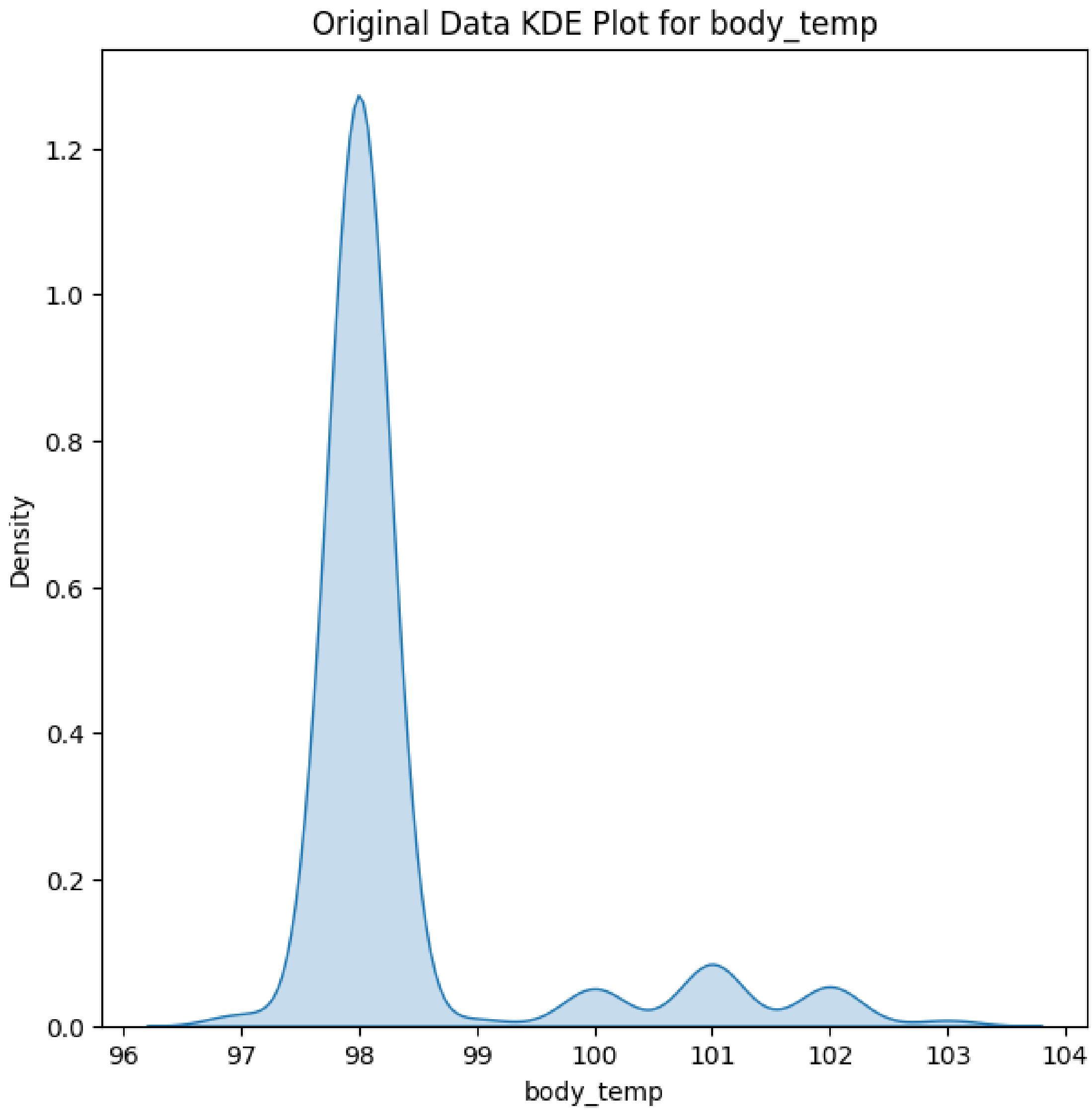
[54]:
```
# For each feature in the dataframe
for i in df_cp.columns:
    plt.figure(figsize=(7,7))
    sns.kdeplot(data=df_cp, x=i, fill=True)  # KDE without hue
    plt.title(f'Original Data KDE Plot for {i}')
    plt.show()
```

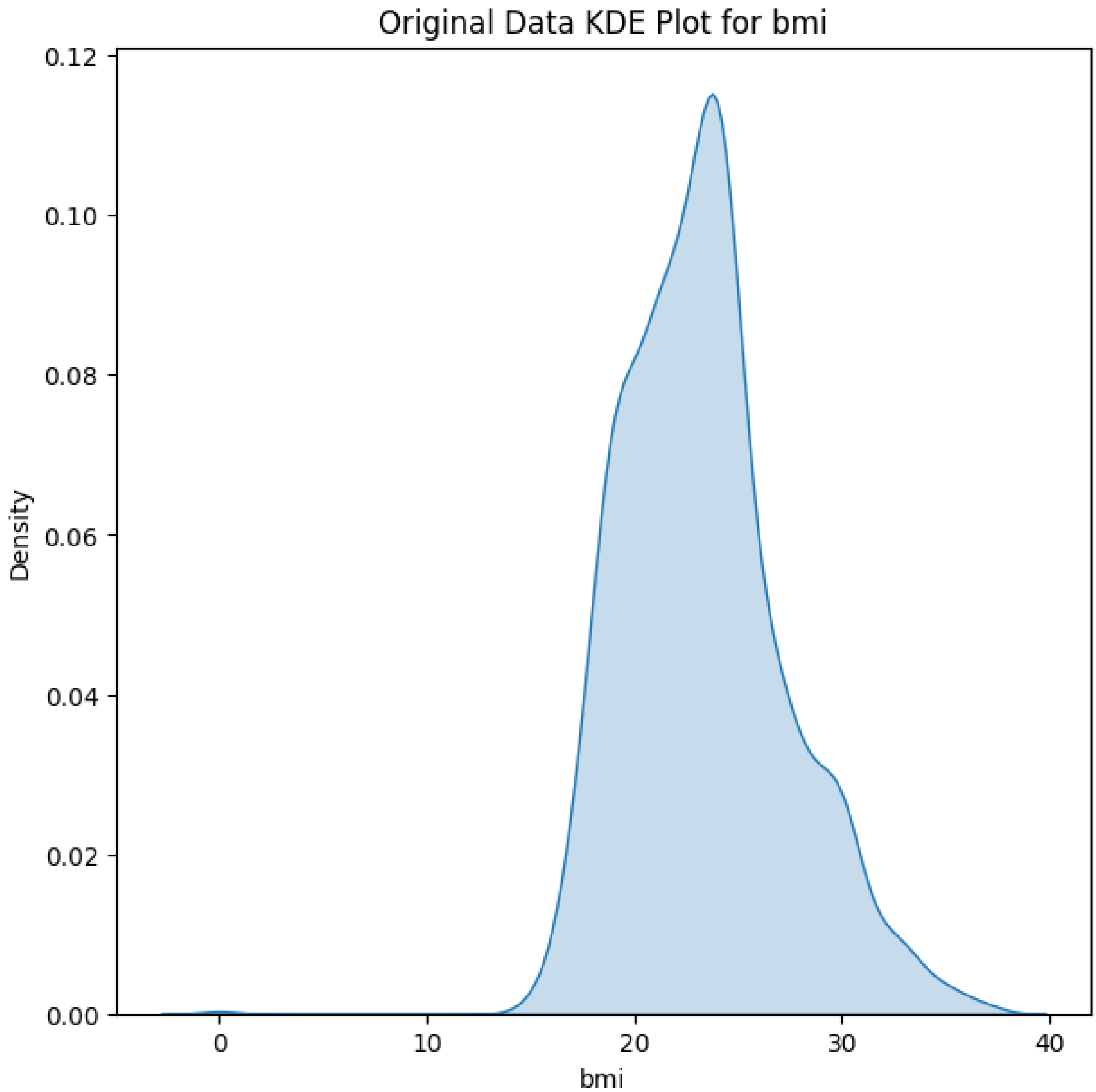

Original Data KDE Plot for age
Density
0.05
0.04
0.03
0.02
0.01
0.00
0
50
100
150
200
250
300
350
age

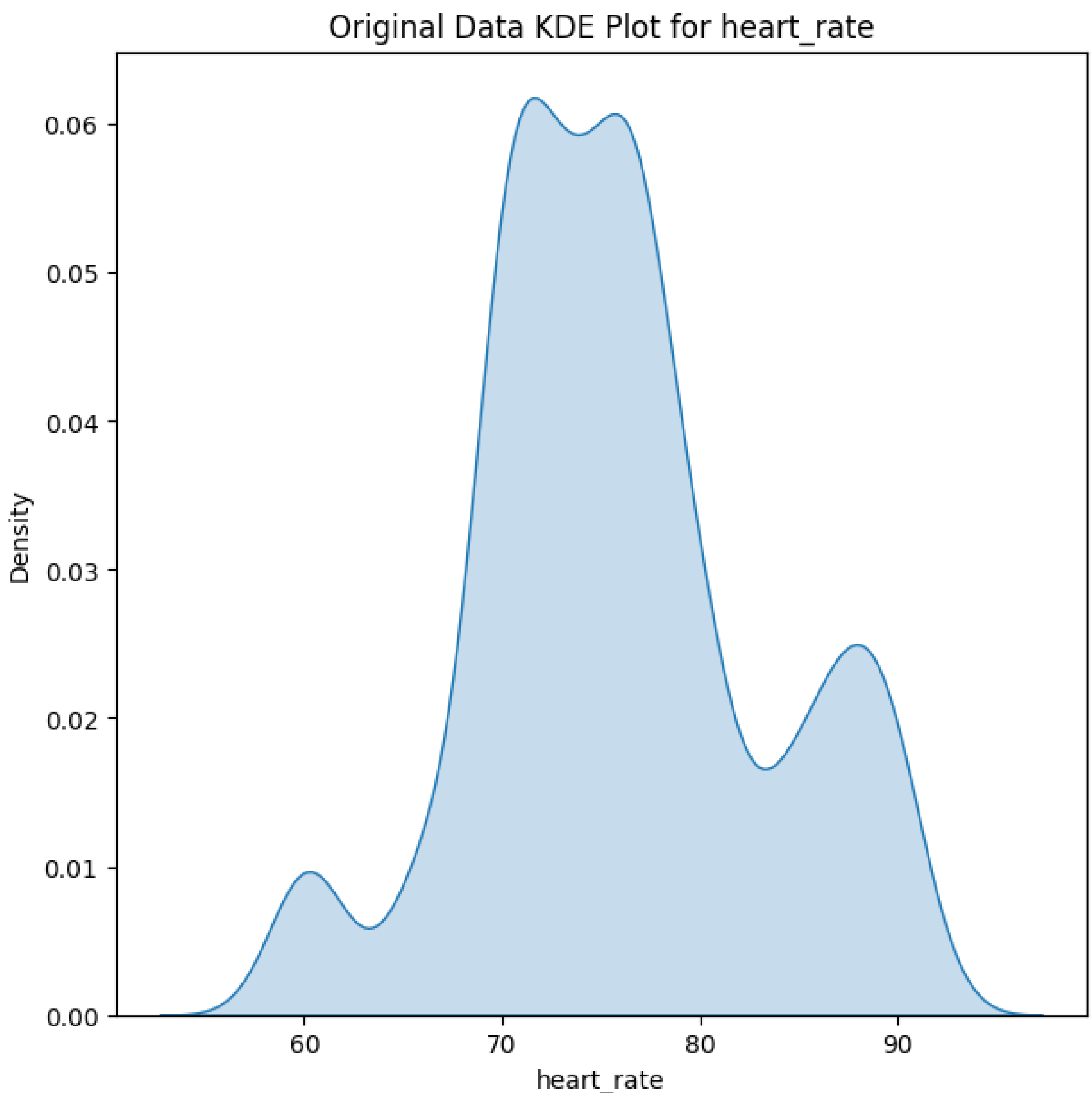

Original Data KDE Plot for systolic_bp
Density
0.030
0.025
0.020
0.015
0.010
0.005
0.000
60
80
100
120
140
160
180
200
220
systolic_bp


Original Data KDE Plot for diastolic
Density
0.025
0.020
0.015
0.010
0.005
0.000
40
60
80
100
120
140
diastolic

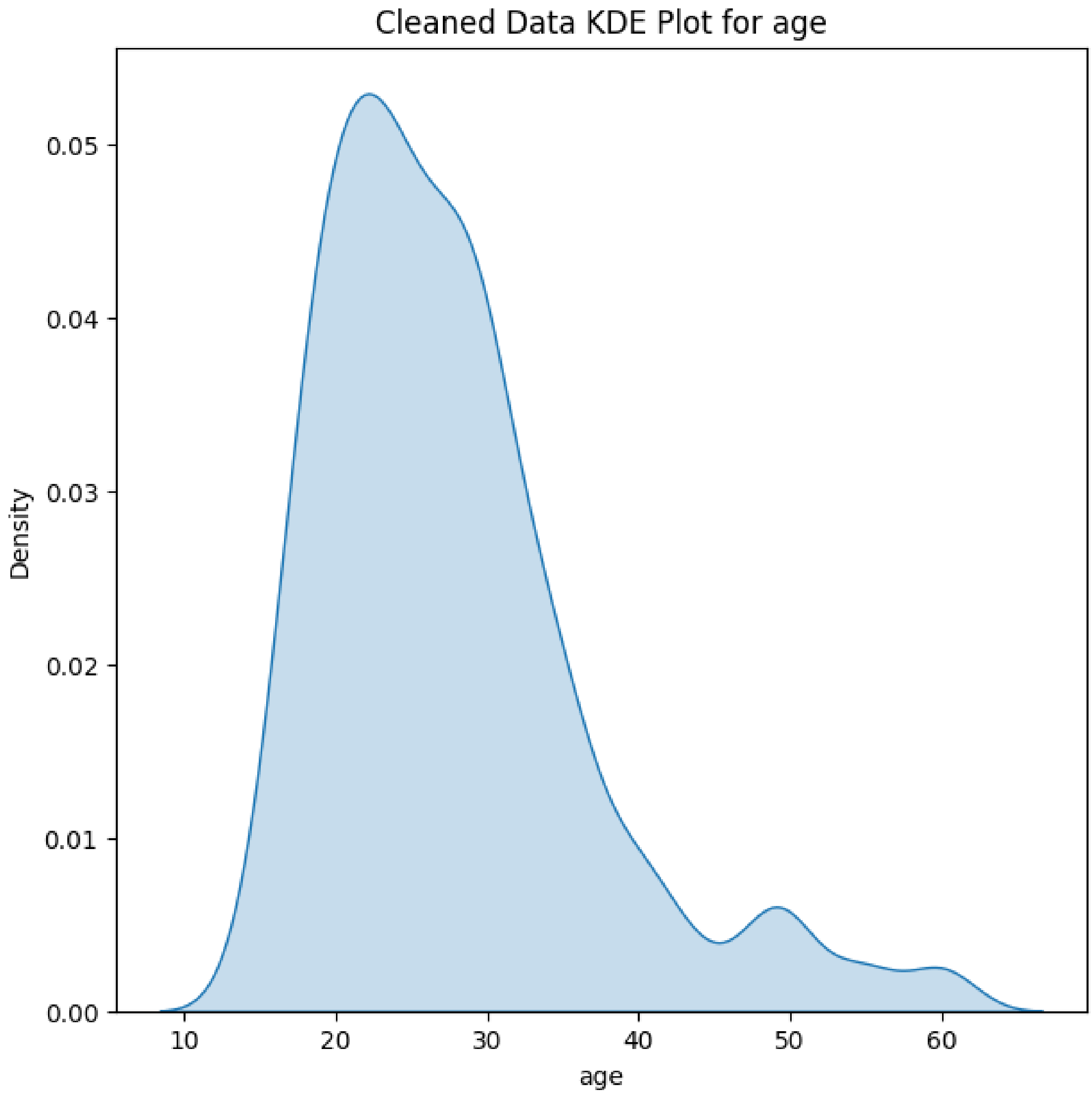

Original Data KDE Plot for bs
Density
0.00
0.05
0.10
0.15
0.20
0
5
10
15
20
bs

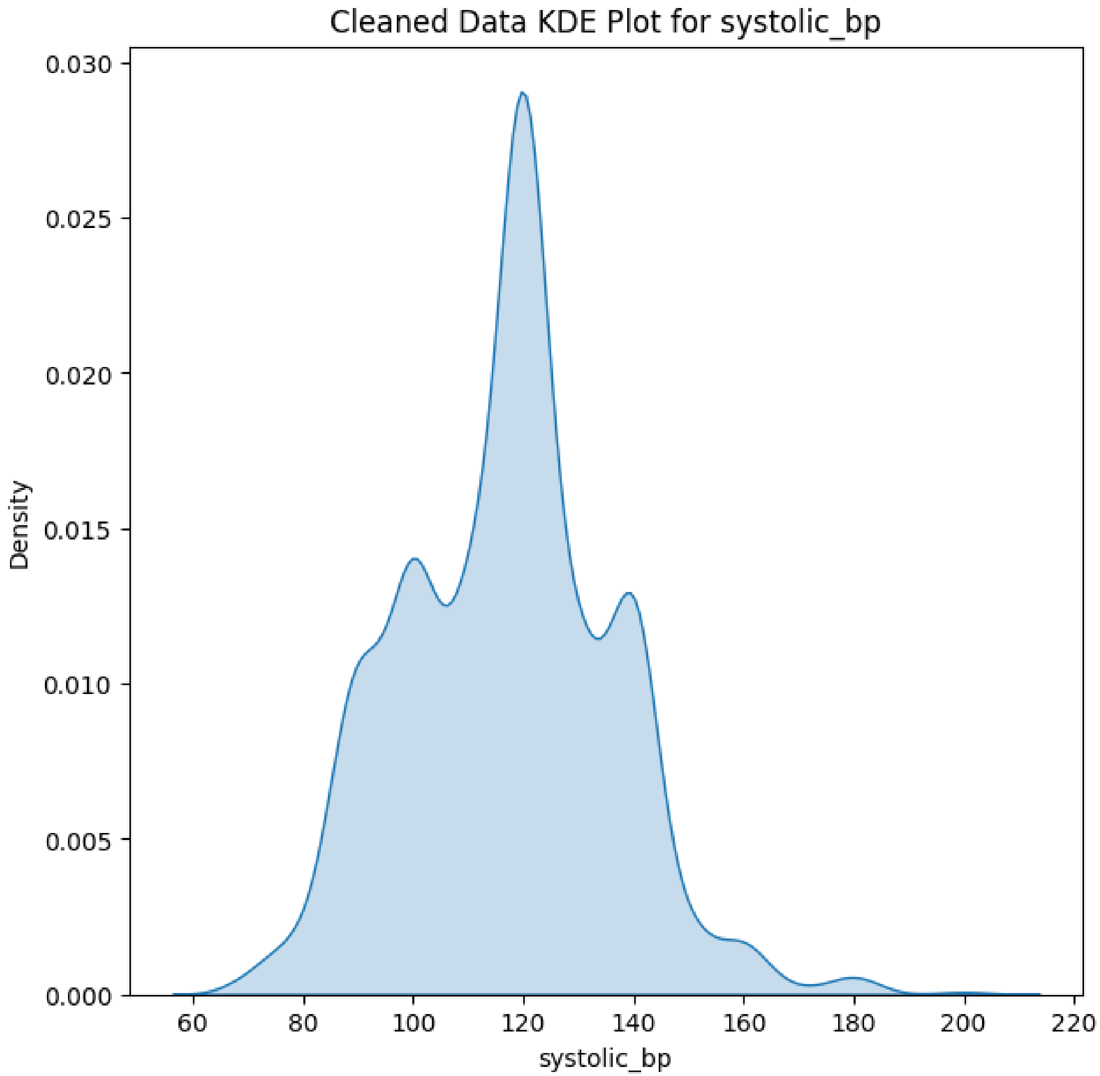

Original Data KDE Plot for body_temp
Density
1.2
1.0
0.8
0.6
0.4
0.2
0.0
96
97
98
99
100
101
102
103
104
body_temp

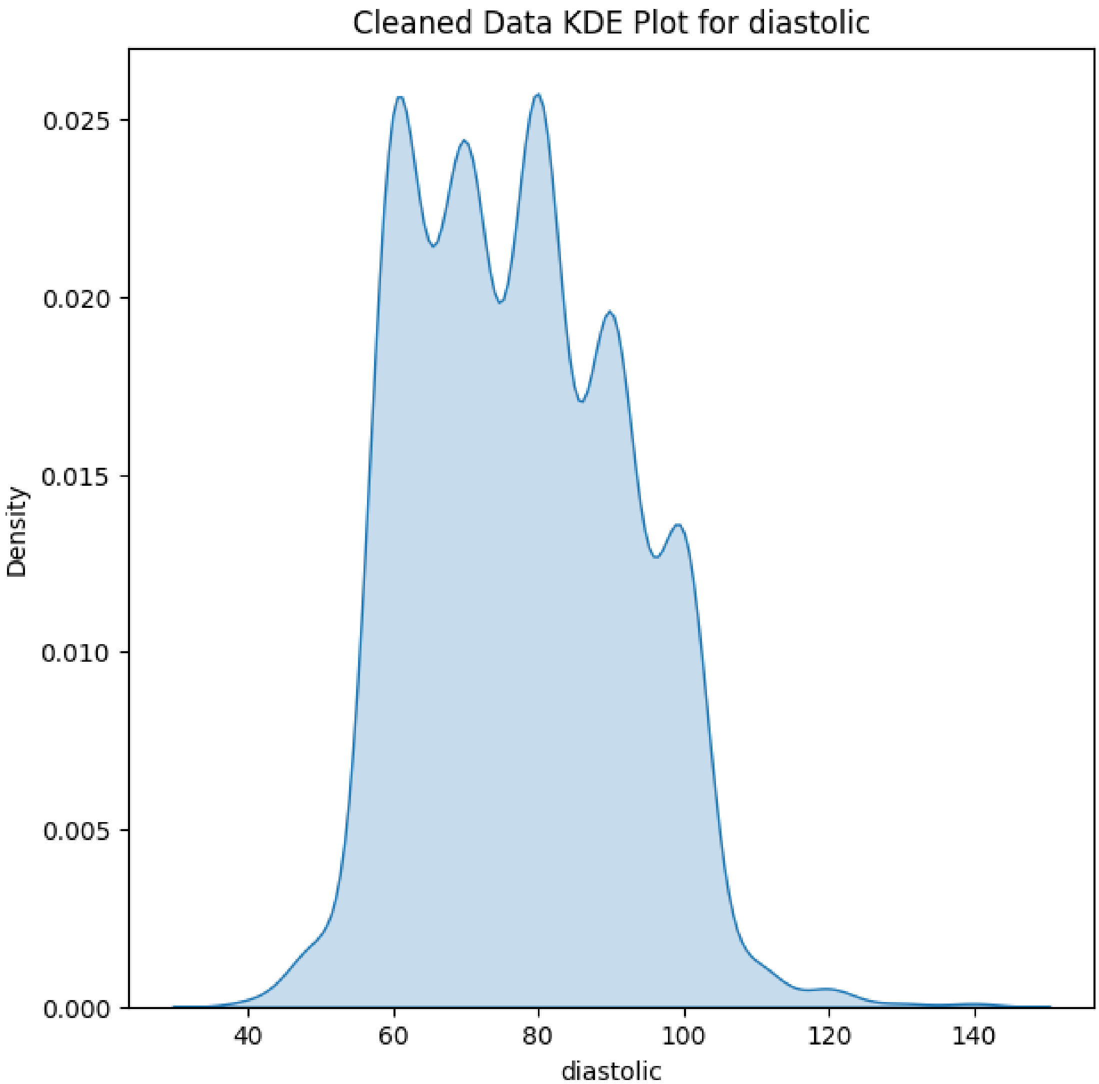

Original Data KDE Plot for bmi
0.12
0.10
0.08
0.06
0.04
0.02
0.00
Density
0
10
20
30
40
bmi

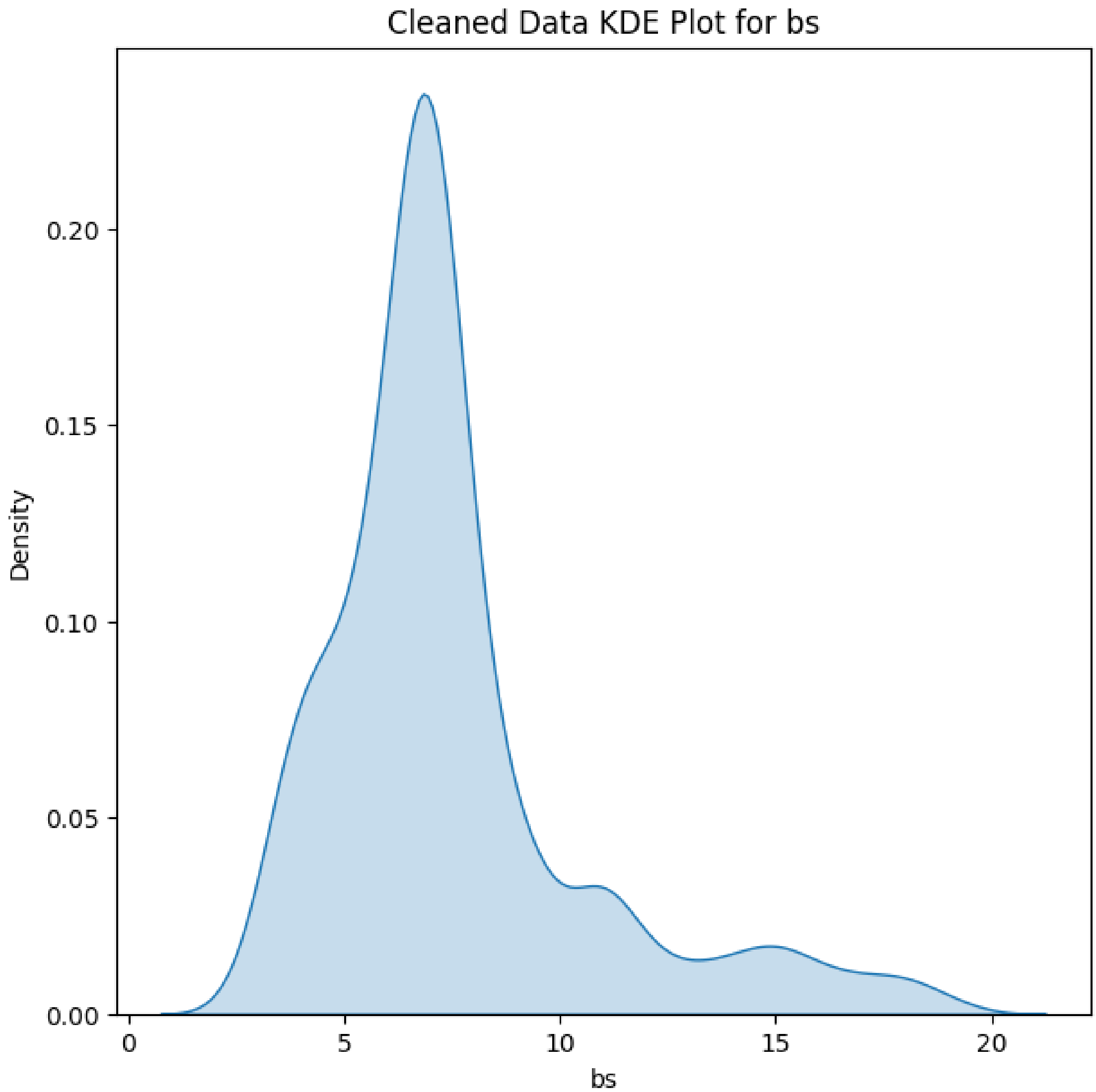


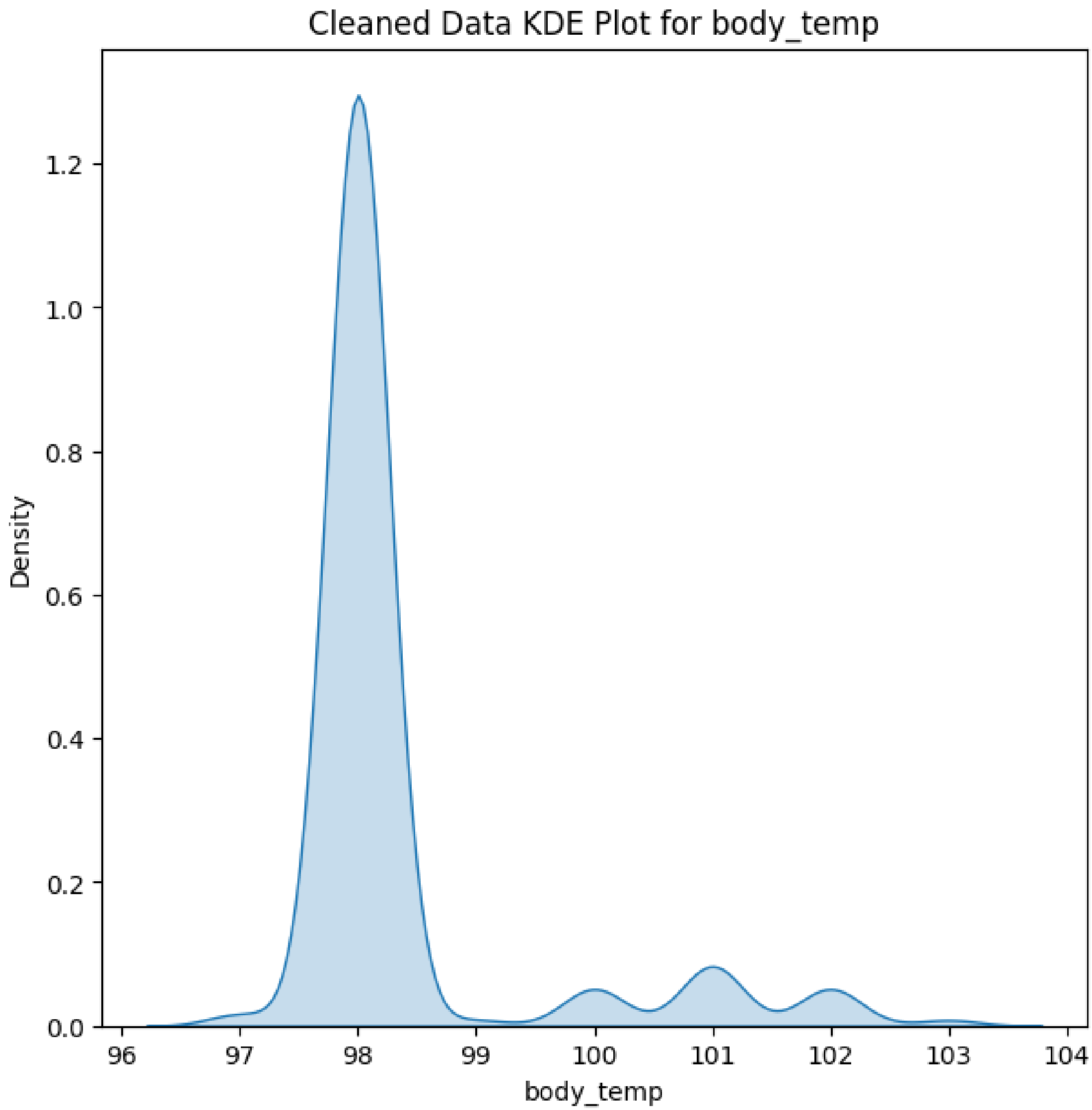
[55]:
```
# For each feature in the dataframe
for i in df_cln.columns:
    plt.figure(figsize=(7,7))
    sns.kdeplot(data=df_cln, x=i, fill=True)  # KDE without hue
    plt.title(f'Cleaned Data KDE Plot for {i}')
    plt.show()
```


Cleaned Data KDE Plot for age
Density
0.05
0.04
0.03
0.02
0.01
0.00
10
20
30
40
50
60
age

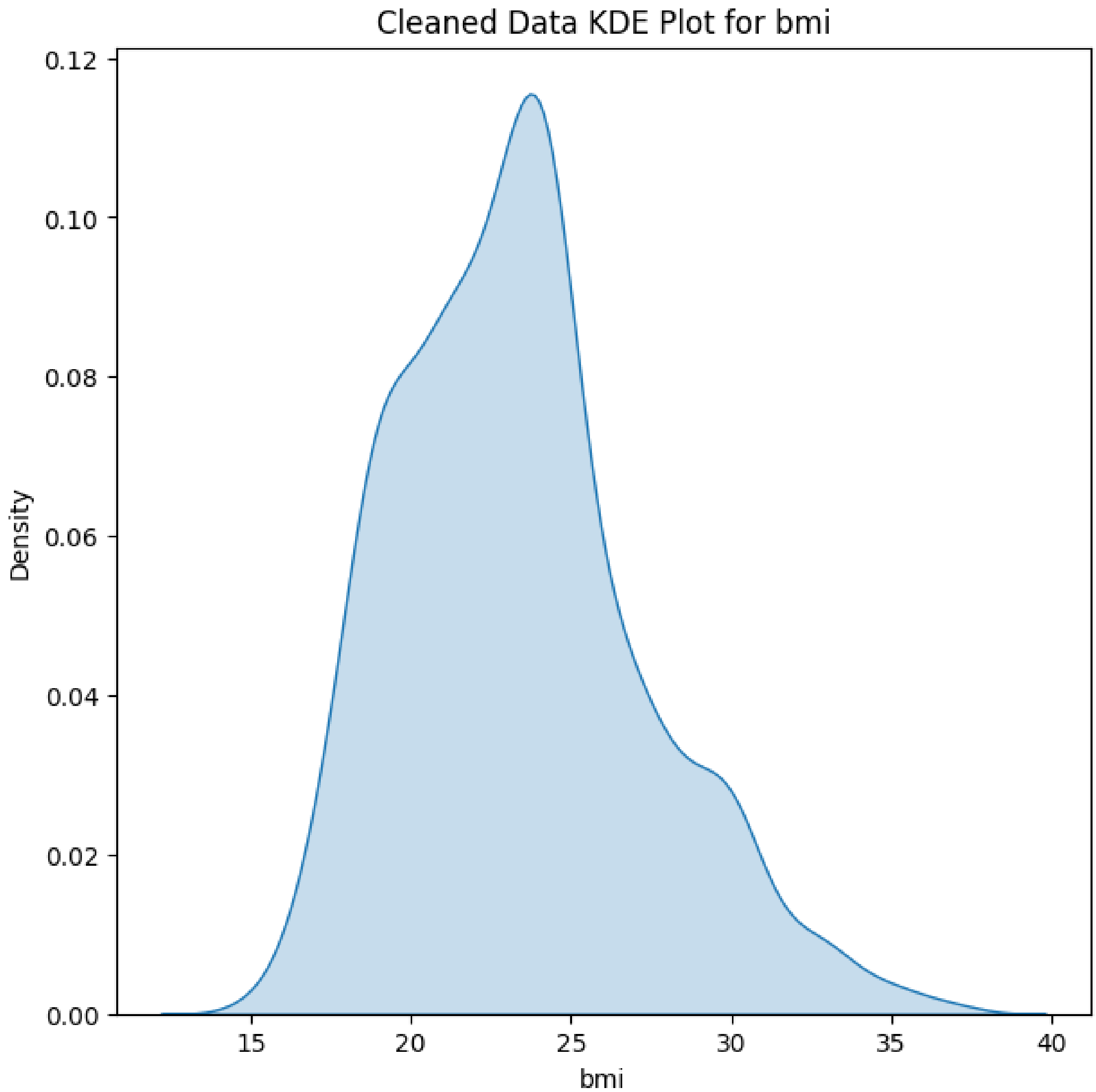

Cleaned Data KDE Plot for systolic_bp
Density
0.000
0.005
0.010
0.015
0.020
0.025
0.030
60
80
100
120
140
160
180
200
220
systolic_bp

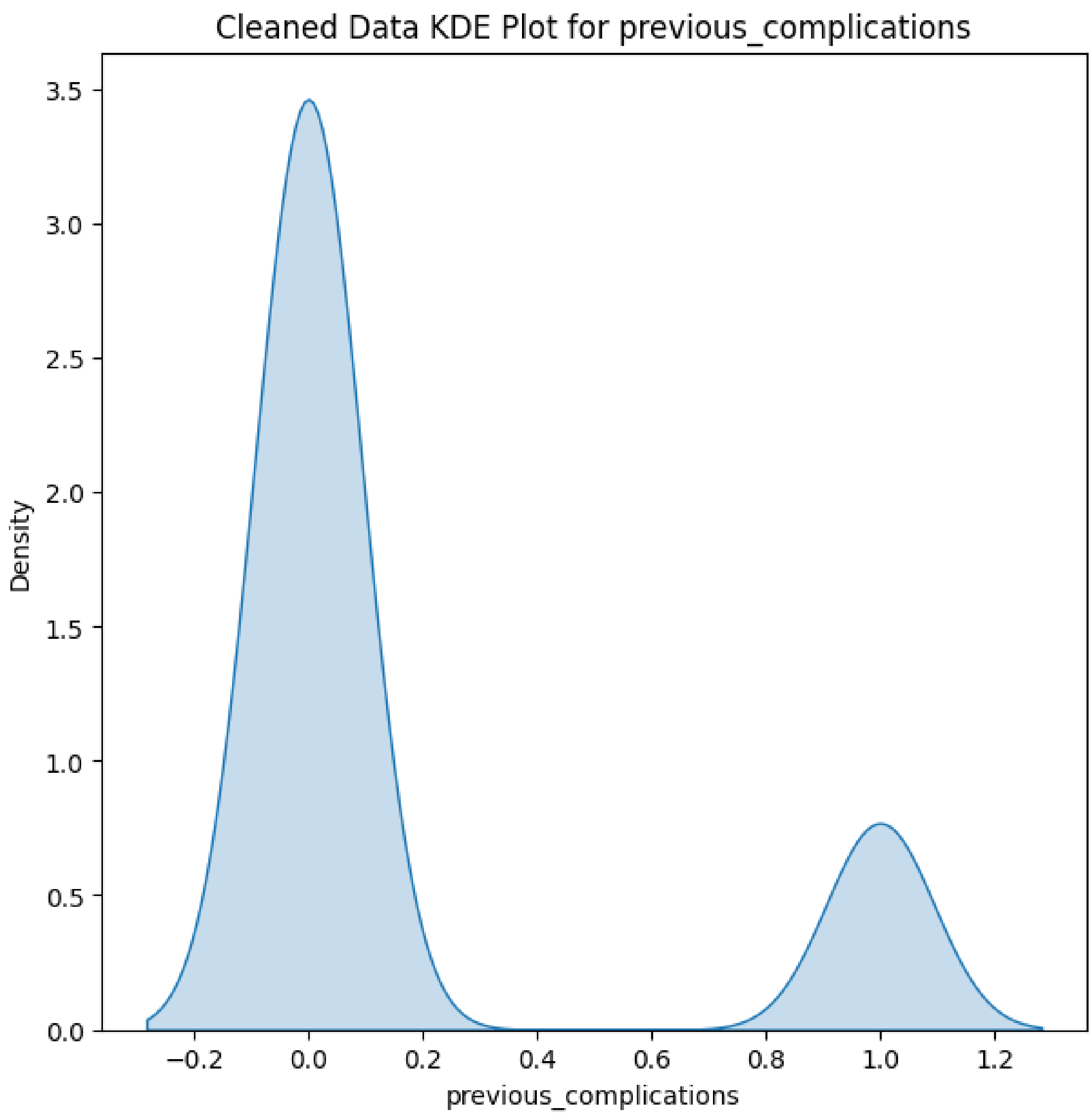

Cleaned Data KDE Plot for diastolic
Density
0.000
0.005
0.010
0.015
0.020
0.025
40
60
80
100
120
140
diastolic

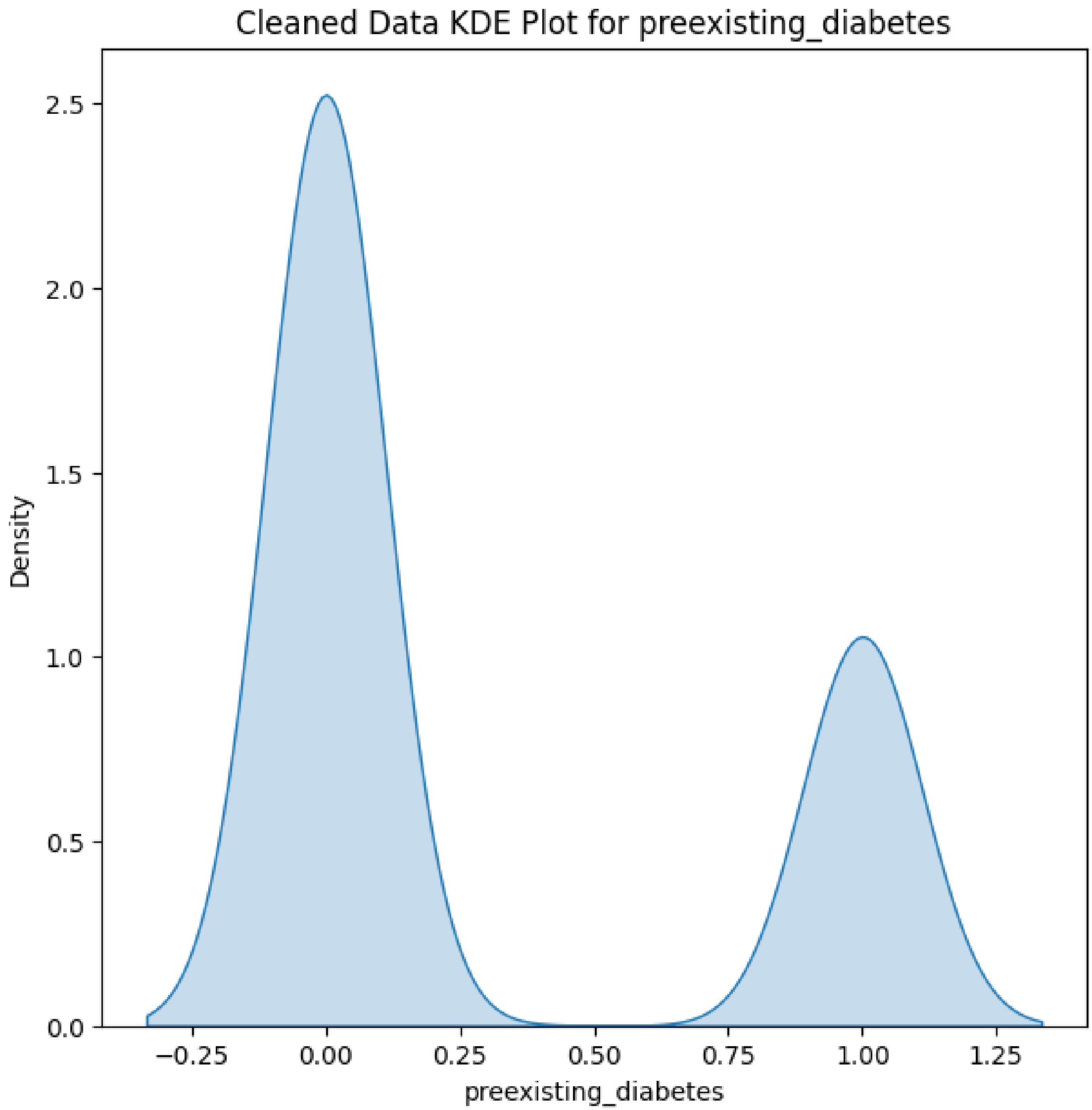

Cleaned Data KDE Plot for bs
Density
0.00
0.05
0.10
0.15
0.20
0
5
10
15
20
bs

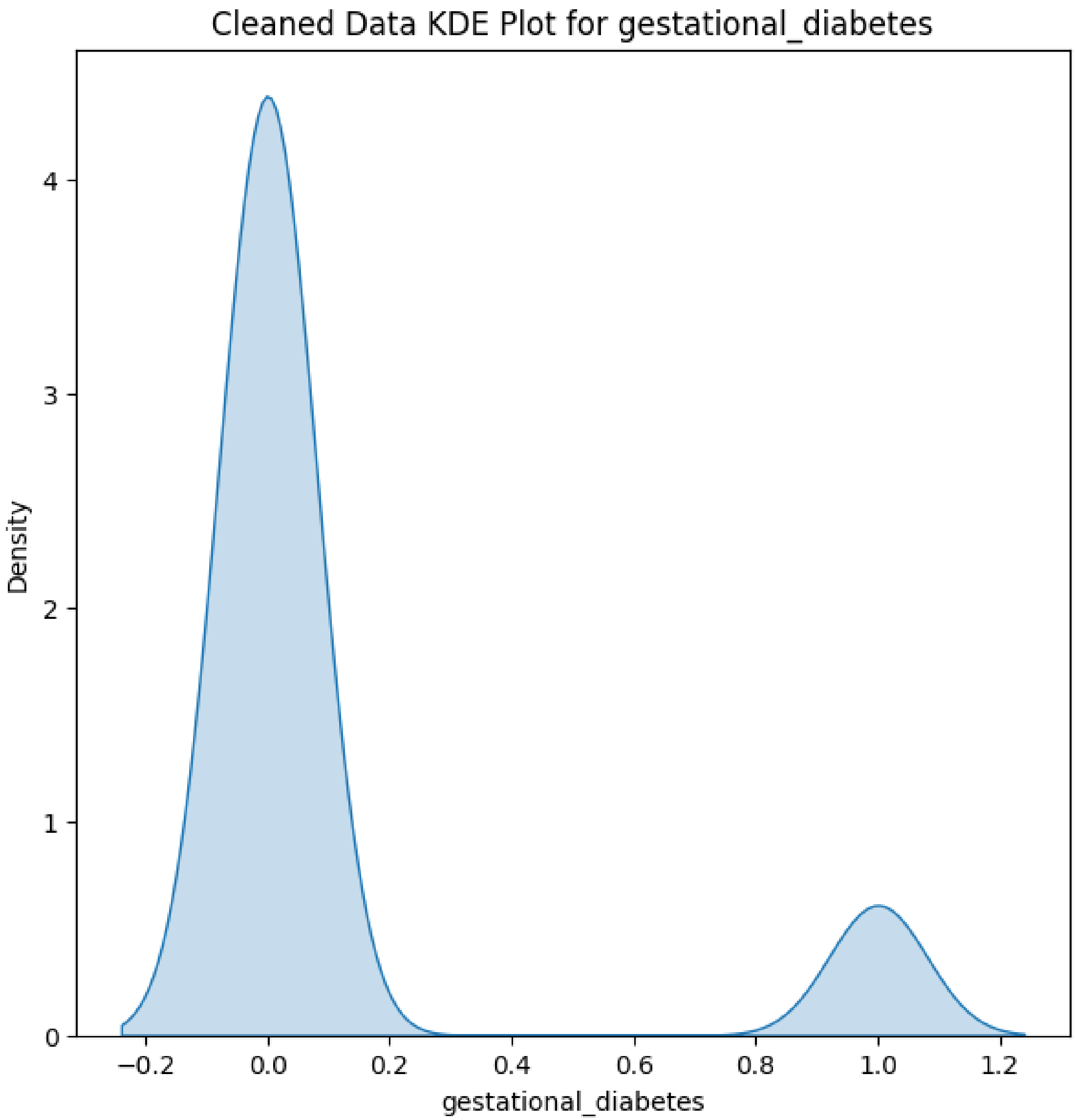


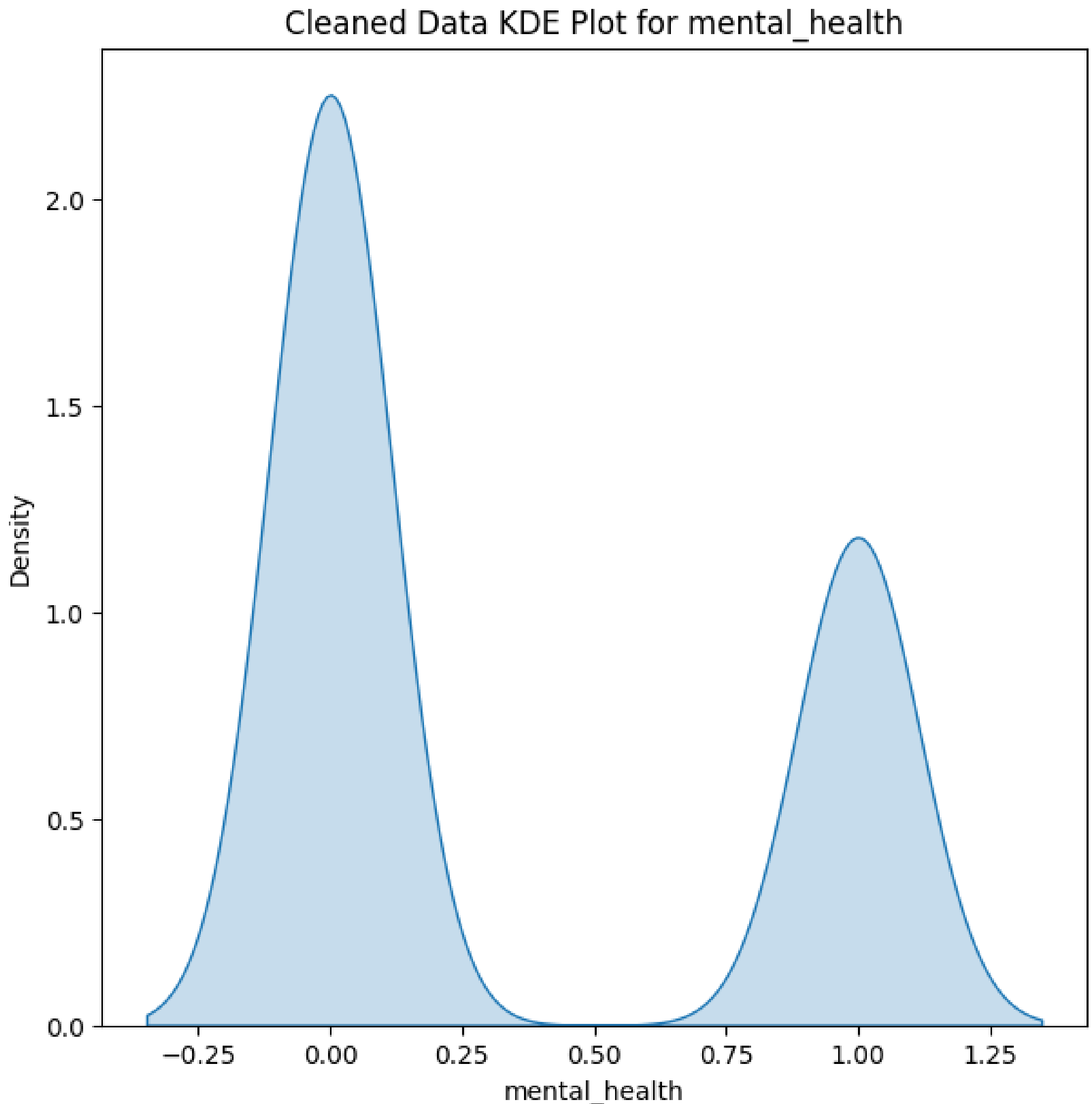
Cleaned Data KDE Plot for body_temp
Density
body_temp
0.0
0.2
0.4
0.6
0.8
1.0
1.2
96
97
98
99
100
101
102
103
104


Cleaned Data KDE Plot for bmi
Density
0.00
0.02
0.04
0.06
0.08
0.10
0.12
15
20
25
30
35
40
bmi

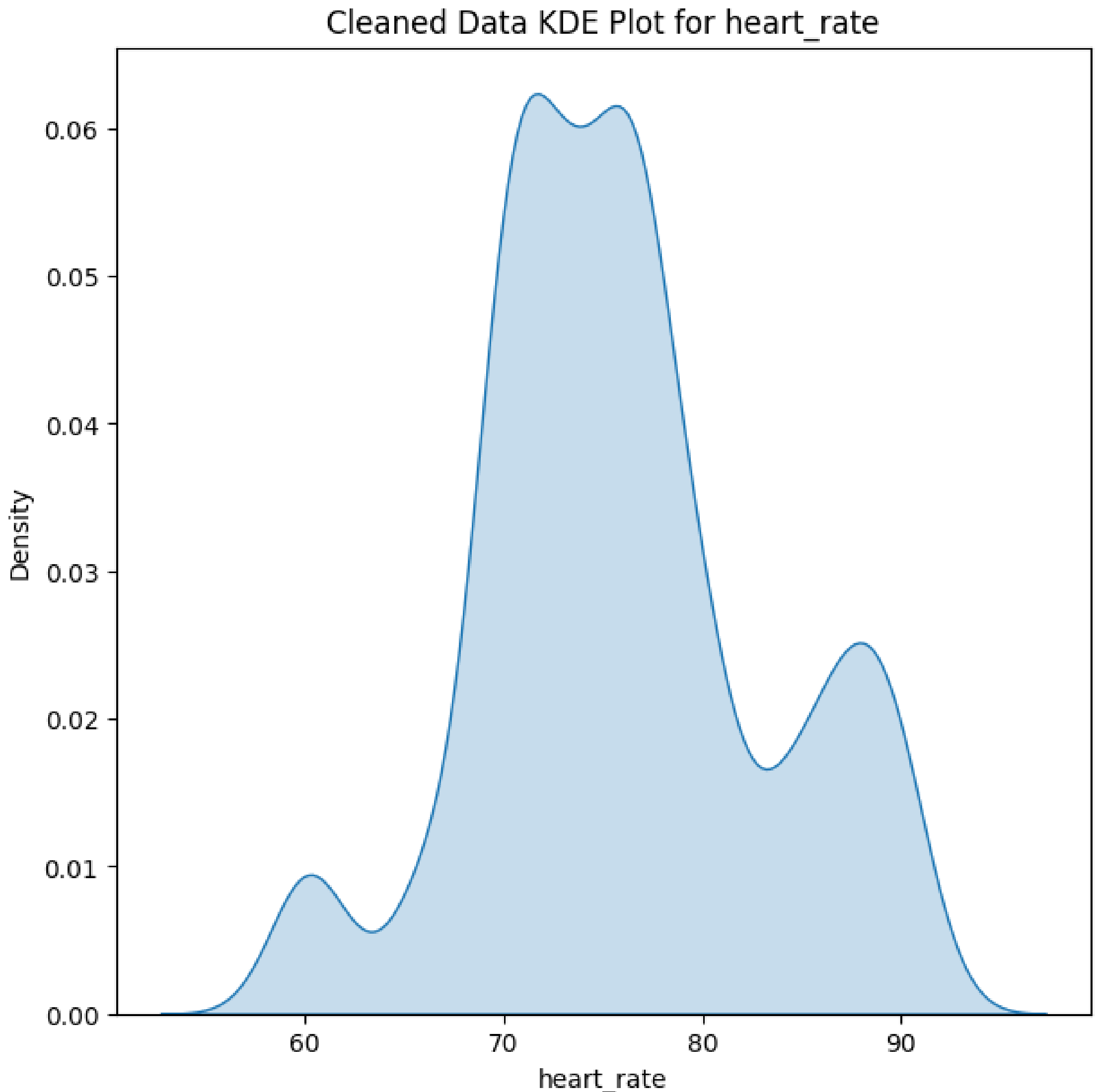

Cleaned Data KDE Plot for previous_complications
Density
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
previous_complications


Cleaned Data KDE Plot for preexisting_diabetes
Density
2.5
2.0
1.5
1.0
0.5
0.0
−0.25
0.00
0.25
0.50
0.75
1.00
1.25
preexisting_diabetes

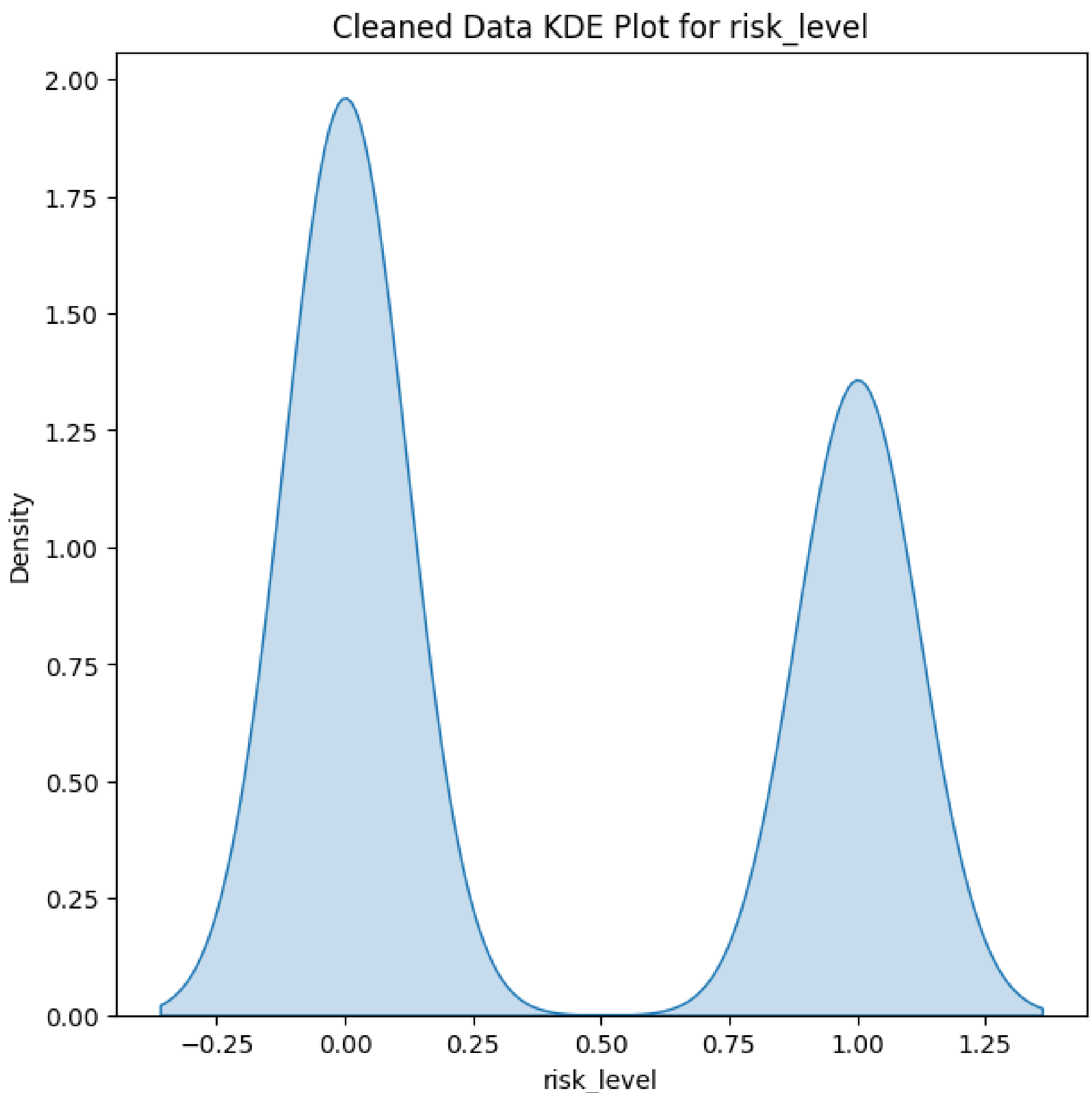

Cleaned Data KDE Plot for gestational_diabetes
Density
gestational_diabetes
0
1
2
3
4
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2



Cleaned Data KDE Plot for mental_health
Density
2.0
1.5
1.0
0.5
0.0
−0.25
0.00
0.25
0.50
0.75
1.00
1.25
mental_health



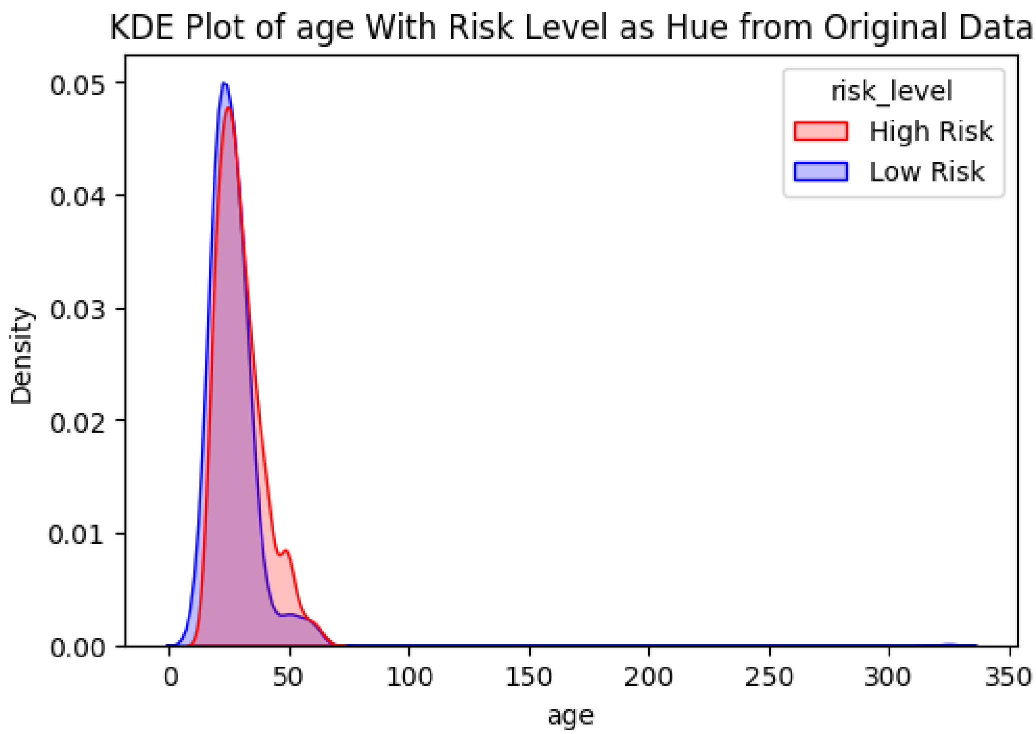
Cleaned Data KDE Plot for heart_rate
Density
heart_rate
0.00
0.01
0.02
0.03
0.04
0.05
0.06
60
70
80
90

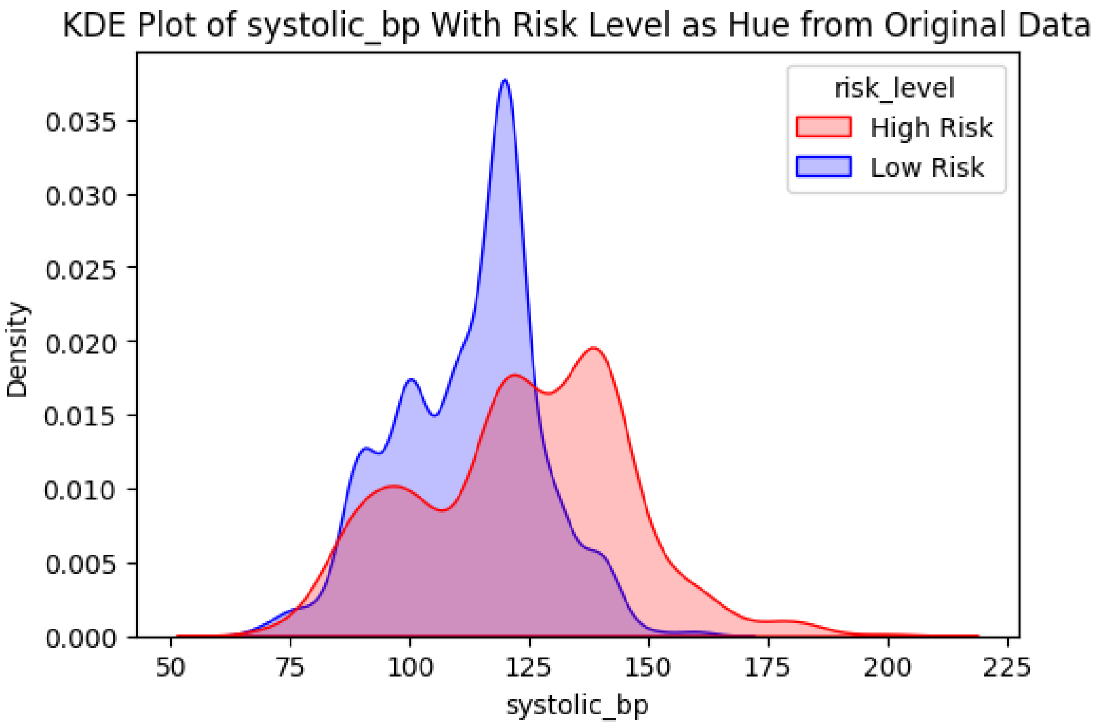


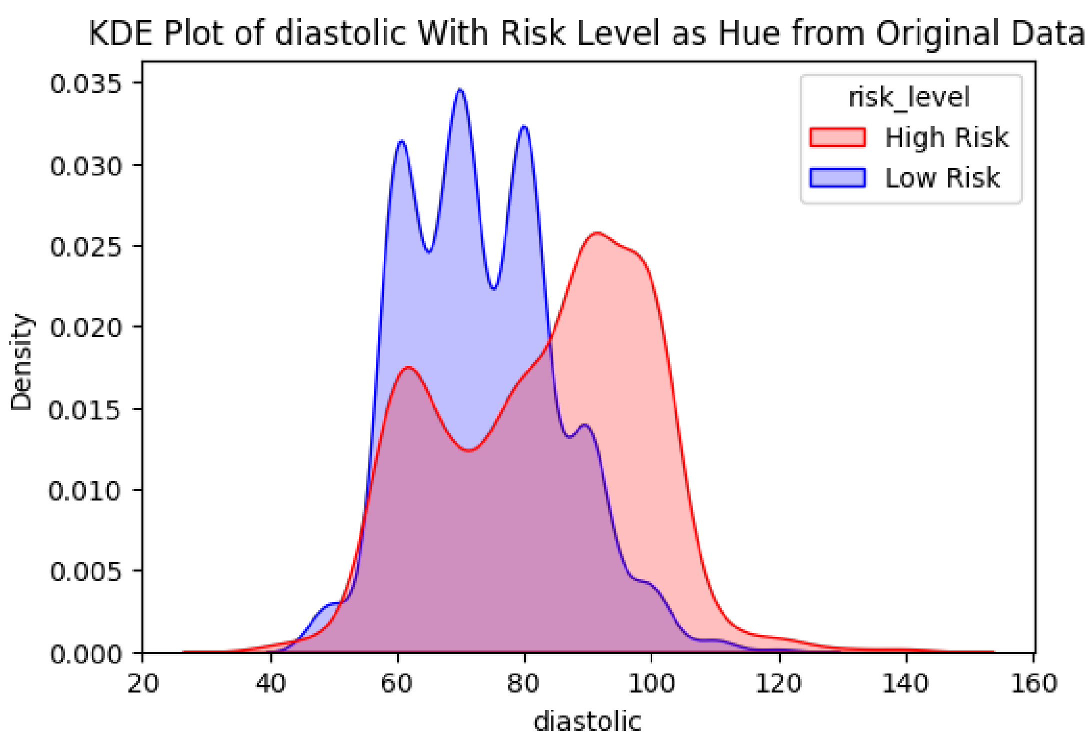
[56]:

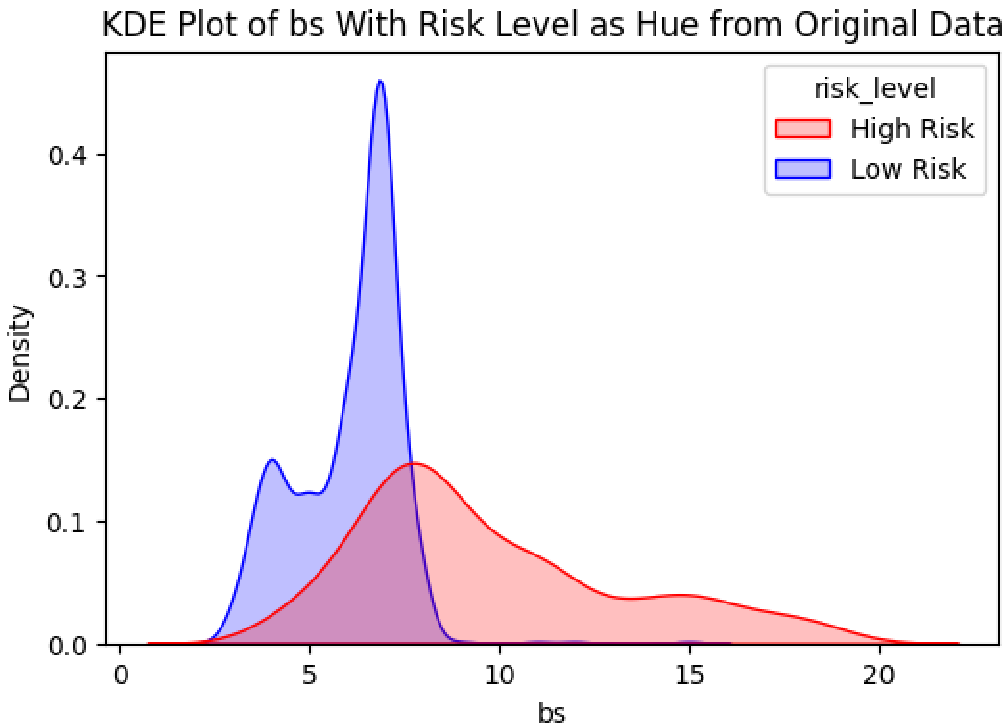
```
# Define color palette
risk_palette = {0: "blue", 1: "red"}
label_map = {0: "Low Risk", 1: "High Risk"}

num_cols = ['age', 'systolic_bp', 'diastolic', 'bs', 'body_temp', 'bmi',␣
↪'heart_rate']

for col in num_cols:
    plt.figure(figsize=(6, 4))

    sns.kdeplot(
        data=df,
        x=col,
```

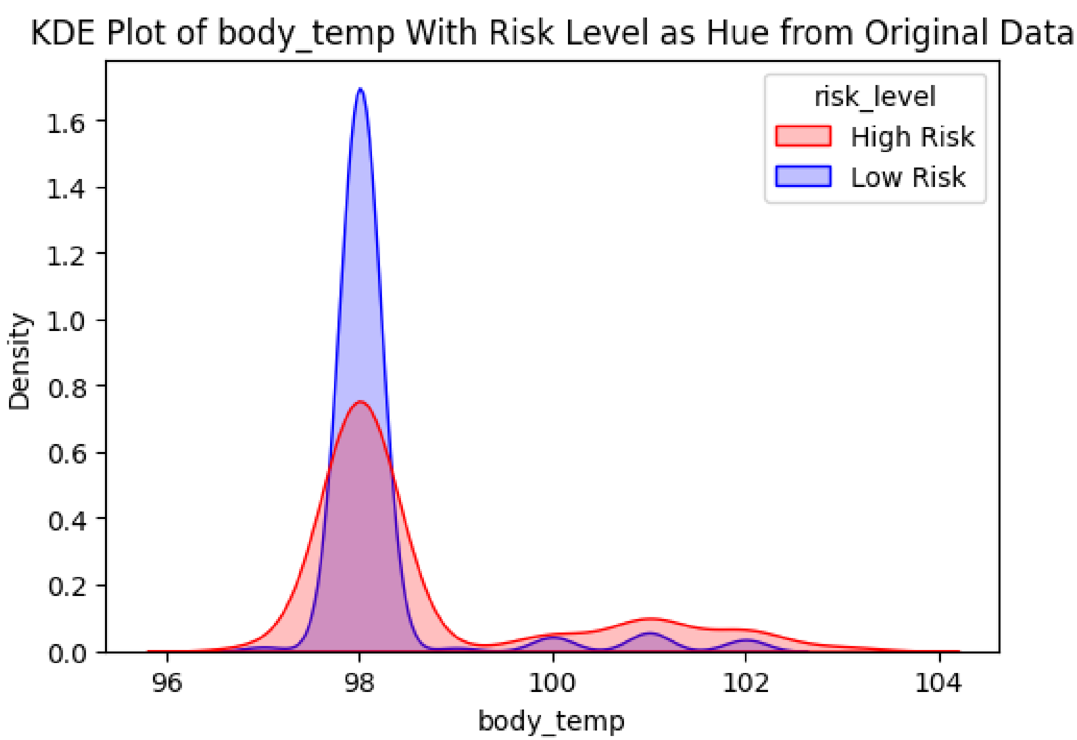
```
        hue=df['risk_level'].map(label_map),
        fill=True,
        common_norm=False,
        palette={"Low Risk": "blue", "High Risk": "red"}
    )

    plt.title(f'KDE Plot of {col} With Risk Level as Hue from Original Data')
    plt.xlabel(col)
    plt.ylabel('Density')
    plt.show()
```

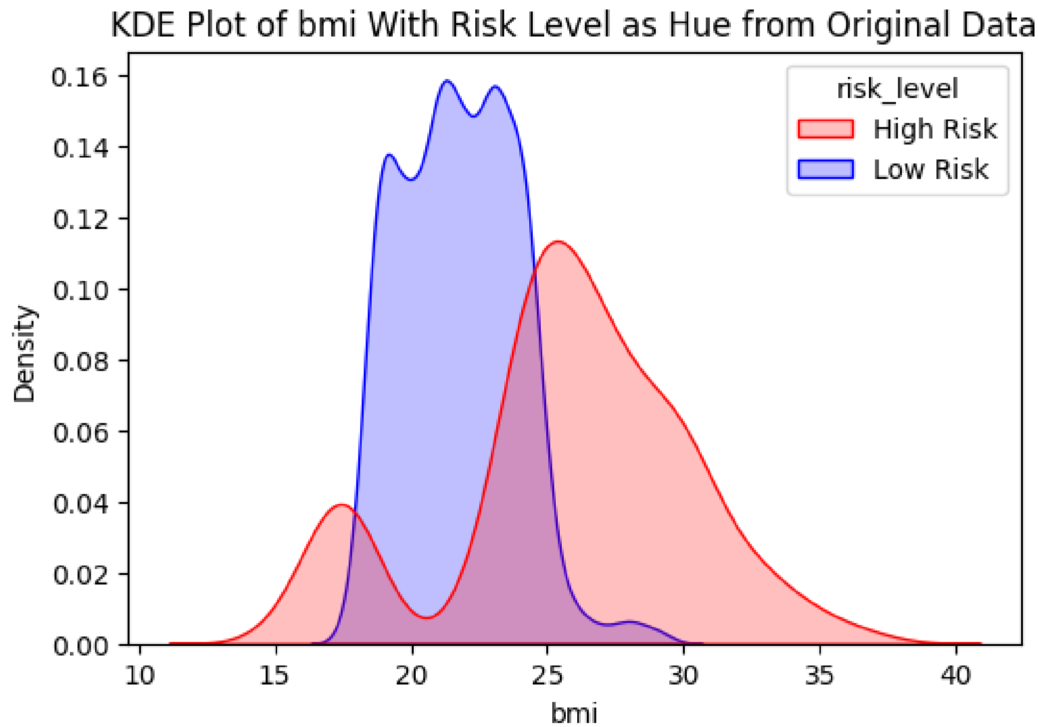


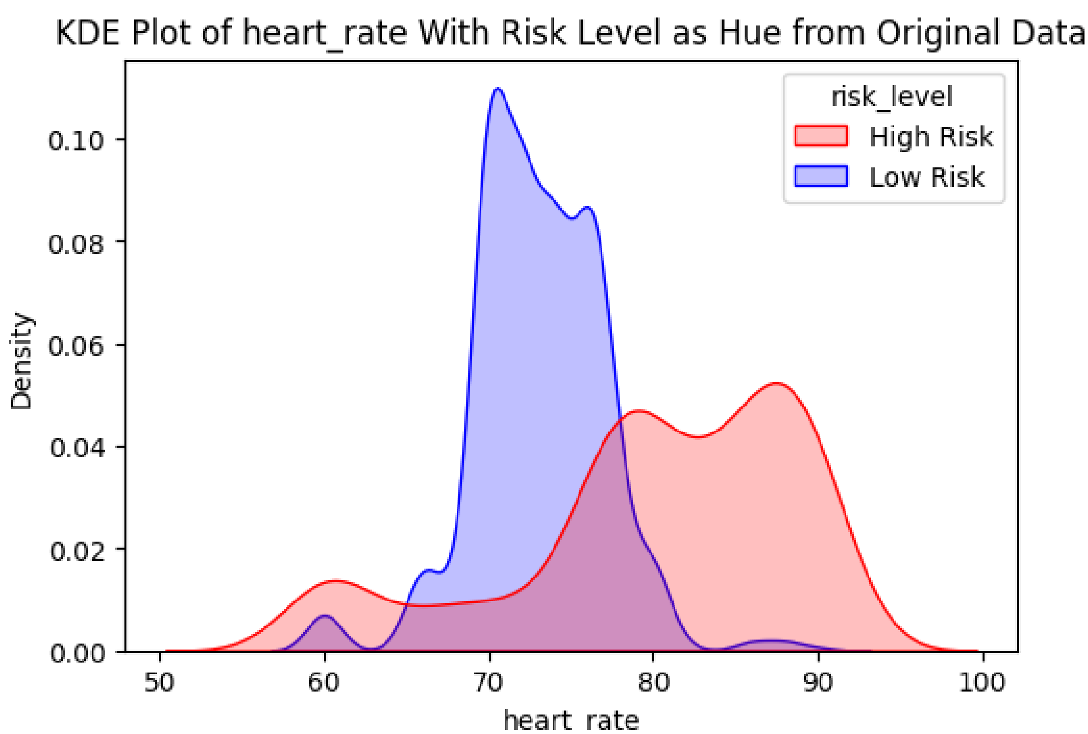

KDE Plot of systolic_bp With Risk Level as Hue from Original Data
risk_level
High Risk
Low Risk
Density
0.000
0.005
0.010
0.015
0.020
0.025
0.030
0.035
50
75
100
125
150
175
200
225
systolic_bp


KDE Plot of diastolic With Risk Level as Hue from Original Data
risk_level
High Risk
Low Risk
Density
0.000
0.005
0.010
0.015
0.020
0.025
0.030
0.035
20
40
60
80
100
120
140
160
diastolic

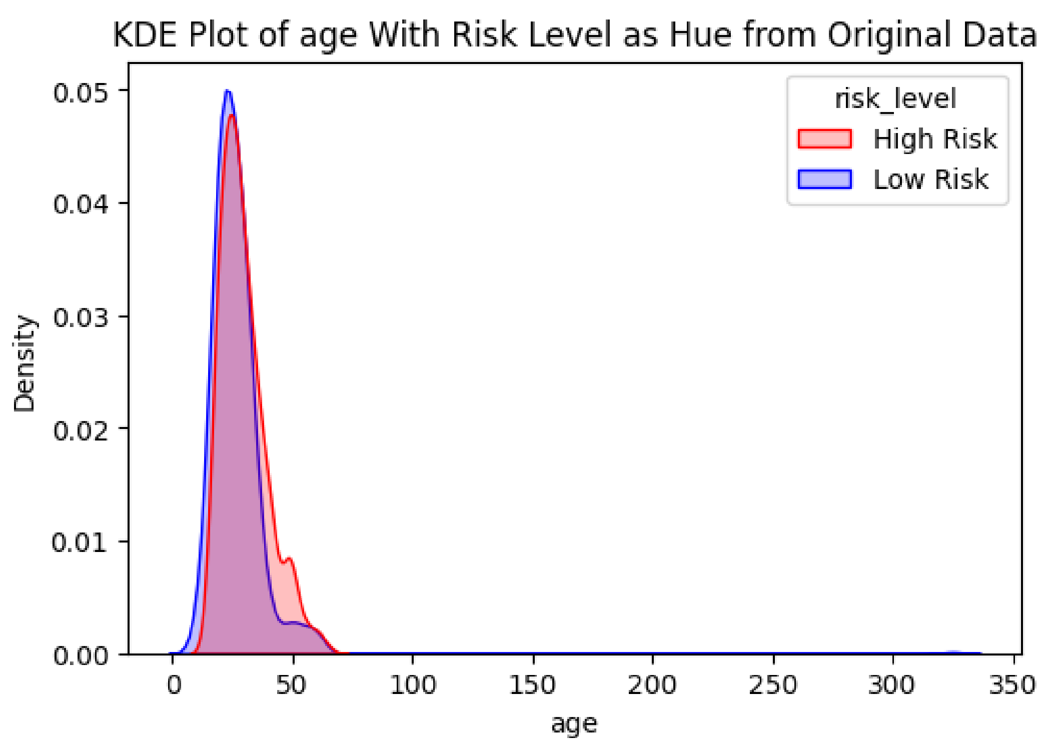

KDE Plot of bs With Risk Level as Hue from Original Data
risk_level
High Risk
Low Risk
Density
0.0
0.1
0.2
0.3
0.4
0
5
10
15
20
bs

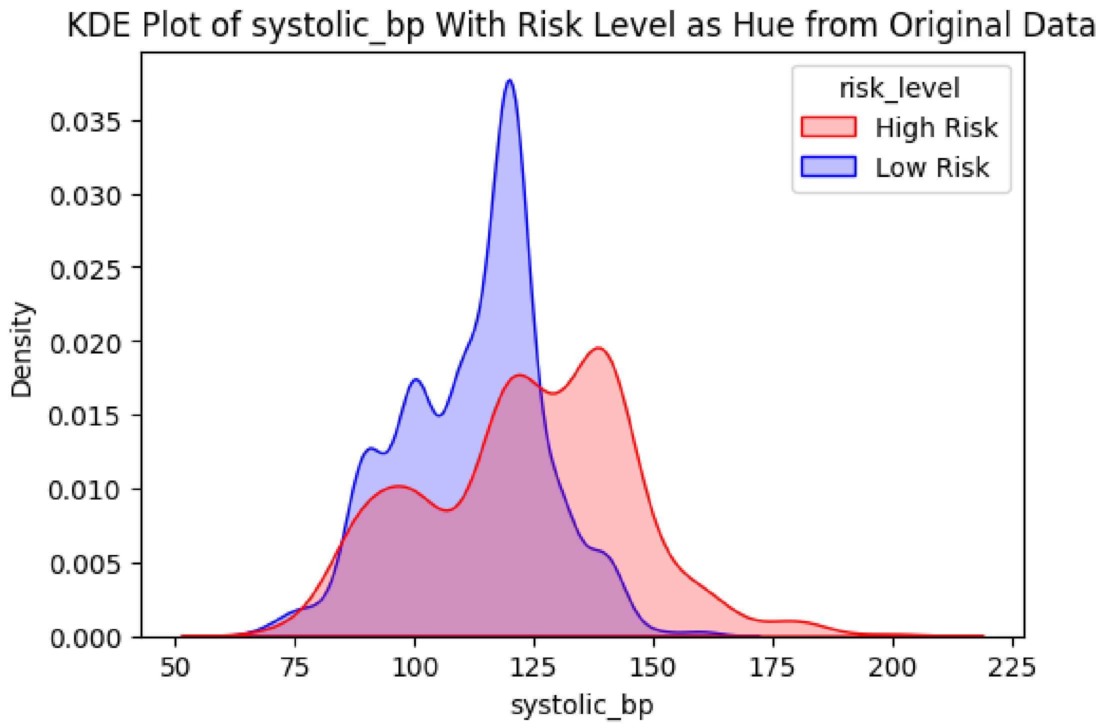

KDE Plot of body_temp With Risk Level as Hue from Original Data
risk_level
High Risk
Low Risk
Density
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
96
98
100
102
104
body_temp

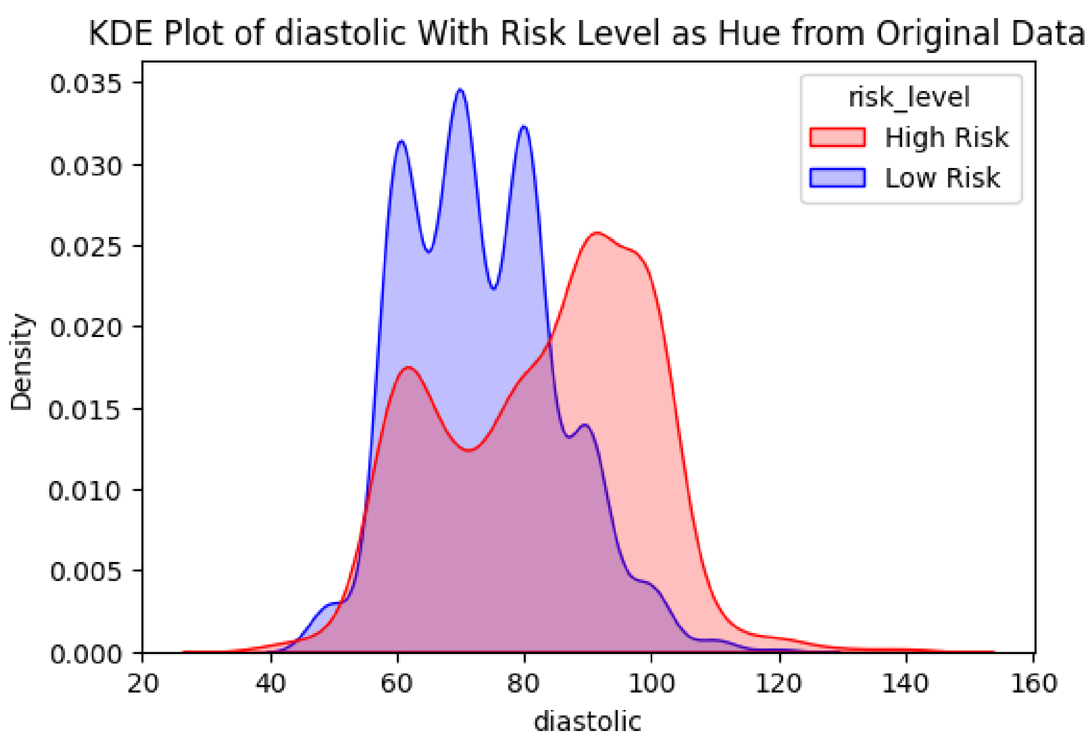

KDE Plot of bmi With Risk Level as Hue from Original Data
risk_level
High Risk
Low Risk
Density
bmi
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
0.16
10
15
20
25
30
35
40

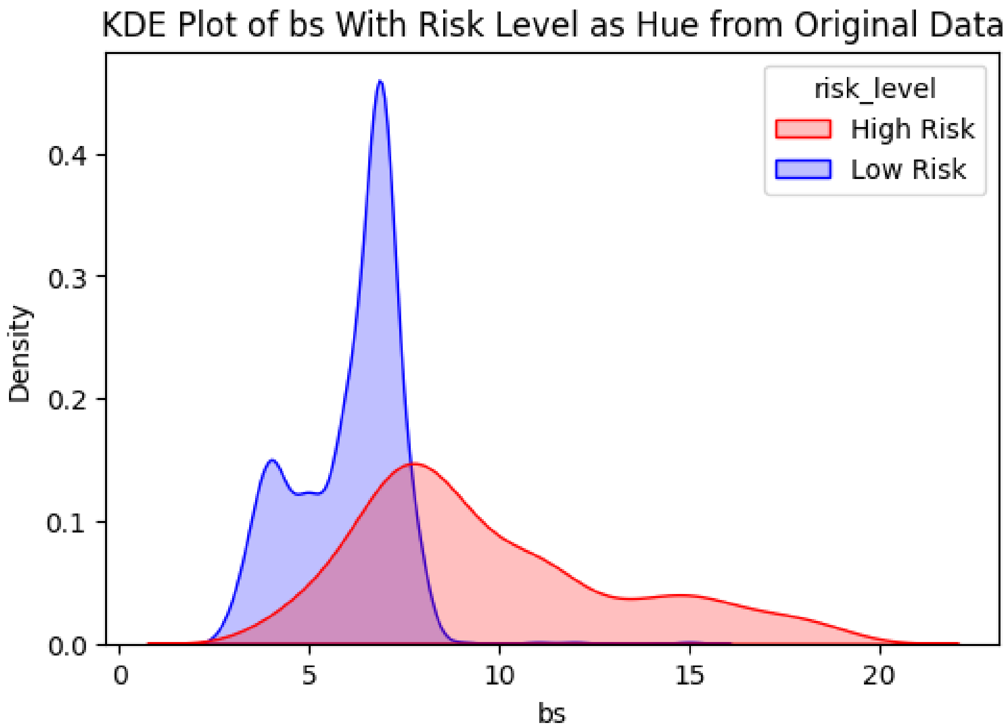

KDE Plot of heart_rate With Risk Level as Hue from Original Data
risk_level
High Risk
Low Risk
Density
heart_rate
0.00
0.02
0.04
0.06
0.08
0.10
50
60
70
80
90
100

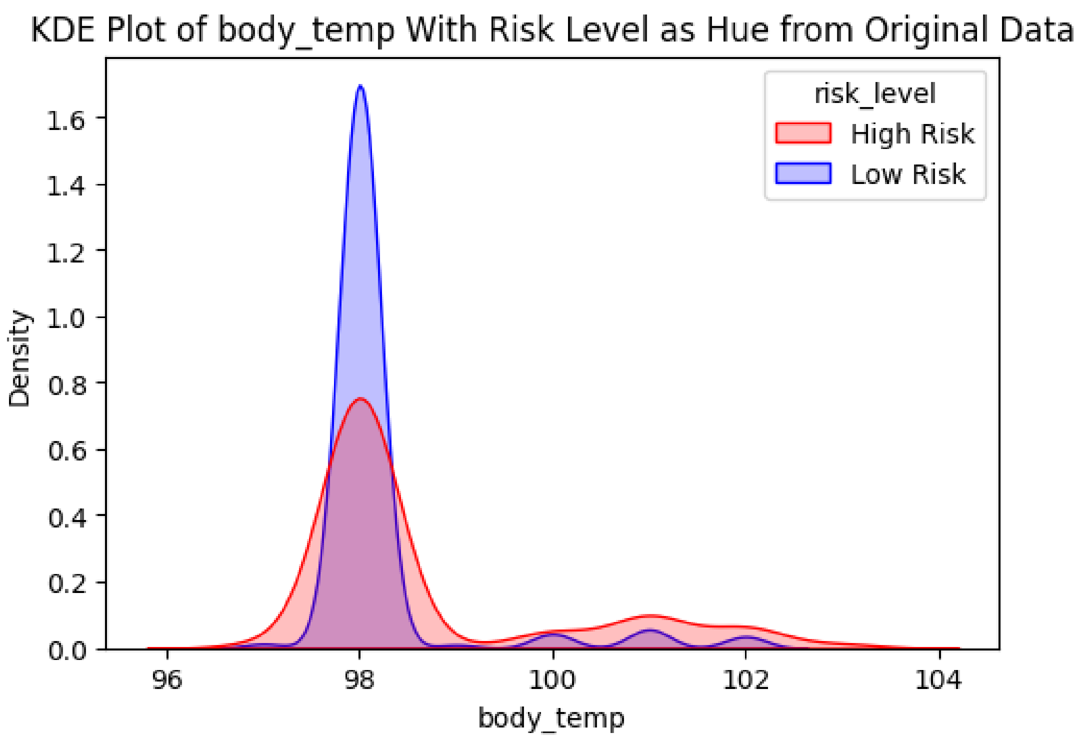
[57]:
```python
# Define color palette
risk_palette = {0: "blue", 1: "red"}
label_map = {0: "Low Risk", 1: "High Risk"}

num_cols = ['age', 'systolic_bp', 'diastolic', 'bs', 'body_temp', 'bmi',␣
 ↪'heart_rate']

for col in num_cols:
    plt.figure(figsize=(6, 4))
    sns.kdeplot(
        data=df,
        x=col,
        hue=df['risk_level'].map(label_map),
        fill=True,
        common_norm=False,
        palette={"Low Risk": "blue", "High Risk": "red"}
    )

    plt.title(f'KDE Plot of {col} With Risk Level as Hue from Original Data')
    plt.xlabel(col)
    plt.ylabel('Density')
    plt.show()
```

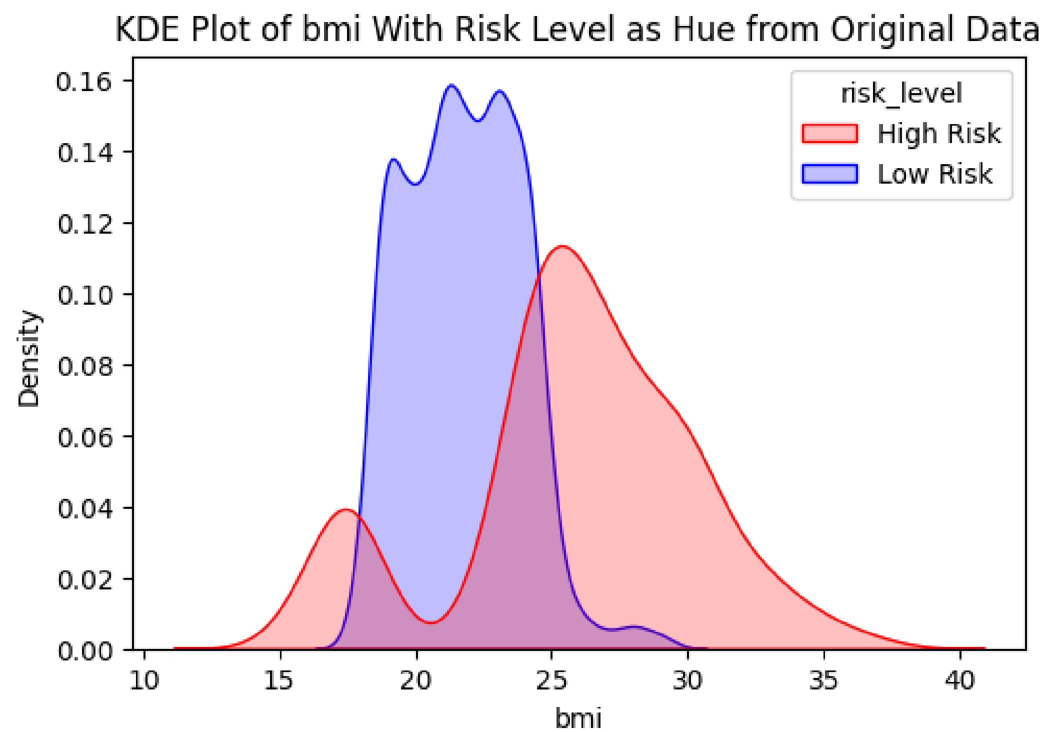


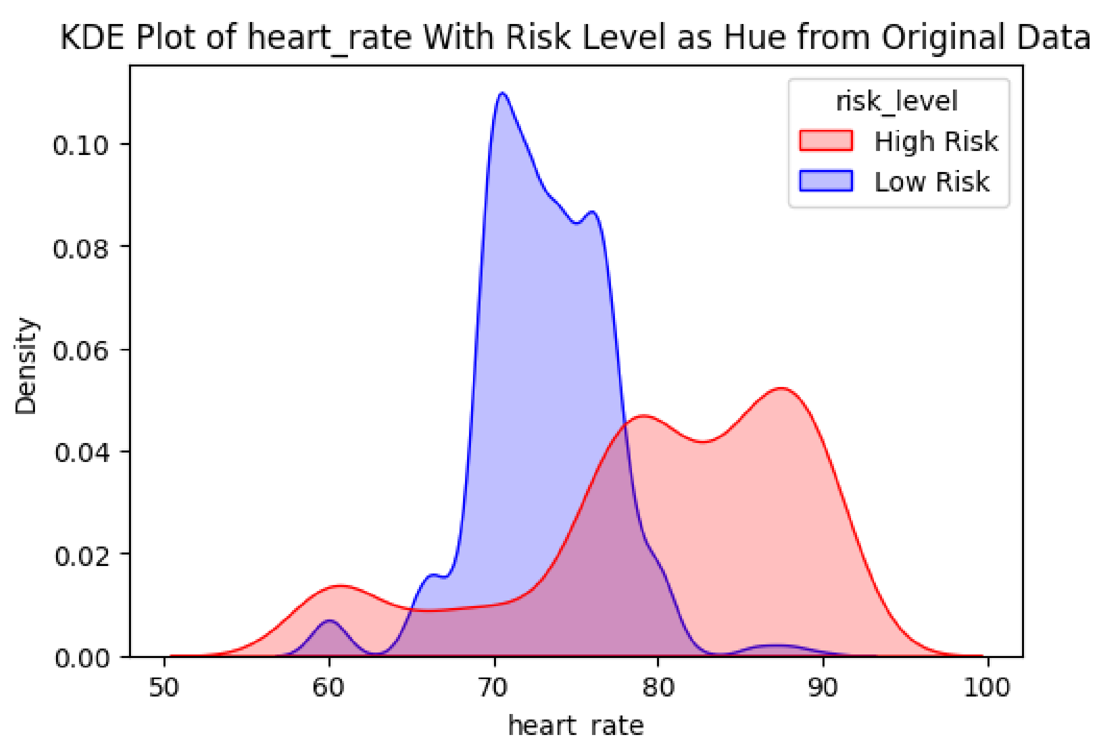

KDE Plot of systolic_bp With Risk Level as Hue from Original Data
risk_level
High Risk
Low Risk
Density
0.000
0.005
0.010
0.015
0.020
0.025
0.030
0.035
50
75
100
125
150
175
200
225
systolic_bp


KDE Plot of diastolic With Risk Level as Hue from Original Data
risk_level
High Risk
Low Risk
Density
0.000
0.005
0.010
0.015
0.020
0.025
0.030
0.035
20
40
60
80
100
120
140
160
diastolic


KDE Plot of bs With Risk Level as Hue from Original Data
risk_level
High Risk
Low Risk
Density
0.0
0.1
0.2
0.3
0.4
0
5
10
15
20
bs

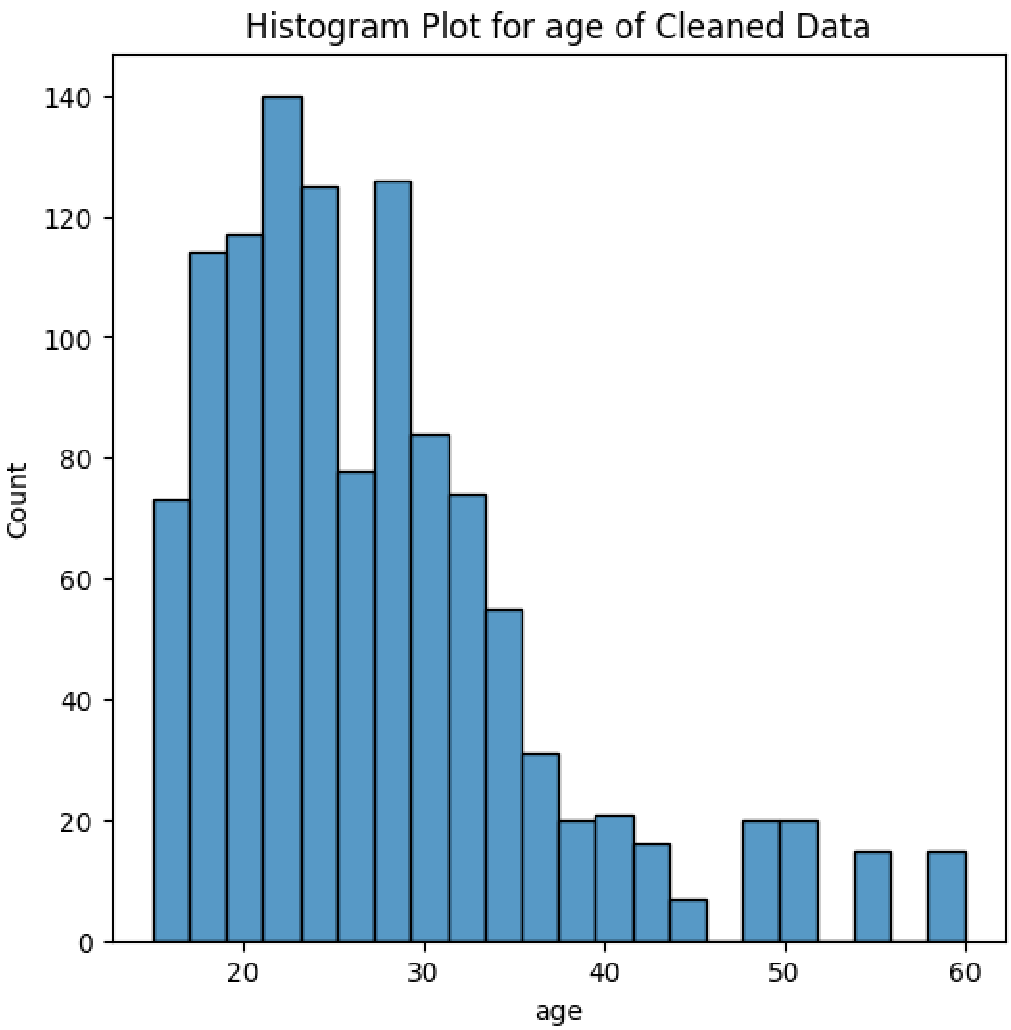

KDE Plot of body_temp With Risk Level as Hue from Original Data
risk_level
High Risk
Low Risk
Density
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
96
98
100
102
104
body_temp

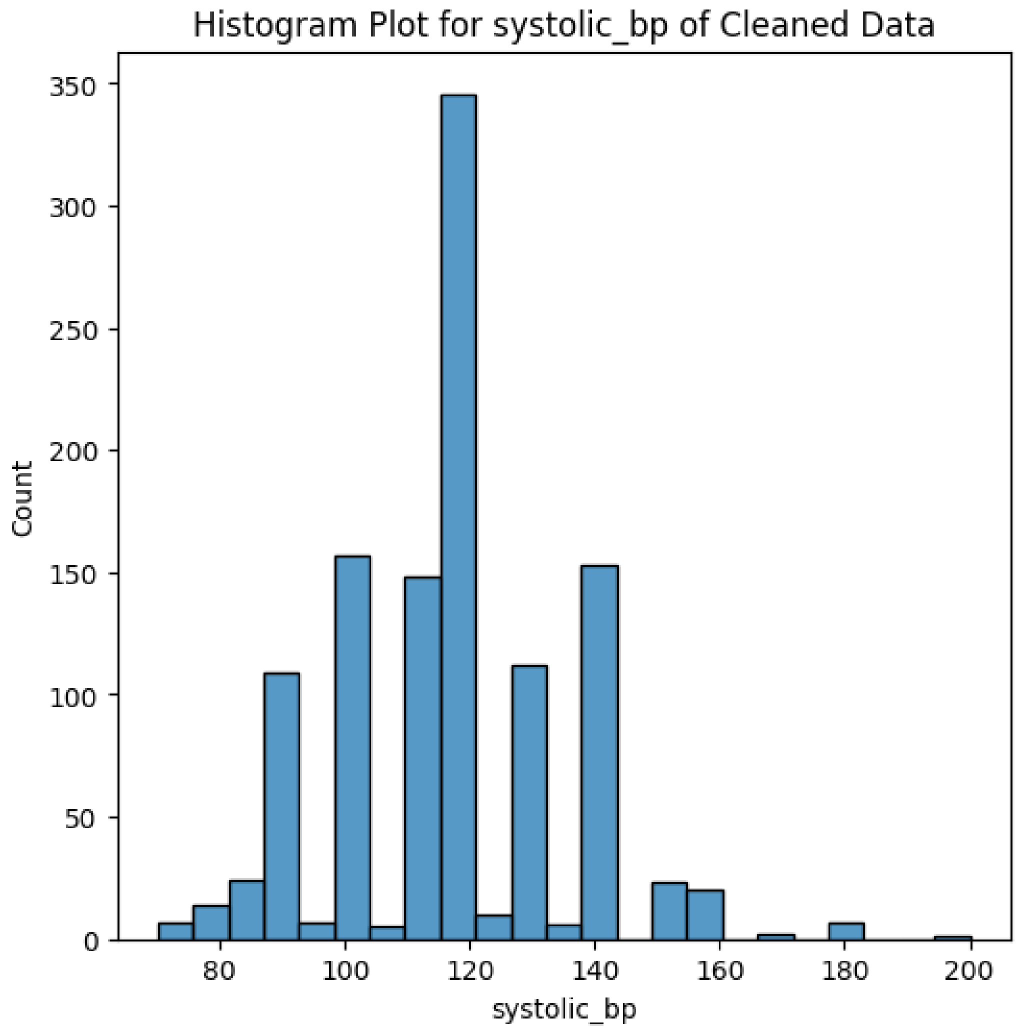

KDE Plot of bmi With Risk Level as Hue from Original Data
Density
bmi
risk_level
High Risk
Low Risk
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
0.16
10
15
20
25
30
35
40

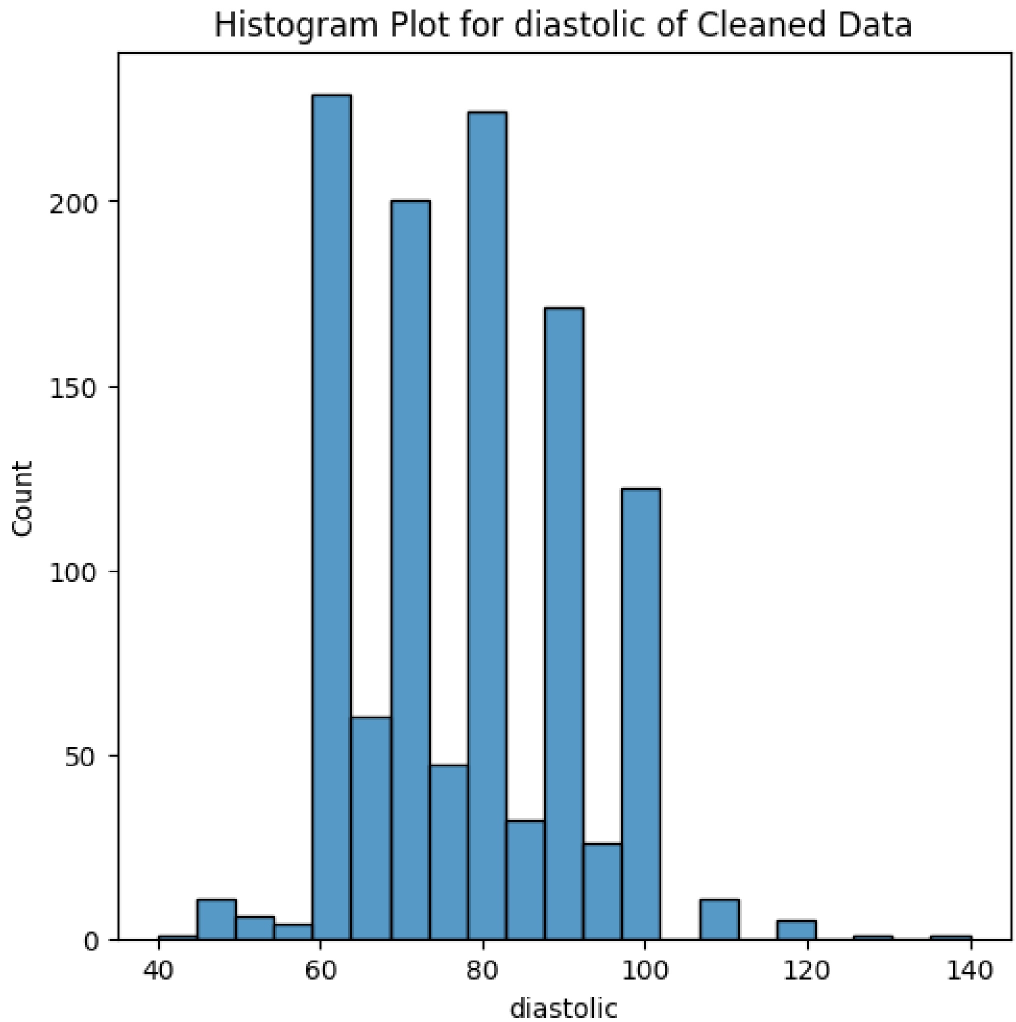

KDE Plot of heart_rate With Risk Level as Hue from Original Data
Density
heart_rate
risk_level
High Risk
Low Risk
0.00
0.02
0.04
0.06
0.08
0.10
50
60
70
80
90
100

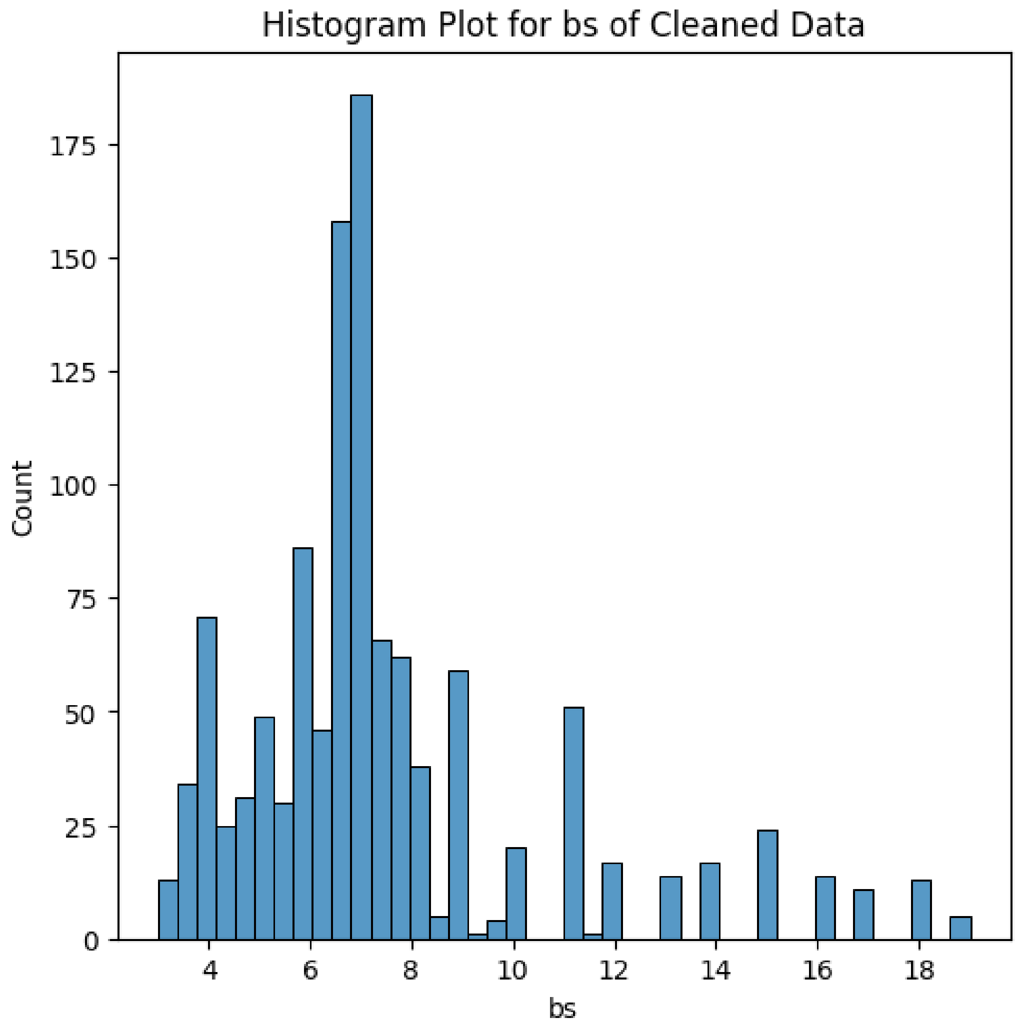
7 Histogram Plot

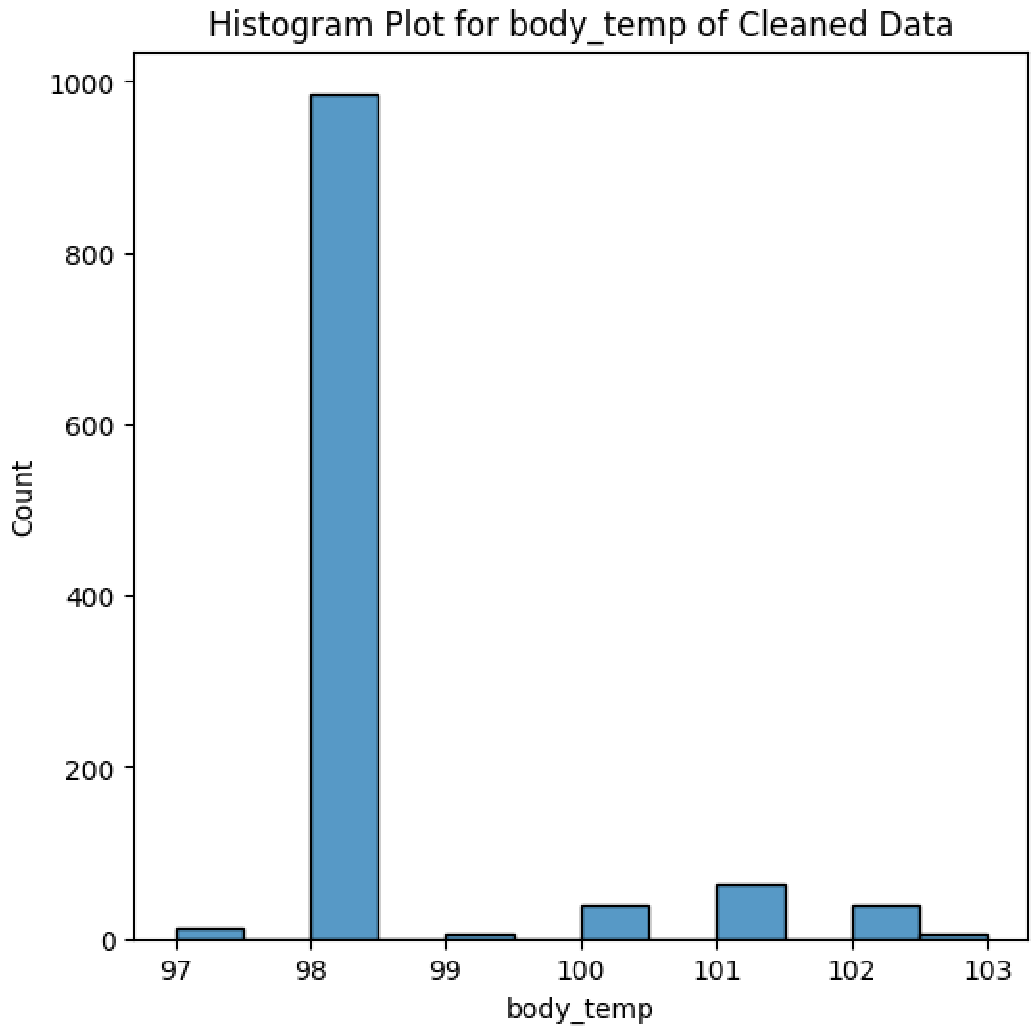
[58]:
```
#histogram for features (non-categorical) without hue
# List of columns(non-categorical) to plot
num_cols = ['age', 'systolic_bp', 'diastolic', 'bs', 'body_temp', 'bmi',␣
↪'heart_rate']

for i in num_cols:
    plt.figure(figsize=(6,6))
    sns.histplot(data=df_cln, x=i, kde=False)  # Histogram with KDE overlay
    plt.title(f'Histogram Plot for {i} of Cleaned Data')
    plt.show()
```

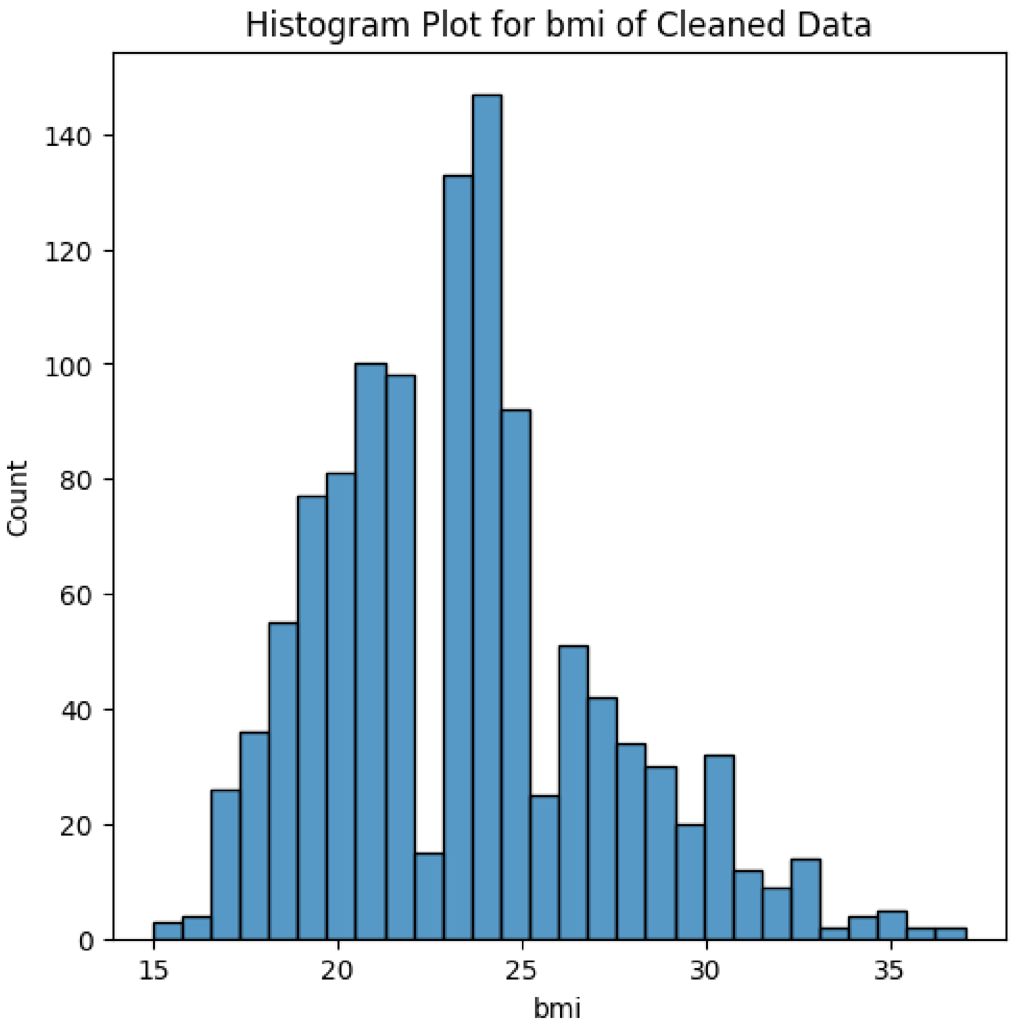



Histogram Plot for systolic_bp of Cleaned Data
Count
350
300
250
200
150
100
50
0
80
100
120
140
160
180
200
systolic_bp

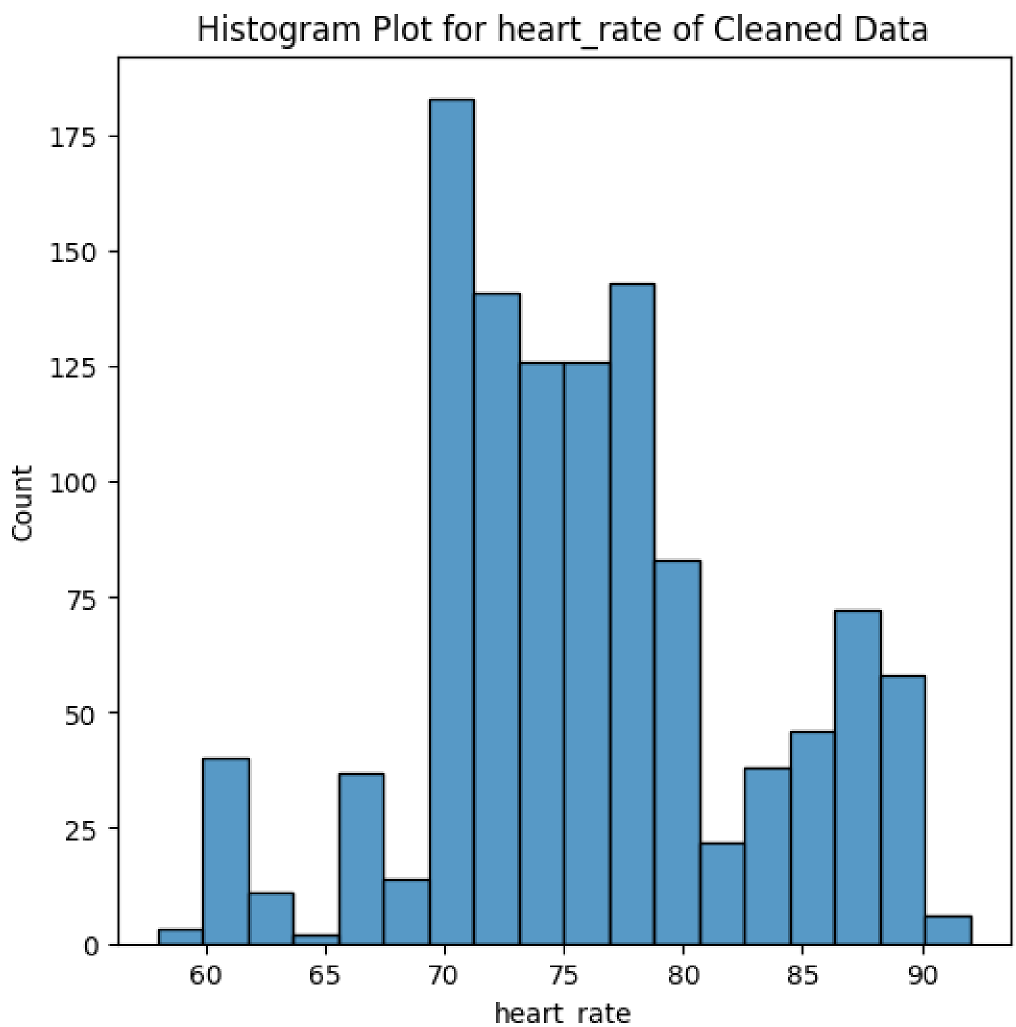

Histogram Plot for diastolic of Cleaned Data
Count
0
50
100
150
200
40
60
80
100
120
140
diastolic


Histogram Plot for bs of Cleaned Data
Count
bs
0
25
50
75
100
125
150
175
4
6
8
10
12
14
16
18


Histogram Plot for body_temp of Cleaned Data
Count
1000
800
600
400
200
0
97
98
99
100
101
102
103
body_temp

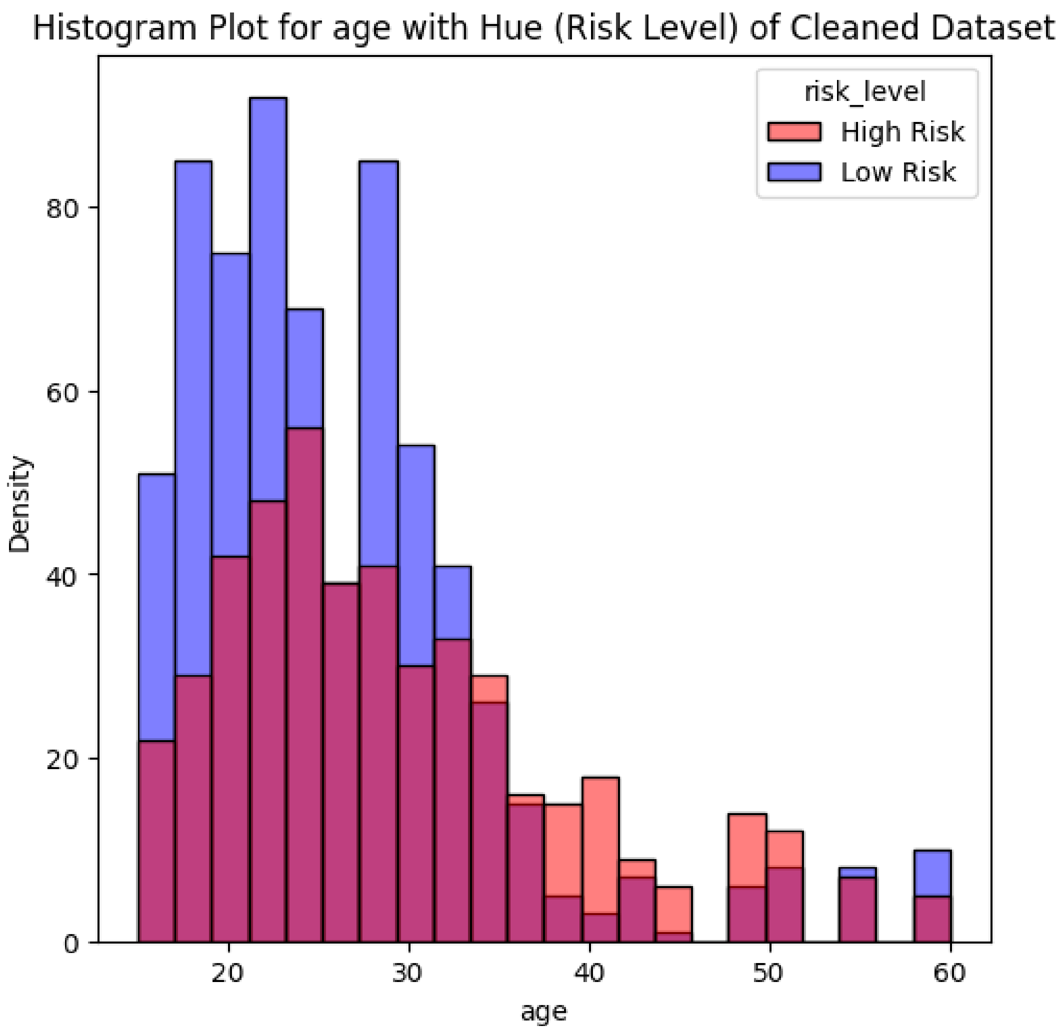


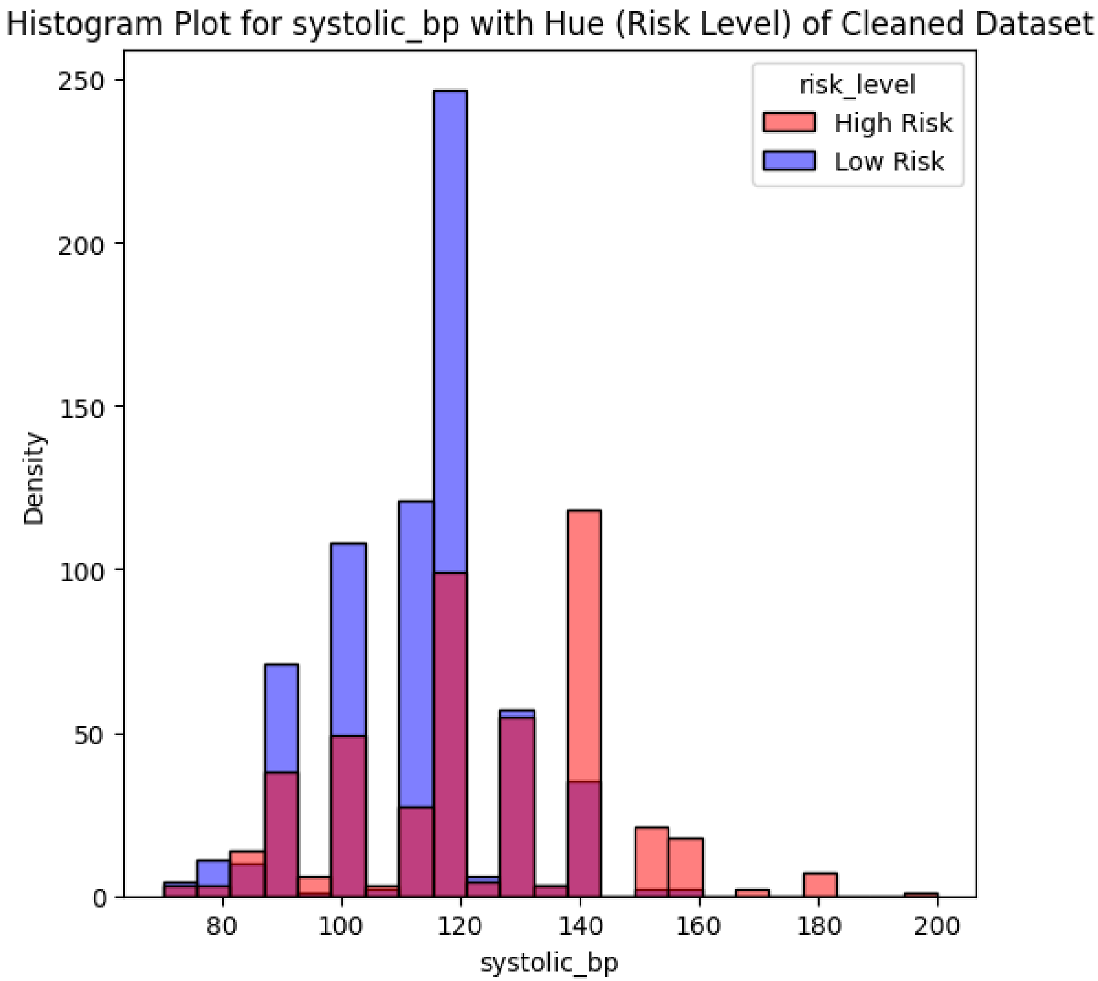
Histogram Plot for bmi of Cleaned Data
Count
bmi
0
20
40
60
80
100
120
140
15
20
25
30
35

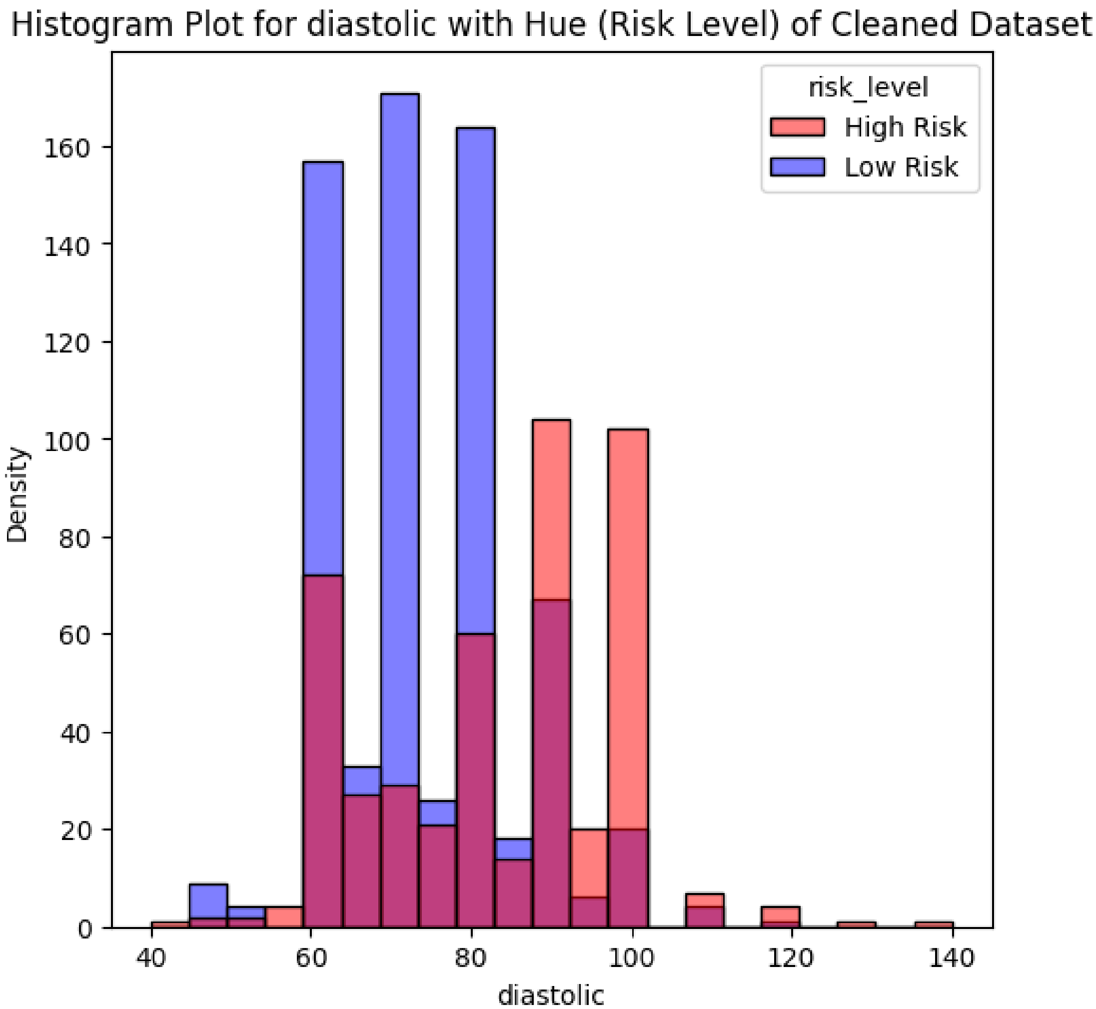


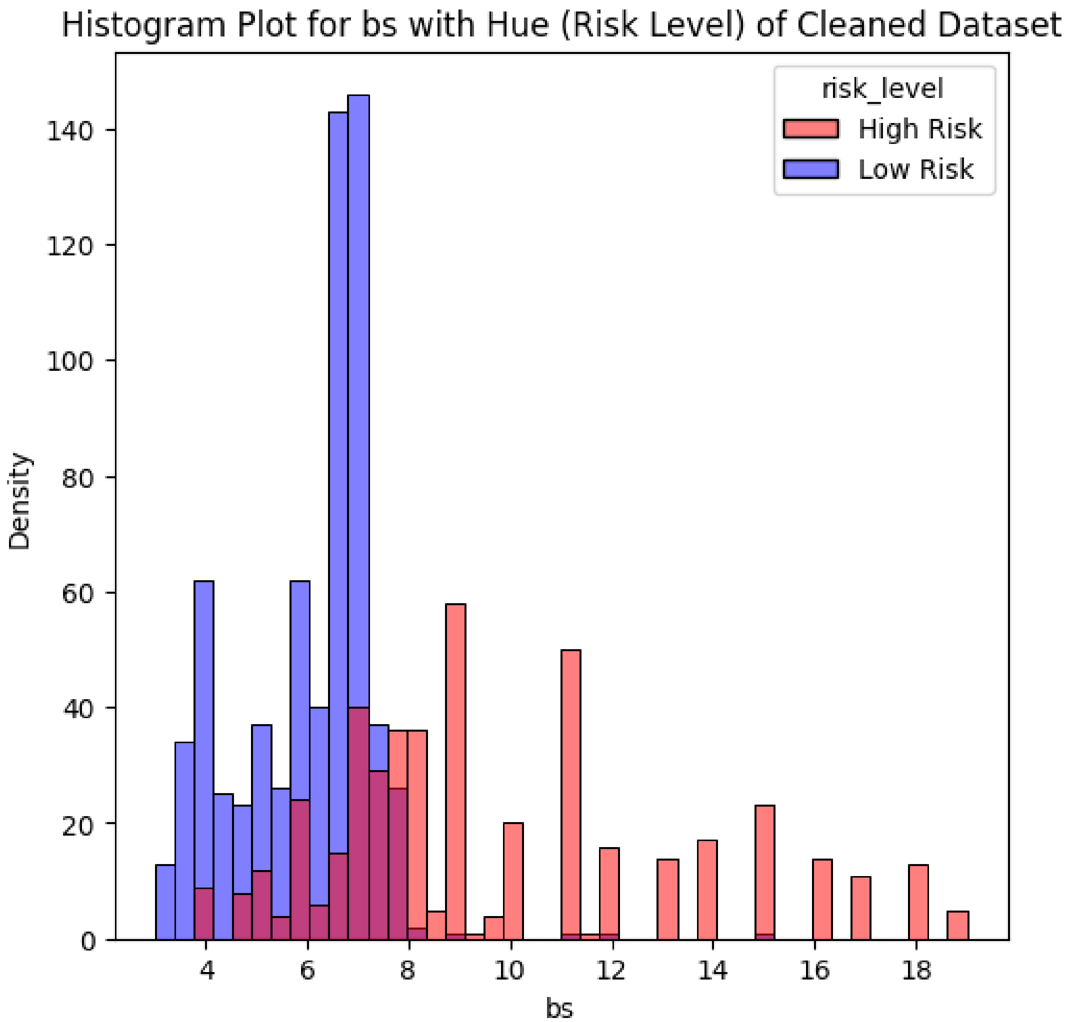

```
[59]: # Keep risk_level as numeric (do NOT map in the DataFrame)
label_map = {0: "Low Risk", 1: "High Risk"}
col_palette = {'Low Risk': 'blue', 'High Risk': 'red'}

# List of numeric columns to plot
num_cols = ['age', 'systolic_bp', 'diastolic', 'bs', 'body_temp', 'bmi',␣
 ↪'heart_rate']

# Plot histograms with mapped labels just for hue
for col in num_cols:
    plt.figure(figsize=(6, 6))
    sns.histplot(
        data=df_cln,
        x=col,
        hue=df_cln['risk_level'].map(label_map),  # map only for hue
```

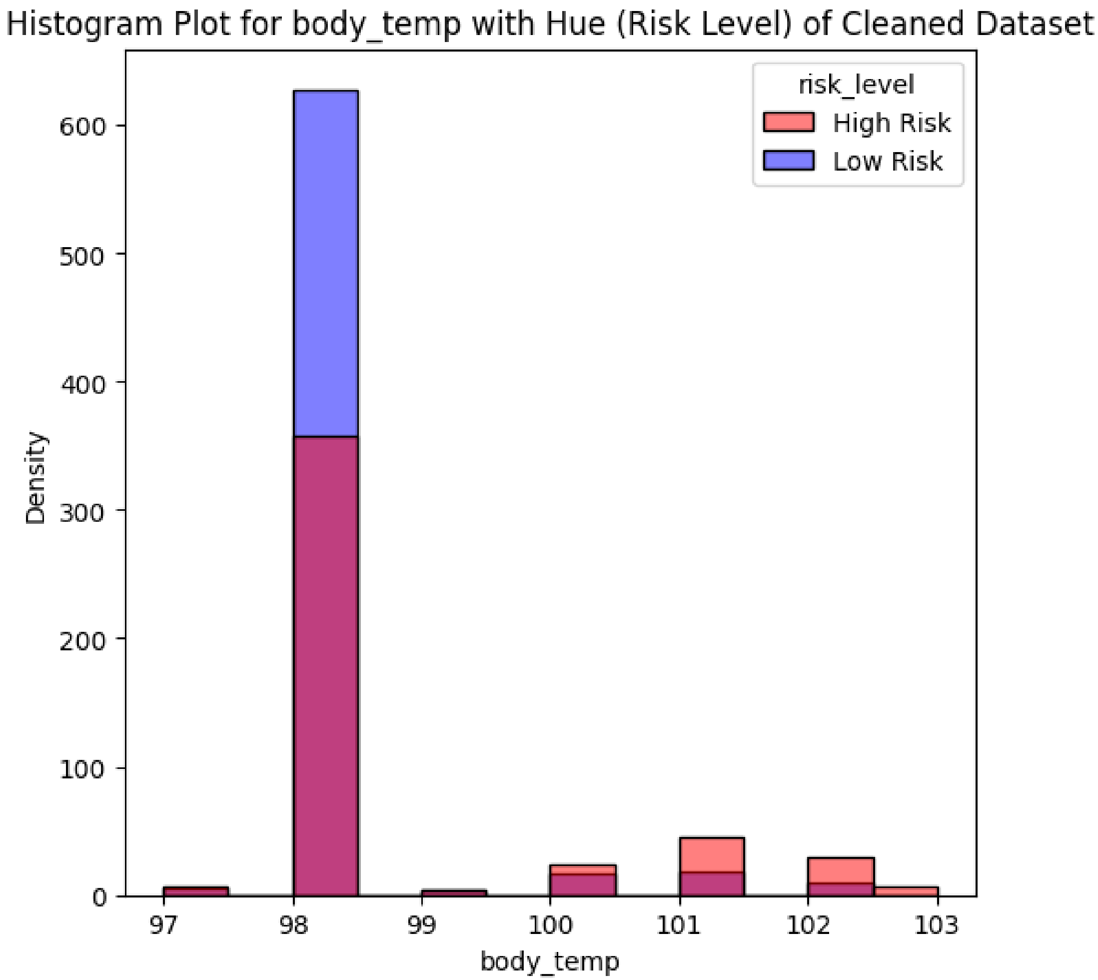
```
        kde=False,
        palette=col_palette
    )
    plt.title(f'Histogram Plot for {col} with Hue (Risk Level) of Cleaned␣
↪Dataset')
    plt.xlabel(col)
    plt.ylabel('Density')
    plt.show()
```



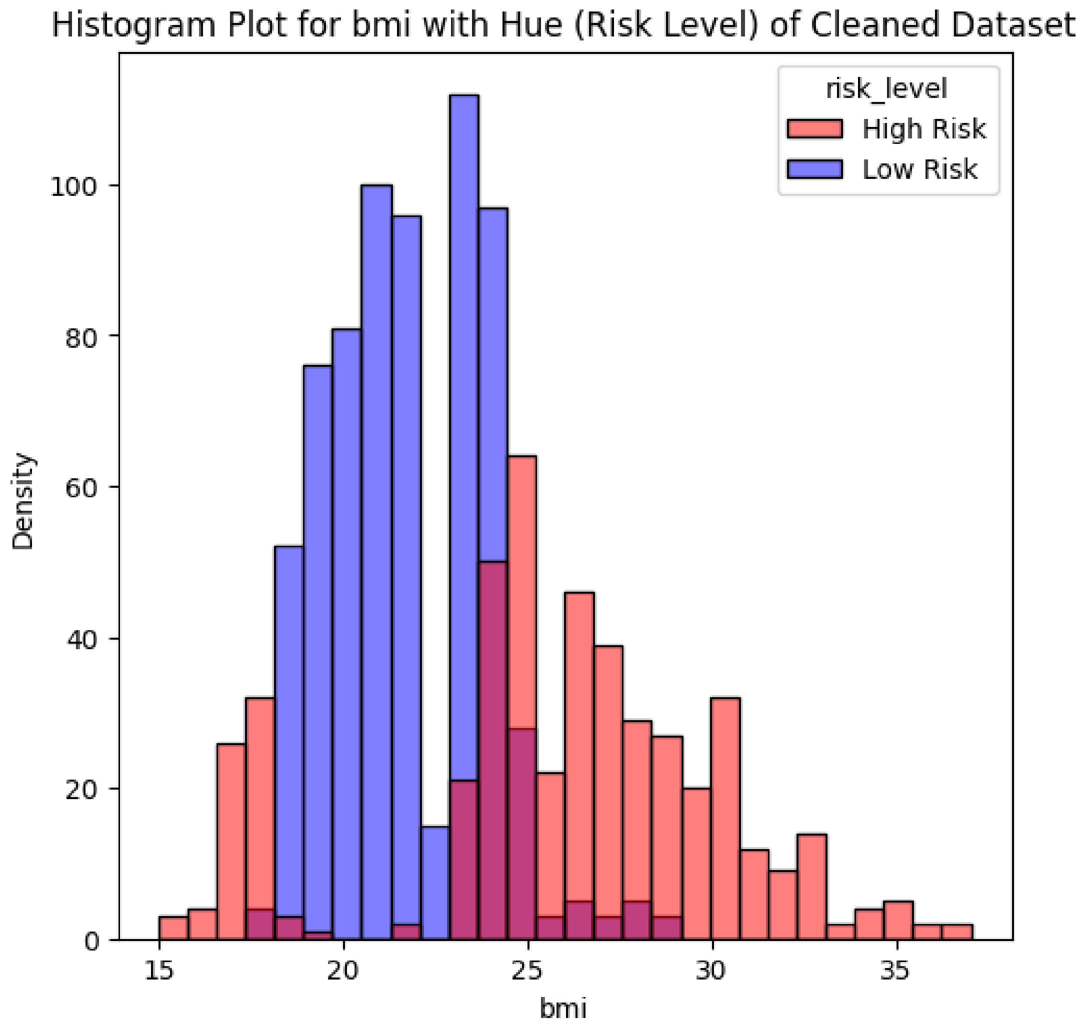

Histogram Plot for systolic_bp with Hue (Risk Level) of Cleaned Dataset
risk_level
High Risk
Low Risk
Density
systolic_bp
0
50
100
150
200
250
80
100
120
140
160
180
200

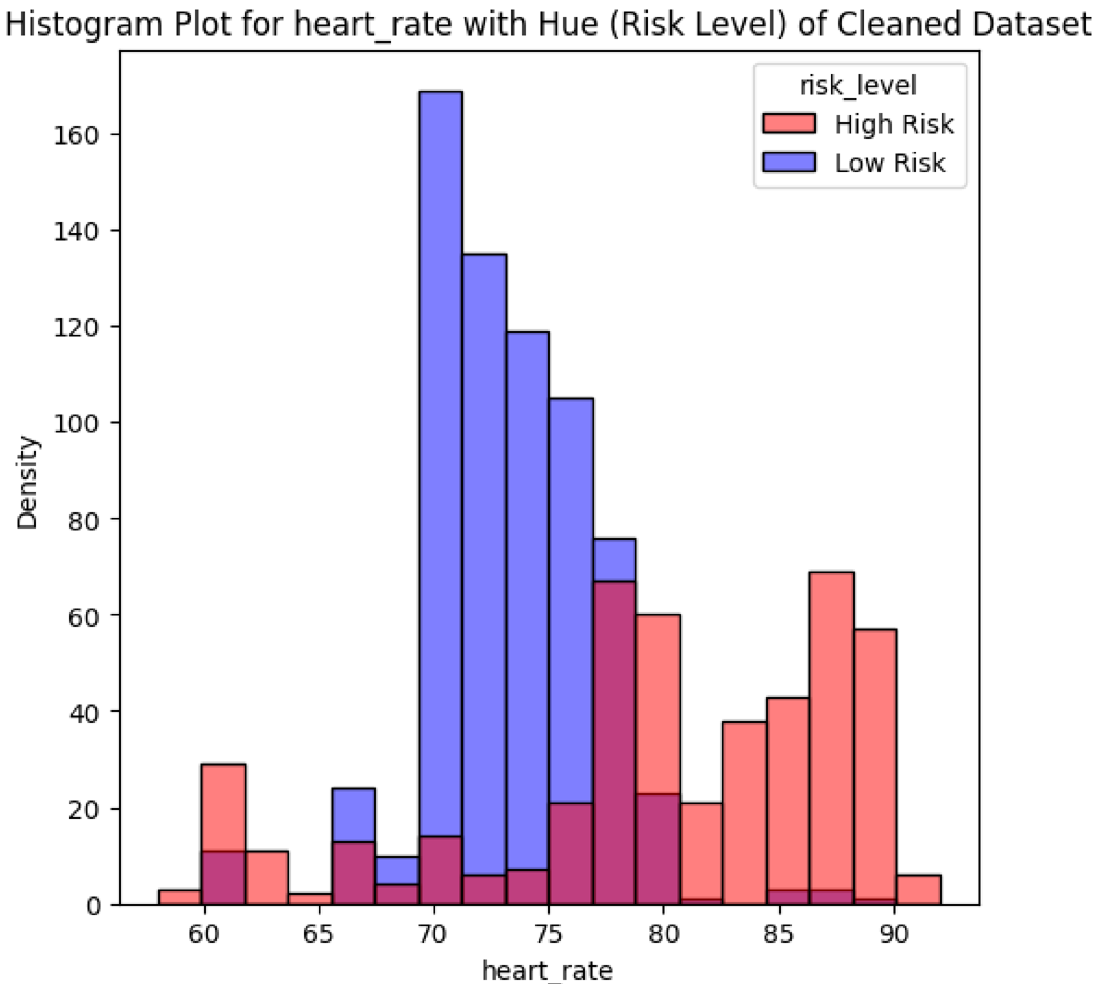

Histogram Plot for diastolic with Hue (Risk Level) of Cleaned Dataset
risk_level
High Risk
Low Risk
Density
diastolic
0
20
40
60
80
100
120
140
160
40
60
80
100
120
140


Histogram Plot for bs with Hue (Risk Level) of Cleaned Dataset
risk_level
High Risk
Low Risk
Density
bs
0
20
40
60
80
100
120
140
4
6
8
10
12
14
16
18



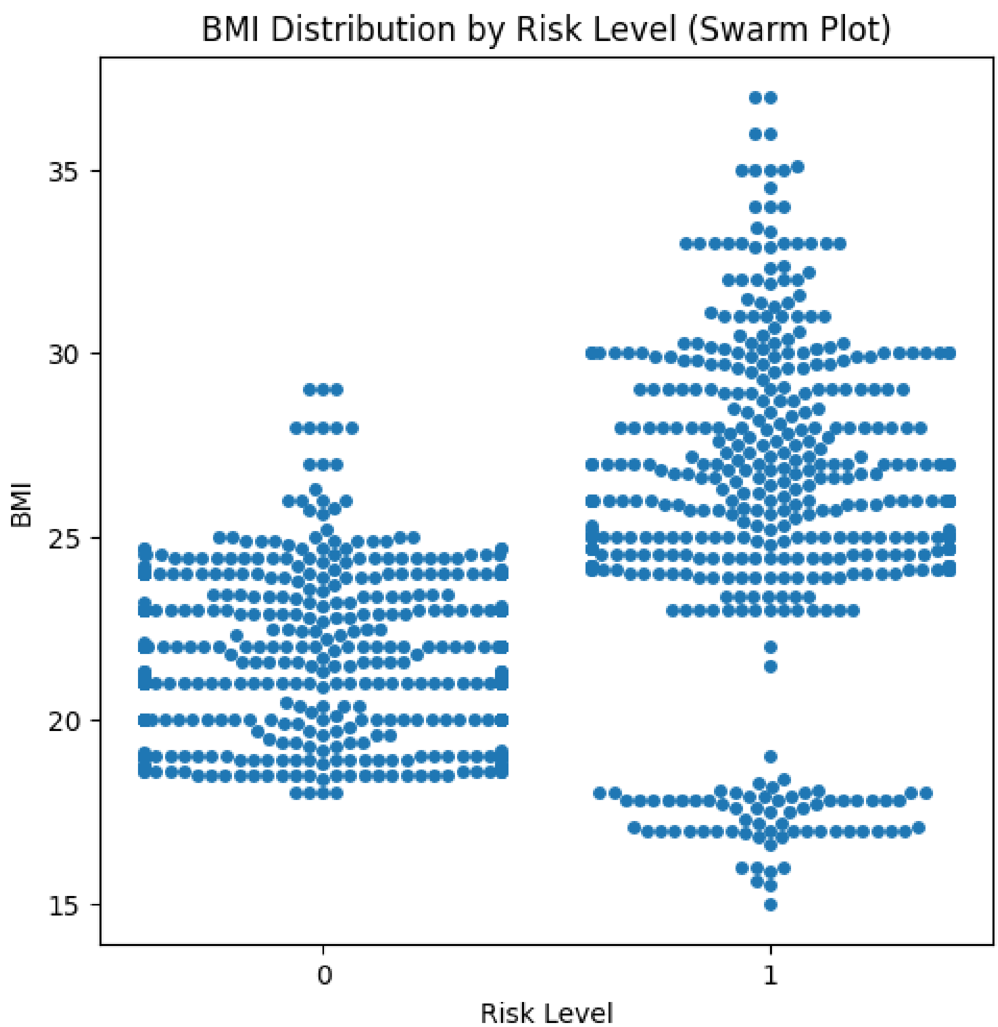
Histogram Plot for body_temp with Hue (Risk Level) of Cleaned Dataset
risk_level
High Risk
Low Risk
Density
600
500
400
300
200
100
0
97
98
99
100
101
102
103
body_temp


Histogram Plot for bmi with Hue (Risk Level) of Cleaned Dataset
risk_level
High Risk
Low Risk
Density
bmi
0
20
40
60
80
100
15
20
25
30
35

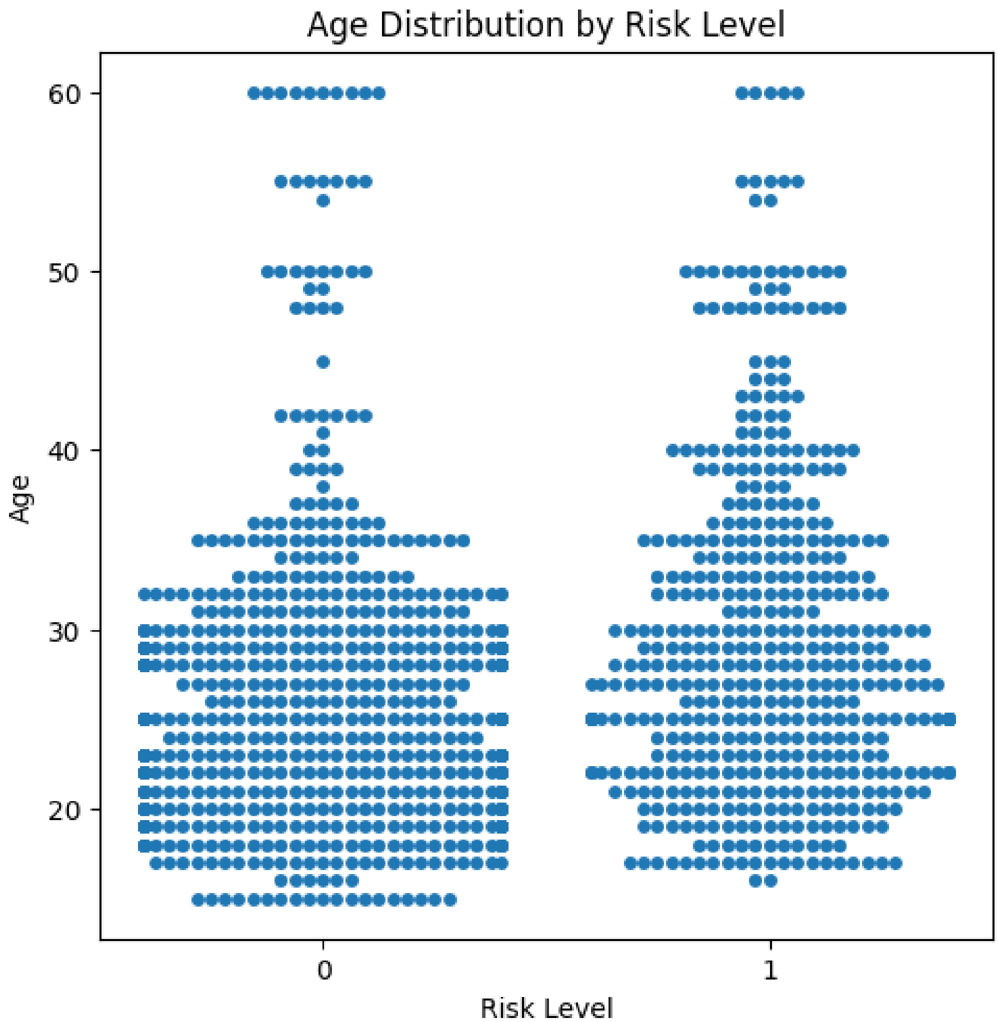


8 Swarm plot

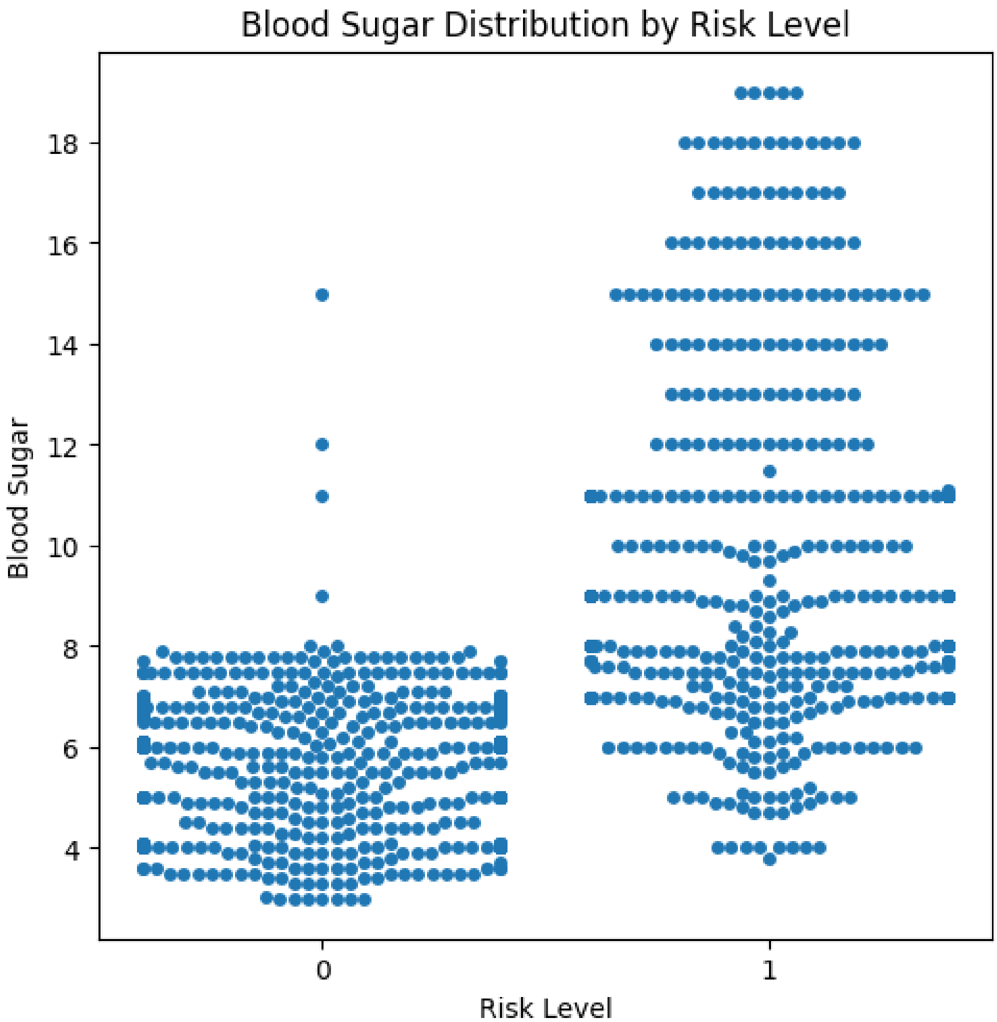
[60]:
```
plt.figure(figsize=(6, 6))
sns.swarmplot(data=df_cln, x='risk_level', y='bmi')
plt.title('BMI Distribution by Risk Level (Swarm Plot)')
plt.xlabel('Risk Level')
plt.ylabel('BMI')
plt.show()
```



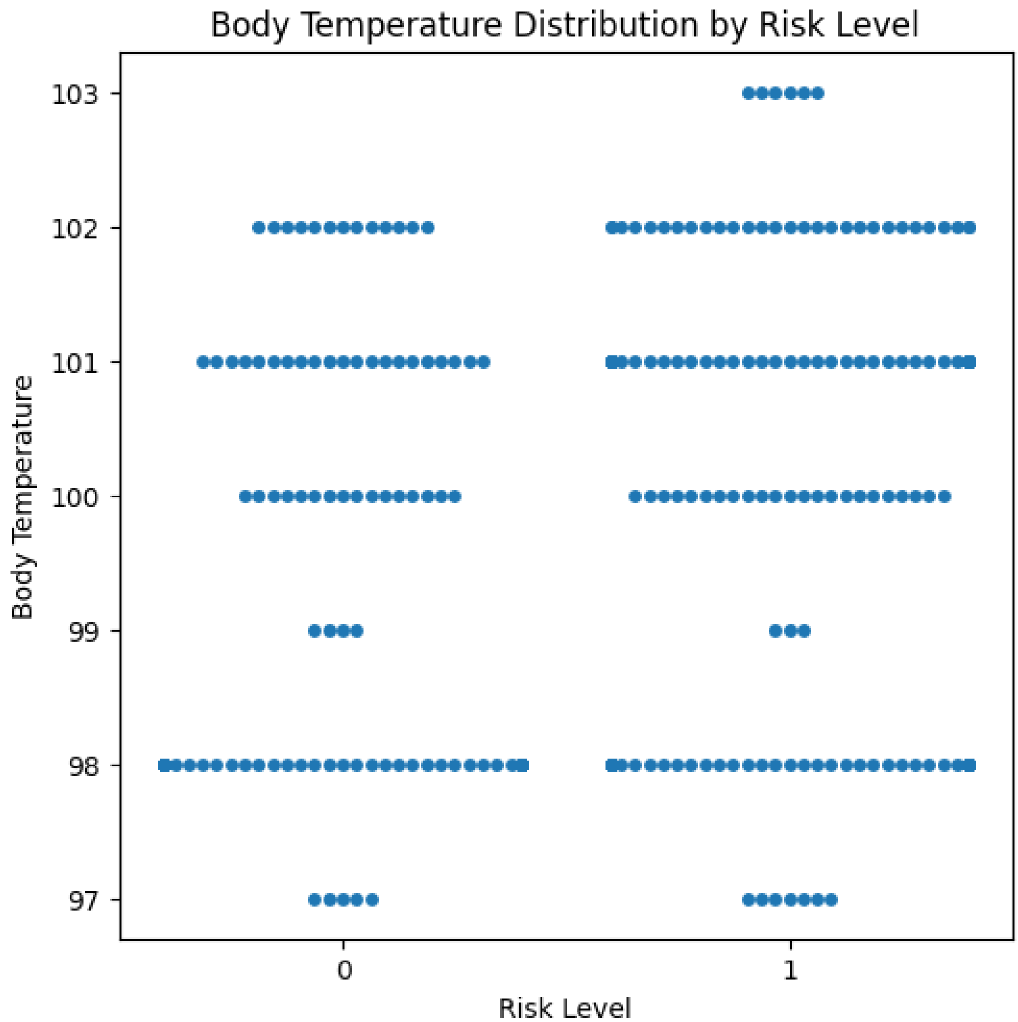
[61]:
```
plt.figure(figsize=(6, 6))
sns.swarmplot(data=df_cln, x='risk_level', y='age')
plt.title('Age Distribution by Risk Level')
plt.xlabel('Risk Level')
plt.ylabel('Age')
plt.show()
```



[62]:
```
plt.figure(figsize=(6, 6))
sns.swarmplot(data=df_cln, x='risk_level', y='bs')
plt.title('Blood Sugar Distribution by Risk Level')
plt.xlabel('Risk Level')
plt.ylabel('Blood Sugar')
plt.show()
```

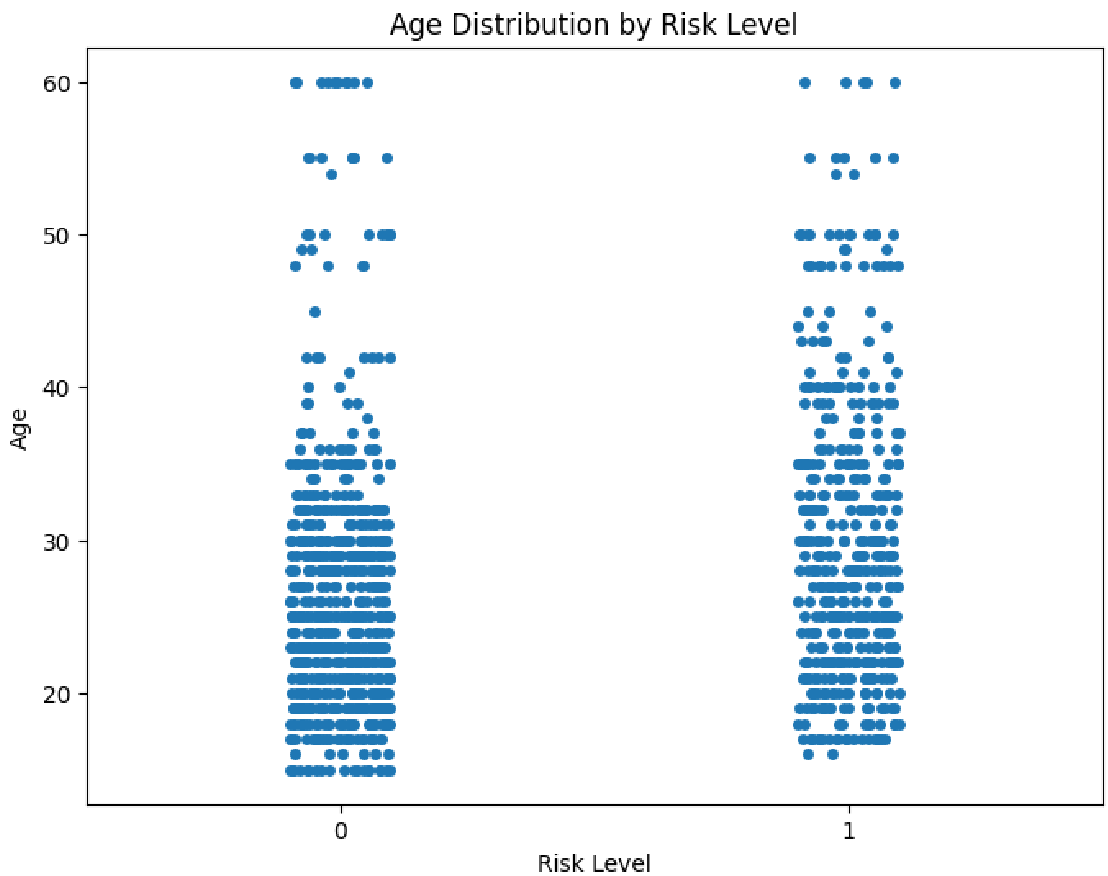


[63]:
```
plt.figure(figsize=(6, 6))
sns.swarmplot(data=df, x='risk_level', y='body_temp')
plt.title('Body Temperature Distribution by Risk Level')
plt.xlabel('Risk Level')
plt.ylabel('Body Temperature')
plt.show()
```

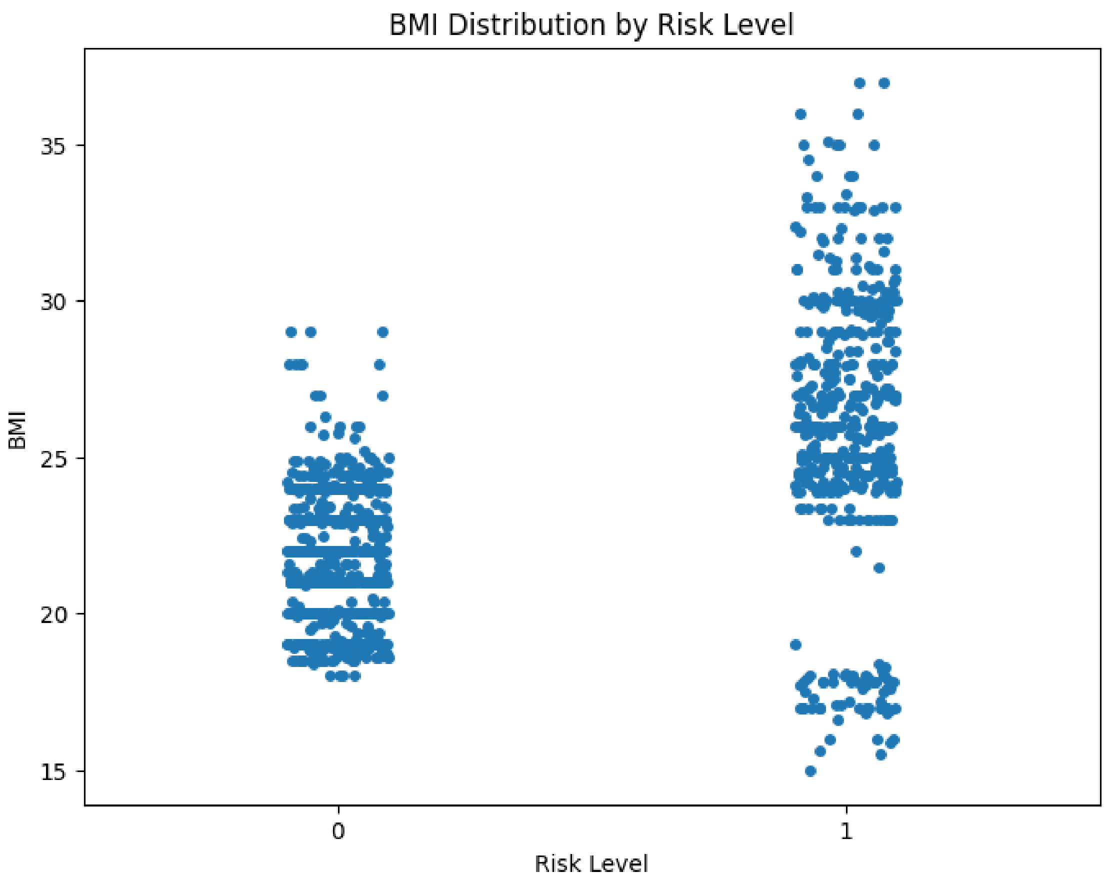


9 Strip Plot

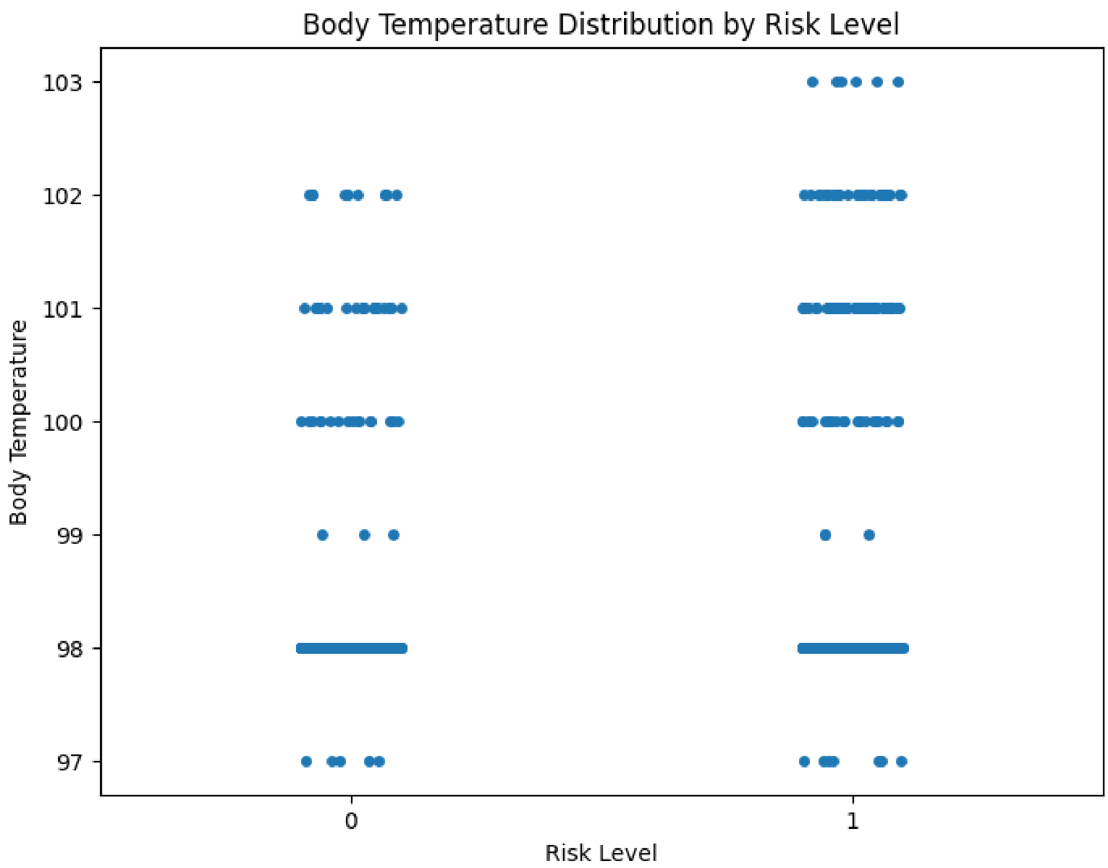
[64]:
```
plt.figure(figsize=(8, 6))
sns.stripplot(data=df_cln, x='risk_level', y='age', jitter=True)
plt.title('Age Distribution by Risk Level')
plt.xlabel('Risk Level')
plt.ylabel('Age')
plt.show()
```



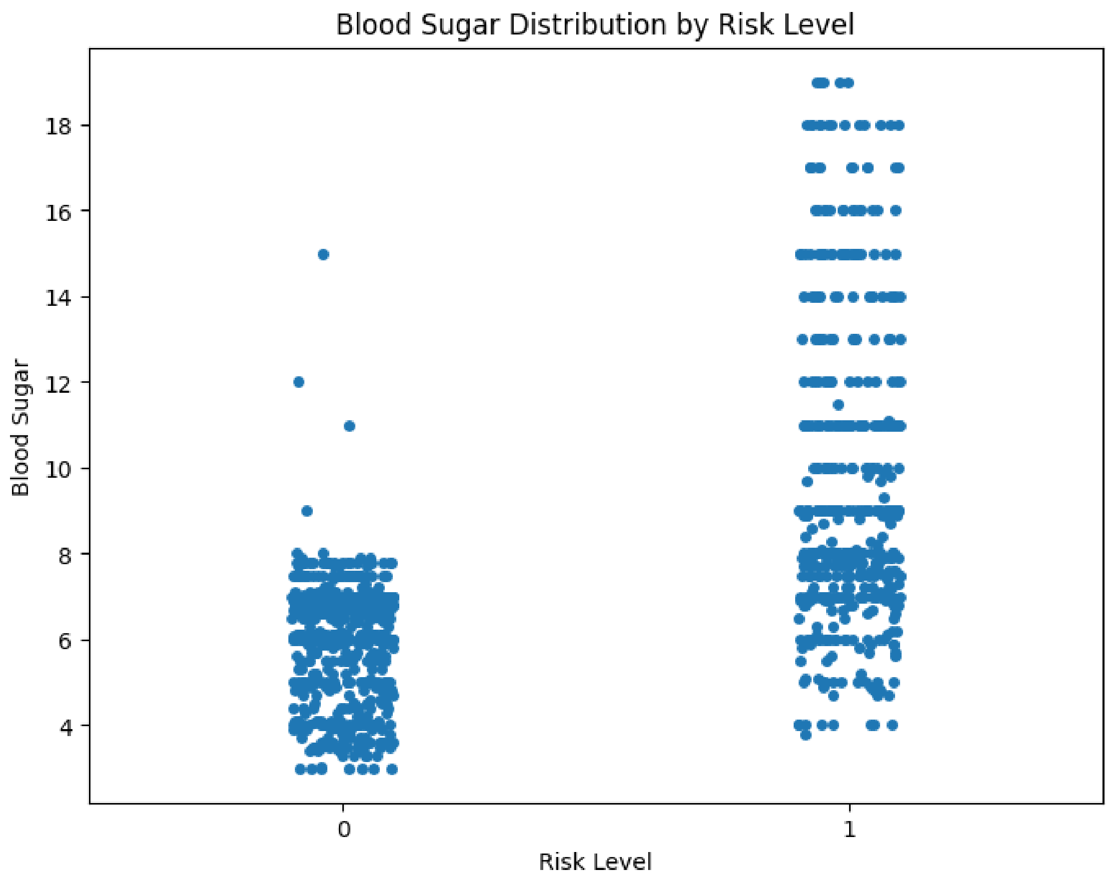
[65]:
```
plt.figure(figsize=(8, 6))
sns.stripplot(data=df_cln, x='risk_level', y='bmi', jitter=True)
plt.title('BMI Distribution by Risk Level')
plt.xlabel('Risk Level')
plt.ylabel('BMI')
plt.show()
```



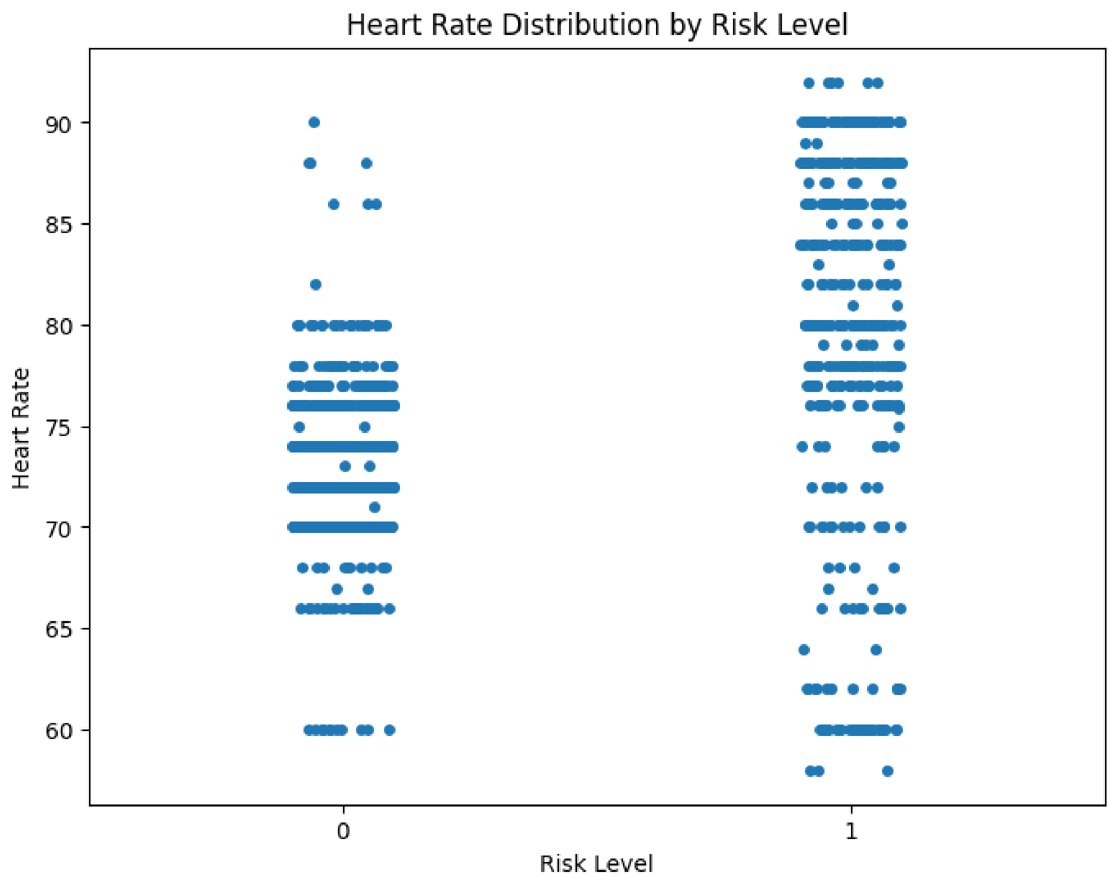
[66]:
```python
plt.figure(figsize=(8, 6))
sns.stripplot(data=df_cln, x='risk_level', y='body_temp', jitter=True)
plt.title('Body Temperature Distribution by Risk Level')
plt.xlabel('Risk Level')
plt.ylabel('Body Temperature')
plt.show()
```



[67]:
```
plt.figure(figsize=(8, 6))
sns.stripplot(data=df_cln, x='risk_level', y='bs', jitter=True)
plt.title('Blood Sugar Distribution by Risk Level')
plt.xlabel('Risk Level')
plt.ylabel('Blood Sugar')
plt.show()
```

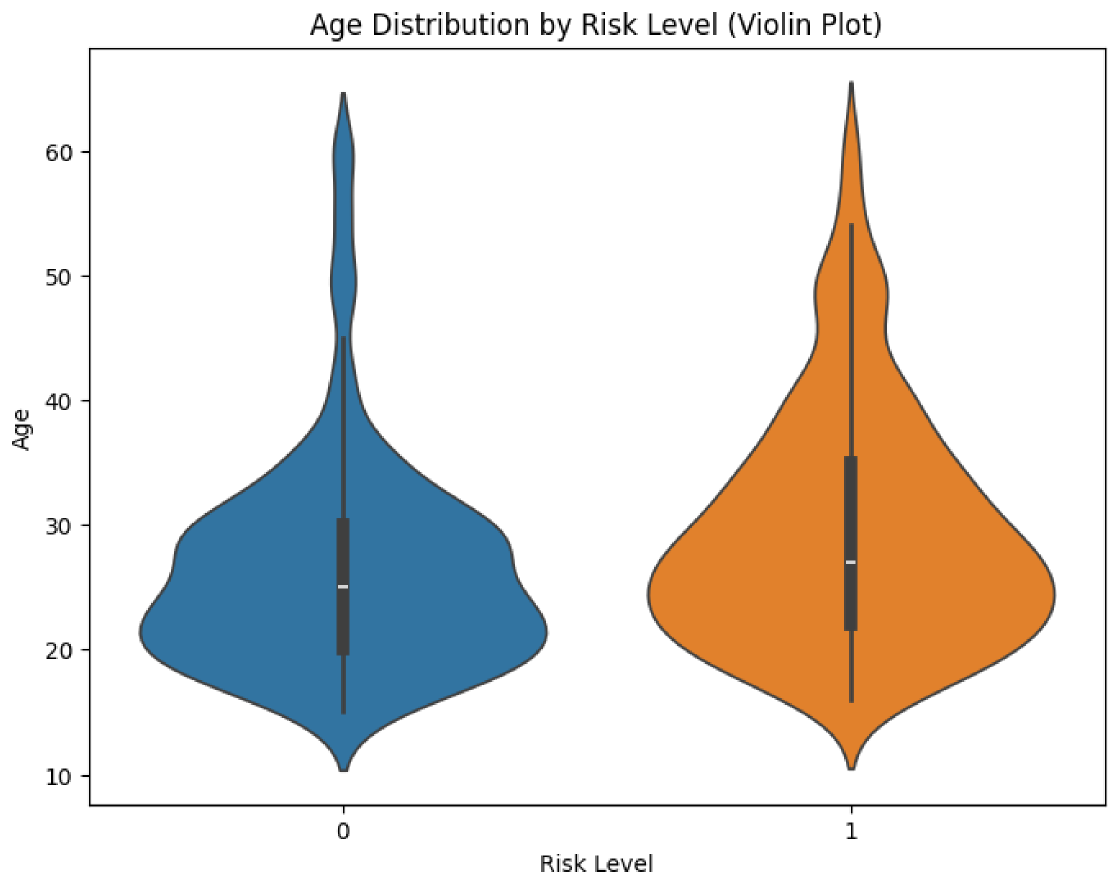


[68]:
```
plt.figure(figsize=(8, 6))
sns.stripplot(data=df_cln, x='risk_level', y='heart_rate', jitter=True)
plt.title('Heart Rate Distribution by Risk Level')
plt.xlabel('Risk Level')
plt.ylabel('Heart Rate')
plt.show()
```

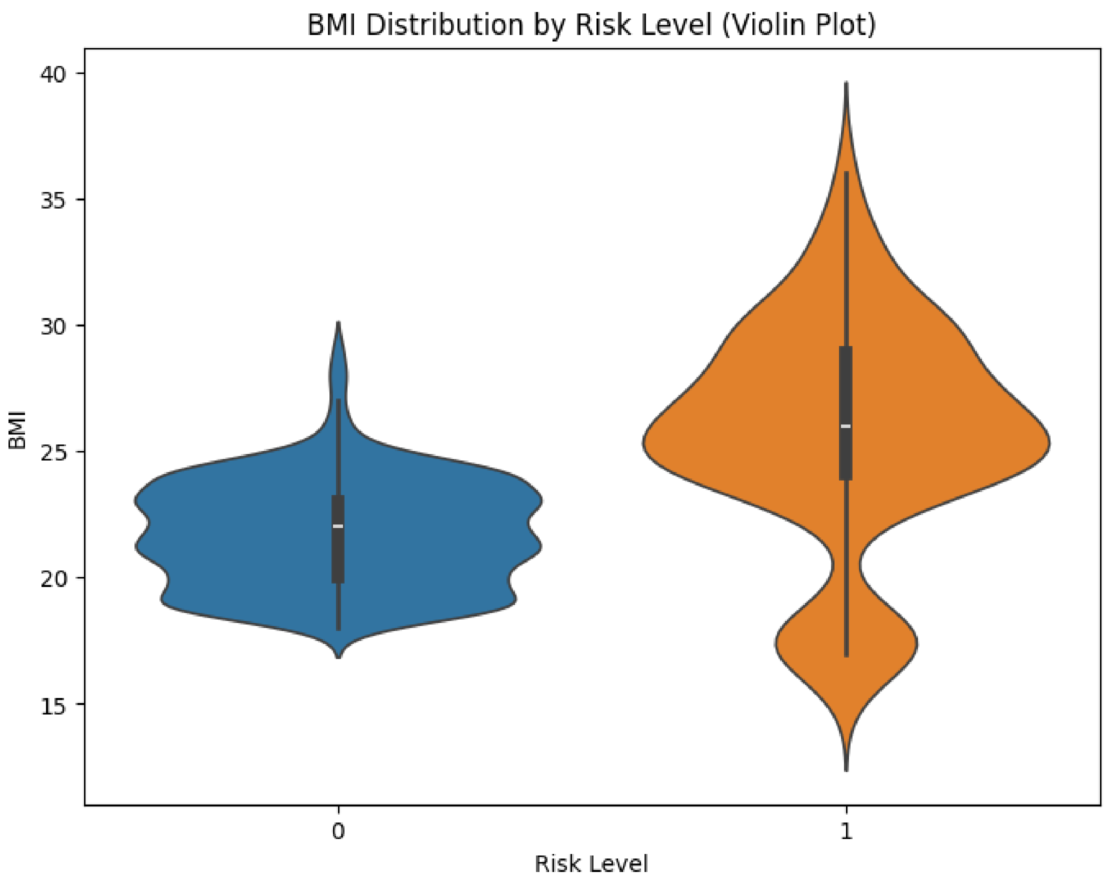


10 Violin Plot

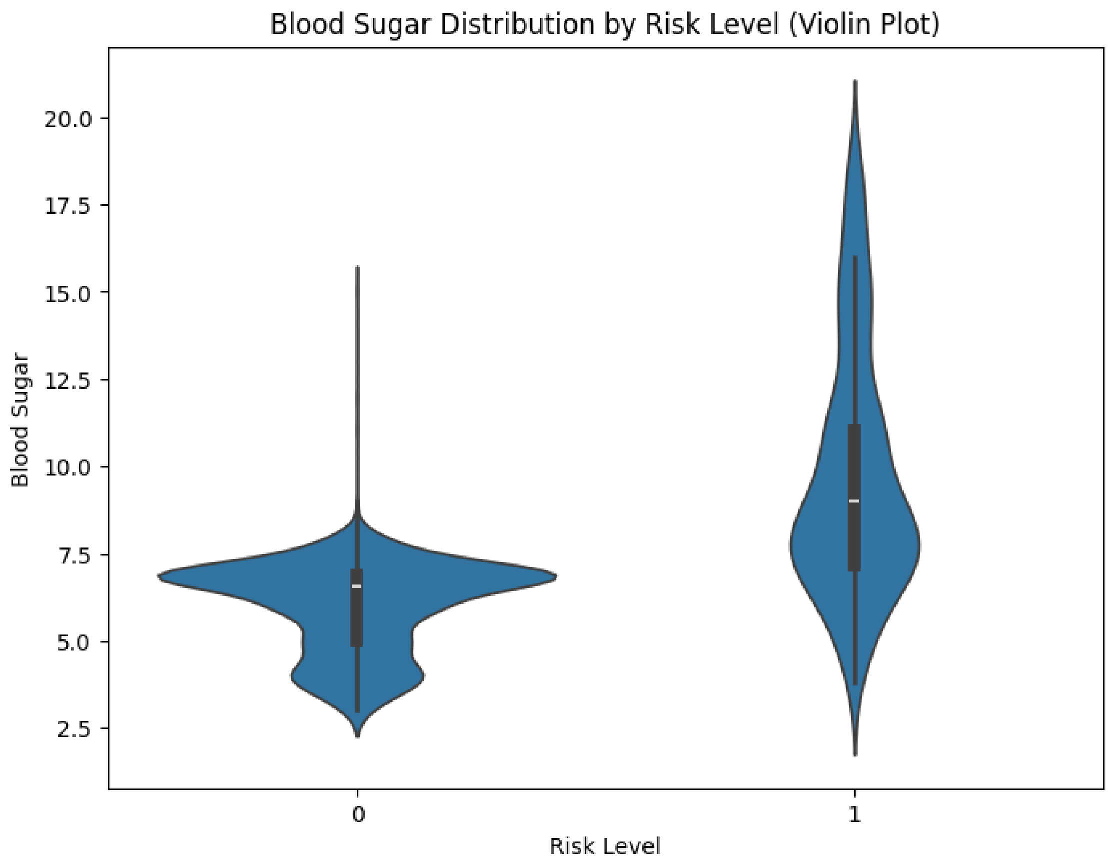
[69]:
```
plt.figure(figsize=(8, 6))
sns.violinplot(data=df_cln, x='risk_level', y='age', palette=['#1f77b4',␣
↪'#ff7f0e', '#2ca02c'])  # Custom colors
plt.title('Age Distribution by Risk Level (Violin Plot)')
plt.xlabel('Risk Level')
plt.ylabel('Age')
plt.show()
```



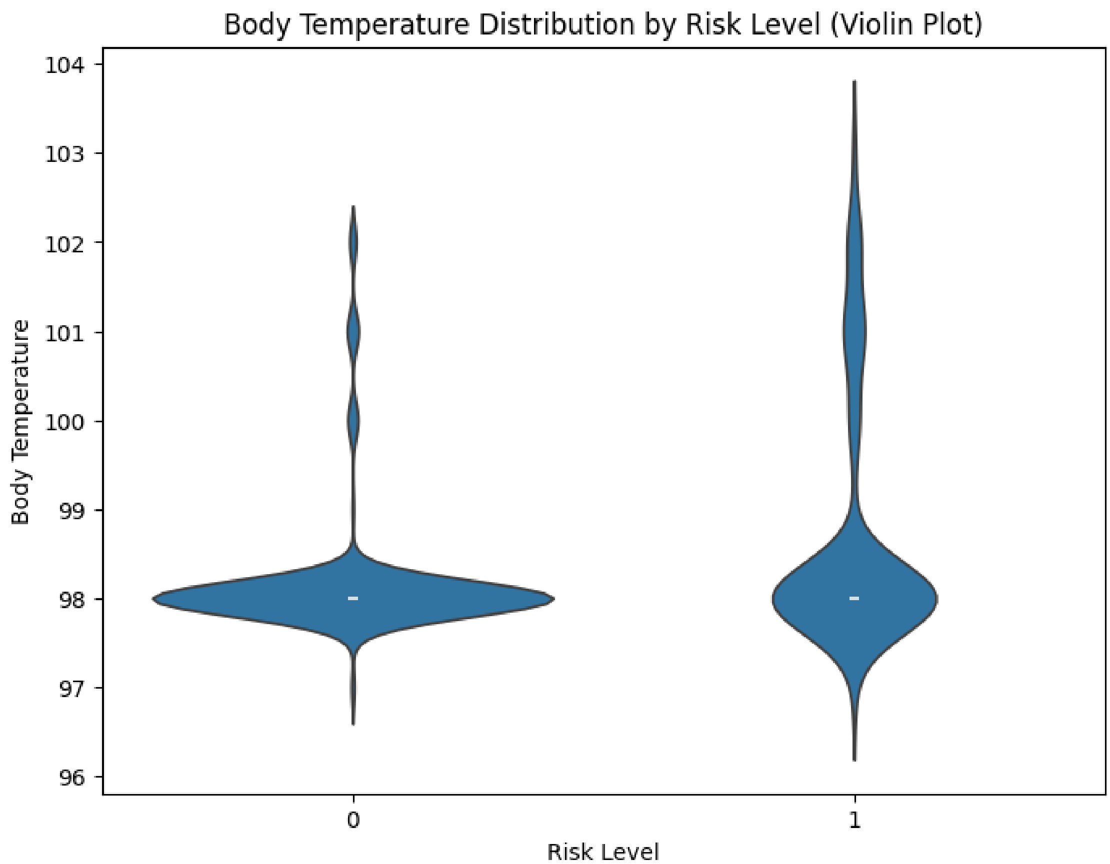
[70]:
```
plt.figure(figsize=(8, 6))
sns.violinplot(data=df_cln, x='risk_level', y='bmi', palette=['#1f77b4',␣
↪'#ff7f0e', '#2ca02c'])  # Custom colors )
plt.title('BMI Distribution by Risk Level (Violin Plot)')
plt.xlabel('Risk Level')
plt.ylabel('BMI')
plt.show()
```



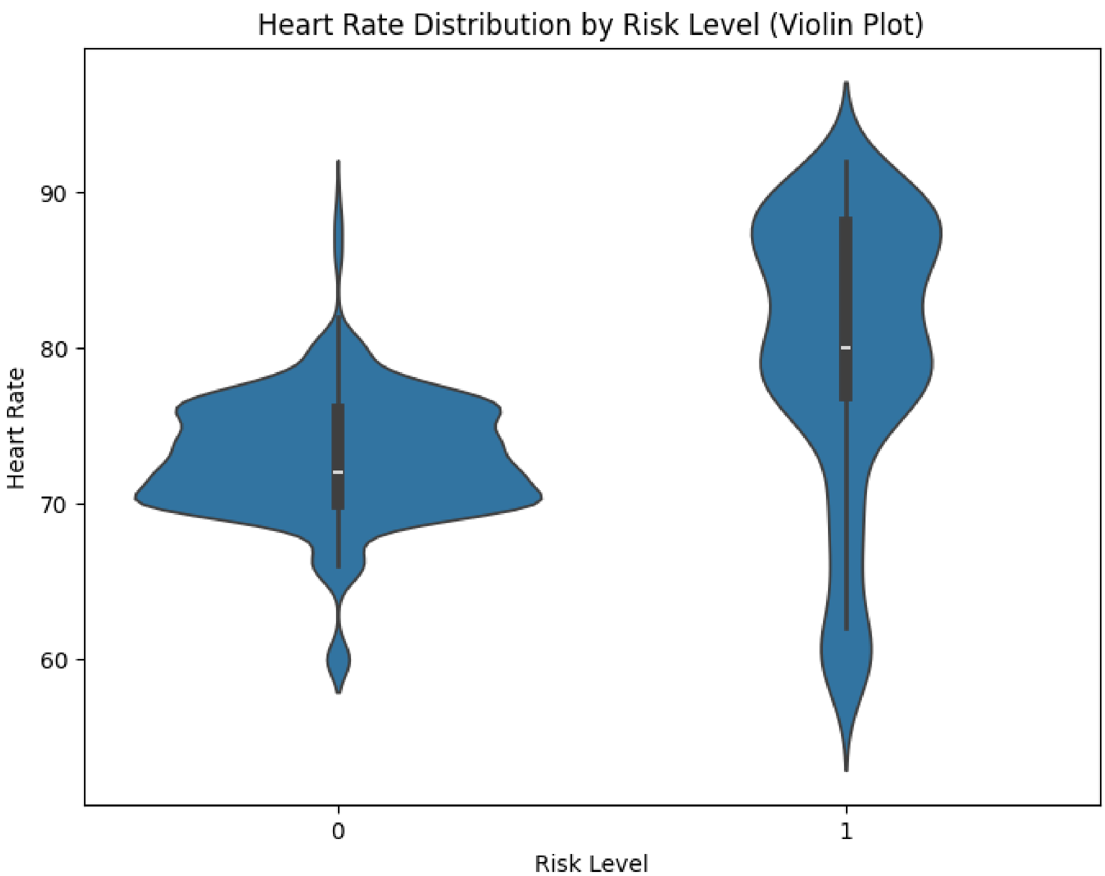
[71]:
```
plt.figure(figsize=(8, 6))
sns.violinplot(data=df_cln, x='risk_level', y='bs')
plt.title('Blood Sugar Distribution by Risk Level (Violin Plot)')
plt.xlabel('Risk Level')
plt.ylabel('Blood Sugar')
plt.show()
```



[72]:
```
plt.figure(figsize=(8, 6))
sns.violinplot(data=df_cln, x='risk_level', y='body_temp')
plt.title('Body Temperature Distribution by Risk Level (Violin Plot)')
plt.xlabel('Risk Level')
plt.ylabel('Body Temperature')
plt.show()
```



[73]:
```
plt.figure(figsize=(8, 6))
sns.violinplot(data=df_cln, x='risk_level', y='heart_rate')
plt.title('Heart Rate Distribution by Risk Level (Violin Plot)')
plt.xlabel('Risk Level')
plt.ylabel('Heart Rate')
plt.show()
```



11 Feature Scaling - Min Max Scaler

[74]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

# 1. Split features and target
X = df_cln.drop('risk_level', axis=1)
y = df_cln['risk_level']

# 2. Split into train and test sets (e.g., 80-20 split)
X_train, X_test, y_train, y_test = train_test_split(X, y,stratify = y,␣
↪test_size=0.2, random_state=42)

# 3. Initialize and fit MinMaxScaler on training data only
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
```

[75]:
```python
# Transform test data using parameters from training data
X_test_scaled = scaler.transform(X_test)
```

12 Imbalanced Dataset Handling using SMOTE

[76]:
```python
from imblearn.over_sampling import SMOTE

# Initialize SMOTE
smote = SMOTE(random_state=42)

# Apply SMOTE to the scaled training data
X_train_smote, y_train_smote = smote.fit_resample(X_train_scaled, y_train)

# Check the class distribution before and after SMOTE
print("Before SMOTE:")
print(y_train.value_counts())
print("\nAfter SMOTE:")
print(pd.Series(y_train_smote).value_counts())
```

```
Before SMOTE:
risk_level
0    544
1    376
Name: count, dtype: int64

After SMOTE:
risk_level
1    544
0    544
Name: count, dtype: int64
```

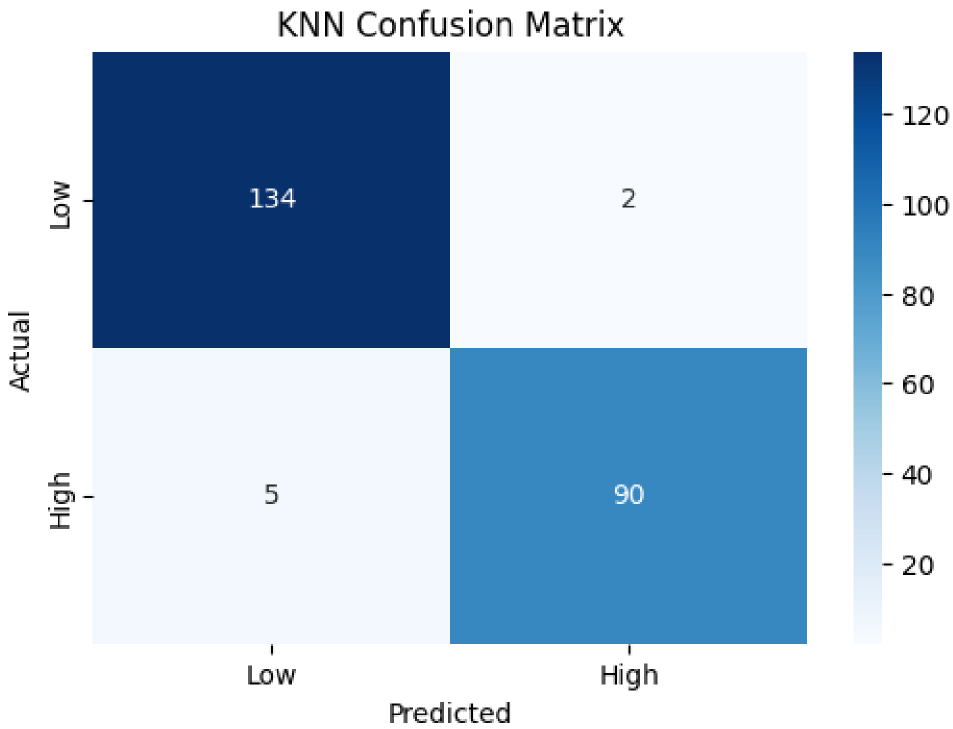
13 KNN

[77]:
```python
# Import necessary libraries
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score, confusion_matrix, classification_report
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# Assuming df_cln is the cleaned dataset from your EDA
# Ensure risk_level is encoded numerically (Low=0, High=1)
```

```python
# If not already done, encode it
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
if df_cln['risk_level'].dtype == 'object':
    df_cln['risk_level'] = le.fit_transform(df_cln['risk_level'])
    y_train = le.transform(y_train)
    y_test = le.transform(y_test)
    y_train_smote = le.transform(y_train_smote)

# --- Step 1: K-Nearest Neighbors (KNN) Model ---

# Initialize KNN classifier
knn = KNeighborsClassifier()

# Define hyperparameter grid for KNN
knn_param_grid = {
    'n_neighbors': [3, 5, 7, 9, 11],
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan']
}

# Perform GridSearchCV for KNN
knn_grid = GridSearchCV(knn, knn_param_grid, cv=5, scoring='f1', n_jobs=-1)
knn_grid.fit(X_train_smote, y_train_smote)

# Best KNN model
best_knn = knn_grid.best_estimator_
print("Best KNN Parameters:", knn_grid.best_params_)
print("Best KNN Cross-Validation F1 Score:", knn_grid.best_score_)

# Predict on test set
y_pred_knn = best_knn.predict(X_test_scaled)

# Evaluate KNN
knn_accuracy = accuracy_score(y_test, y_pred_knn)
knn_precision = precision_score(y_test, y_pred_knn, average='weighted')
knn_recall = recall_score(y_test, y_pred_knn, average='weighted')
knn_f1 = f1_score(y_test, y_pred_knn, average='weighted')

print("\nKNN Performance on Test Set:")
print(f"Accuracy: {knn_accuracy:.4f}")
print(f"Precision: {knn_precision:.4f}")
print(f"Recall: {knn_recall:.4f}")
print(f"F1 Score: {knn_f1:.4f}")
print("\nKNN Classification Report:")
print(classification_report(y_test, y_pred_knn, target_names=['Low', 'High']))
```

```python
# Confusion Matrix for KNN
knn_cm = confusion_matrix(y_test, y_pred_knn)
plt.figure(figsize=(6, 4))
sns.heatmap(knn_cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Low',␣
↪'High'], yticklabels=['Low', 'High'])
plt.title('KNN Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
Best KNN Parameters: {'metric': 'manhattan', 'n_neighbors': 3, 'weights':
'uniform'}
Best KNN Cross-Validation F1 Score: 0.9853879755199347

KNN Performance on Test Set:
Accuracy: 0.9697
Precision: 0.9699
Recall: 0.9697
F1 Score: 0.9696

KNN Classification Report:
              precision    recall  f1-score   support

         Low       0.96      0.99      0.97       136
        High       0.98      0.95      0.96        95

    accuracy                           0.97       231
   macro avg       0.97      0.97      0.97       231
weighted avg       0.97      0.97      0.97       231
```

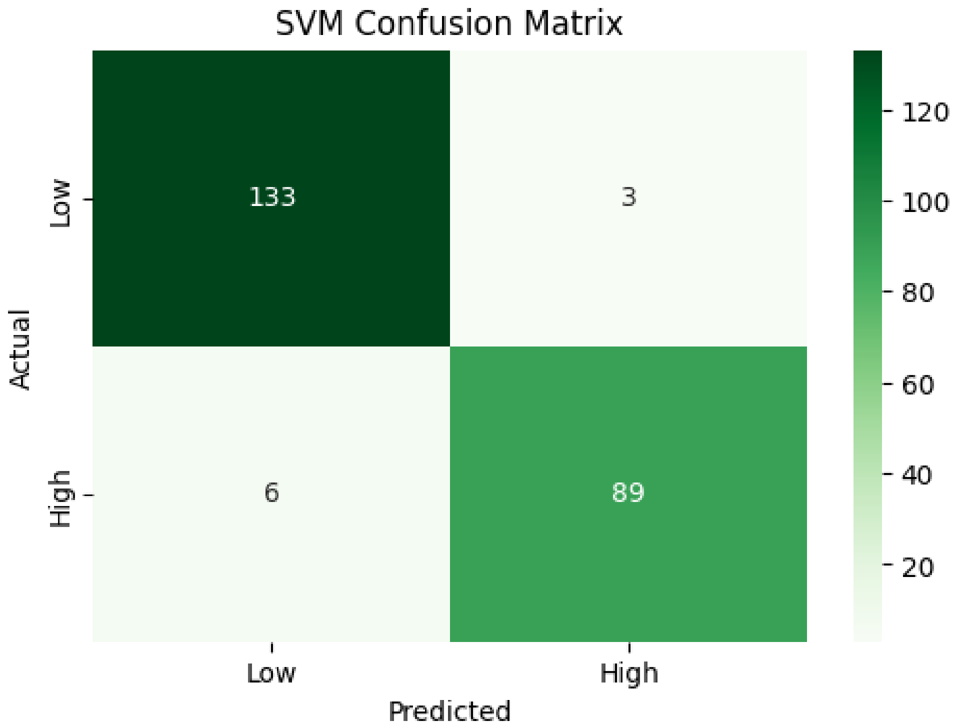


14 SVM

[78]:
```
# --- Step 2: Support Vector Machine (SVM) Model ---

# Initialize SVM classifier
svm = SVC()

# Define hyperparameter grid for SVM
svm_param_grid = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'rbf'],
    'gamma': ['scale', 'auto']
}

# Perform GridSearchCV for SVM
svm_grid = GridSearchCV(svm, svm_param_grid, cv=5, scoring='f1', n_jobs=-1)
svm_grid.fit(X_train_smote, y_train_smote)

# Best SVM model
best_svm = svm_grid.best_estimator_
print("\nBest SVM Parameters:", svm_grid.best_params_)
print("Best SVM Cross-Validation F1 Score:", svm_grid.best_score_)
```

```python
# Predict on test set
y_pred_svm = best_svm.predict(X_test_scaled)

# Evaluate SVM
svm_accuracy = accuracy_score(y_test, y_pred_svm)
svm_precision = precision_score(y_test, y_pred_svm, average='weighted')
svm_recall = recall_score(y_test, y_pred_svm, average='weighted')
svm_f1 = f1_score(y_test, y_pred_svm, average='weighted')

print("\nSVM Performance on Test Set:")
print(f"Accuracy: {svm_accuracy:.4f}")
print(f"Precision: {svm_precision:.4f}")
print(f"Recall: {svm_recall:.4f}")
print(f"F1 Score: {svm_f1:.4f}")
print("\nSVM Classification Report:")
print(classification_report(y_test, y_pred_svm, target_names=['Low', 'High']))

# Confusion Matrix for SVM
svm_cm = confusion_matrix(y_test, y_pred_svm)
plt.figure(figsize=(6, 4))
sns.heatmap(svm_cm, annot=True, fmt='d', cmap='Greens', xticklabels=['Low',␣
↪'High'], yticklabels=['Low', 'High'])
plt.title('SVM Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

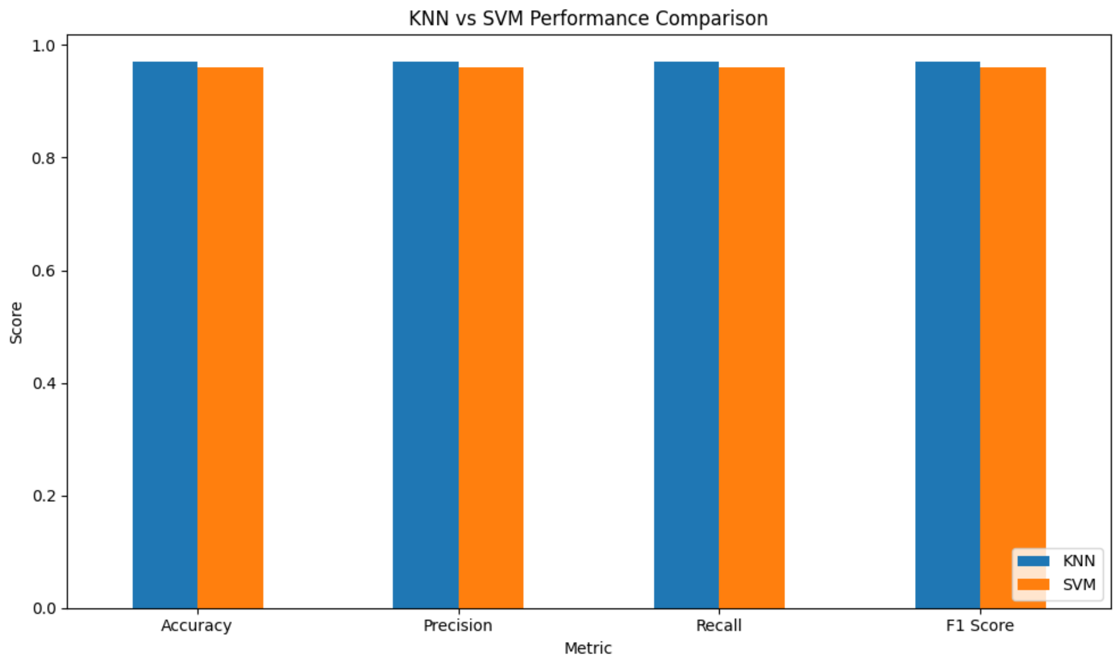
```
Best SVM Parameters: {'C': 10, 'gamma': 'scale', 'kernel': 'rbf'}
Best SVM Cross-Validation F1 Score: 0.9789247529214029

SVM Performance on Test Set:
Accuracy: 0.9610
Precision: 0.9612
Recall: 0.9610
F1 Score: 0.9609

SVM Classification Report:
              precision    recall  f1-score   support

         Low       0.96      0.98      0.97       136
        High       0.97      0.94      0.95        95

    accuracy                           0.96       231
   macro avg       0.96      0.96      0.96       231
weighted avg       0.96      0.96      0.96       231
```



Compare Model

[79]:
```python
# --- Step 3: Compare Model Performance ---

# Create a DataFrame to compare metrics
metrics_df = pd.DataFrame({
    'Metric': ['Accuracy', 'Precision', 'Recall', 'F1 Score'],
    'KNN': [knn_accuracy, knn_precision, knn_recall, knn_f1],
    'SVM': [svm_accuracy, svm_precision, svm_recall, svm_f1]
})

print("\nModel Performance Comparison:")
print(metrics_df)

# Plot comparison
metrics_df.set_index('Metric').plot(kind='bar', figsize=(10, 6))
plt.title('KNN vs SVM Performance Comparison')
plt.ylabel('Score')
plt.xticks(rotation=0)
plt.legend(loc='lower right')
plt.tight_layout()
plt.show()
```

```
Model Performance Comparison:
      Metric       KNN       SVM
0   Accuracy  0.969697  0.961039
1  Precision  0.969882  0.961176
2     Recall  0.969697  0.961039
3   F1 Score  0.969619  0.960939
```



15 Decision Tree and KNN and SVM

[80]:
```python
# Import necessary libraries
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
↪f1_score, confusion_matrix, classification_report
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# Assuming the following from your previous code:
# - X_train_smote, y_train_smote: SMOTE-balanced training data
# - X_test_scaled, y_test: Scaled test data
# - df_cln: Cleaned dataset with feature names
# - y_test is numerically encoded (0=Low, 1=High)
```

```python
# - best_knn, best_svm: Best KNN and SVM models for comparison

# Ensure y_test is a numpy array (to avoid indexing issues from previous error)
if isinstance(y_test, pd.Series):
    y_test = y_test.values
elif isinstance(y_test, pd.DataFrame):
    y_test = y_test.iloc[:, 0].values

# --- Step 1: Train Decision Tree Model ---

# Initialize Decision Tree classifier
dt = DecisionTreeClassifier(random_state=42)

# Define hyperparameter grid for Decision Tree
dt_param_grid = {
    'max_depth': [3, 5, 7, 10, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'criterion': ['gini', 'entropy']
}

# Perform GridSearchCV for Decision Tree
dt_grid = GridSearchCV(dt, dt_param_grid, cv=5, scoring='f1', n_jobs=-1)
dt_grid.fit(X_train_smote, y_train_smote)

# Best Decision Tree model
best_dt = dt_grid.best_estimator_
print("Best Decision Tree Parameters:", dt_grid.best_params_)
print("Best Decision Tree Cross-Validation F1 Score:", dt_grid.best_score_)

# Predict on test set
y_pred_dt = best_dt.predict(X_test_scaled)

# --- Step 2: Evaluate Decision Tree ---

# Calculate metrics
dt_accuracy = accuracy_score(y_test, y_pred_dt)
dt_precision = precision_score(y_test, y_pred_dt, average='weighted')
dt_recall = recall_score(y_test, y_pred_dt, average='weighted')
dt_f1 = f1_score(y_test, y_pred_dt, average='weighted')

print("\nDecision Tree Performance on Test Set:")
print(f"Accuracy: {dt_accuracy:.4f}")
print(f"Precision: {dt_precision:.4f}")
print(f"Recall: {dt_recall:.4f}")
print(f"F1 Score: {dt_f1:.4f}")
print("\nDecision Tree Classification Report:")
```

```
print(classification_report(y_test, y_pred_dt, target_names=['Low', 'High']))
```

```
Best Decision Tree Parameters: {'criterion': 'gini', 'max_depth': 10,
'min_samples_leaf': 1, 'min_samples_split': 2}
Best Decision Tree Cross-Validation F1 Score: 0.9697213769253338

Decision Tree Performance on Test Set:
Accuracy: 0.9957
Precision: 0.9957
Recall: 0.9957
F1 Score: 0.9957

Decision Tree Classification Report:
              precision    recall  f1-score   support

         Low       1.00      0.99      1.00       136
        High       0.99      1.00      0.99        95

    accuracy                           1.00       231
   macro avg       0.99      1.00      1.00       231
weighted avg       1.00      1.00      1.00       231
```

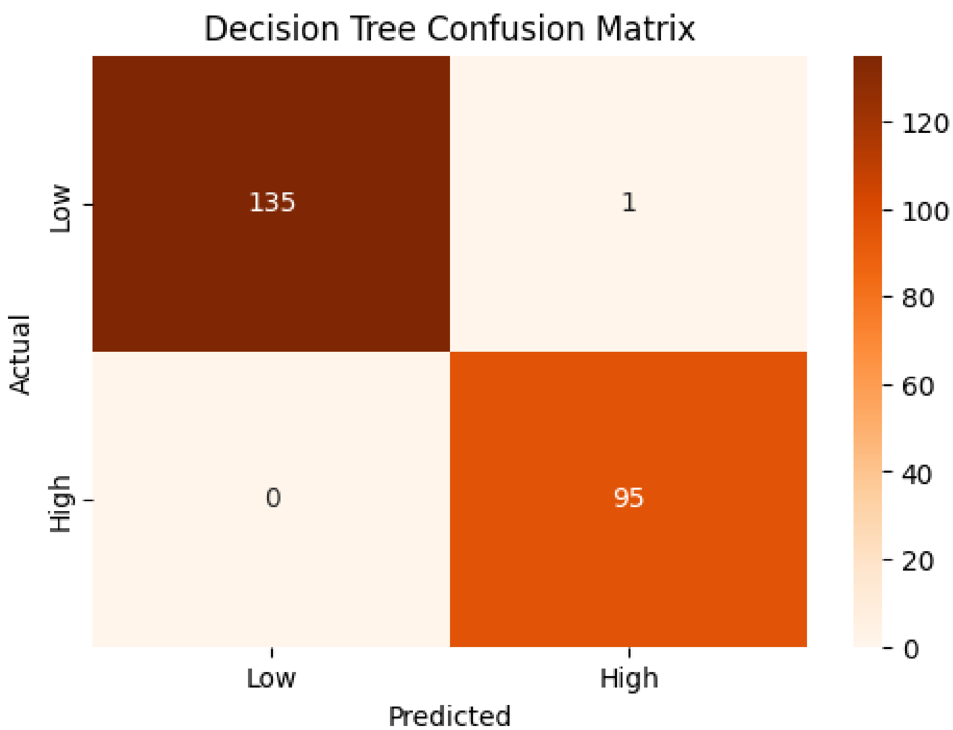
[81]:

```
# --- Step 3: Predict on a single test sample ---

# Choose an index of the sample you want to test
sample_index = 154  # You can change this to test other samples

# Extract the sample
sample_features = X_test_scaled[sample_index].reshape(1, -1)

# Make the prediction
predicted_class = best_dt.predict(sample_features)[0]

# Optional: Also show actual label for comparison
actual_class = y_test[sample_index]

# Map numeric prediction to class label
label_map = {0: "Low Risk", 1: "High Risk"}
predicted_label = label_map[predicted_class]
actual_label = label_map[actual_class]

# Display the result
print(f"\nPrediction for Test Sample at Index {sample_index}:")
print(f"Predicted Risk: {predicted_label}")
print(f"Actual Risk: {actual_label}")
```

```
Prediction for Test Sample at Index 154:
Predicted Risk: High Risk
Actual Risk: High Risk
```

[82]:

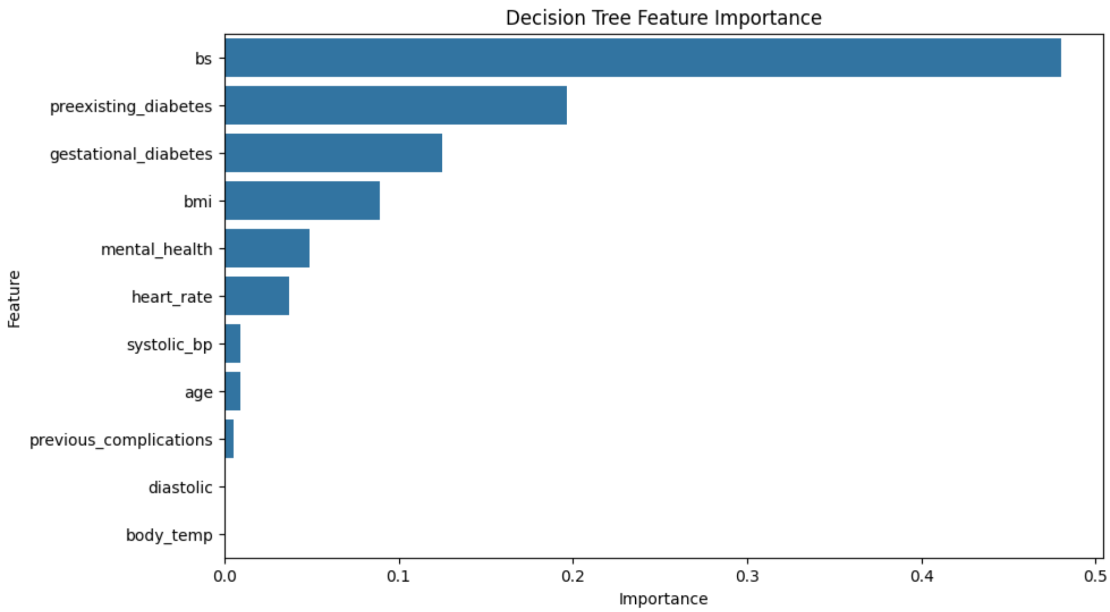
```
# Confusion Matrix for Decision Tree
dt_cm = confusion_matrix(y_test, y_pred_dt)
plt.figure(figsize=(6, 4))
sns.heatmap(dt_cm, annot=True, fmt='d', cmap='Oranges', xticklabels=['Low',␣
 ↪'High'], yticklabels=['Low', 'High'])
plt.title('Decision Tree Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



[83]:

```
# --- Step 3: Visualize Feature Importance ---

# Get feature names
feature_names = df_cln.drop('risk_level', axis=1).columns.tolist()

# Get feature importance from the best Decision Tree model
feature_importance = pd.DataFrame({
    'Feature': feature_names,
    'Importance': best_dt.feature_importances_
```

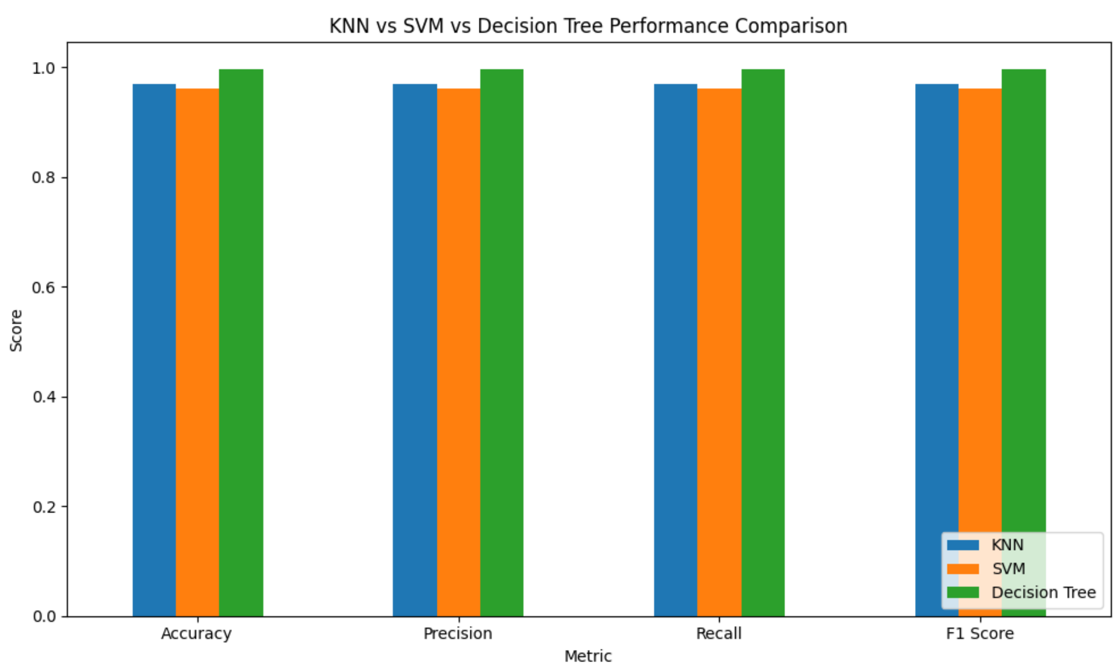
```
}).sort_values(by='Importance', ascending=False)

# Plot feature importance
plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=feature_importance)
plt.title('Decision Tree Feature Importance')
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.show()
```



[84]:
```
# --- Step 4: Compare with KNN and SVM ---

# Assuming KNN and SVM predictions from previous code
y_pred_knn = best_knn.predict(X_test_scaled)
y_pred_svm = best_svm.predict(X_test_scaled)

# Calculate metrics for KNN and SVM
knn_accuracy = accuracy_score(y_test, y_pred_knn)
knn_precision = precision_score(y_test, y_pred_knn, average='weighted')
knn_recall = recall_score(y_test, y_pred_knn, average='weighted')
knn_f1 = f1_score(y_test, y_pred_knn, average='weighted')

svm_accuracy = accuracy_score(y_test, y_pred_svm)
svm_precision = precision_score(y_test, y_pred_svm, average='weighted')
svm_recall = recall_score(y_test, y_pred_svm, average='weighted')
svm_f1 = f1_score(y_test, y_pred_svm, average='weighted')
```

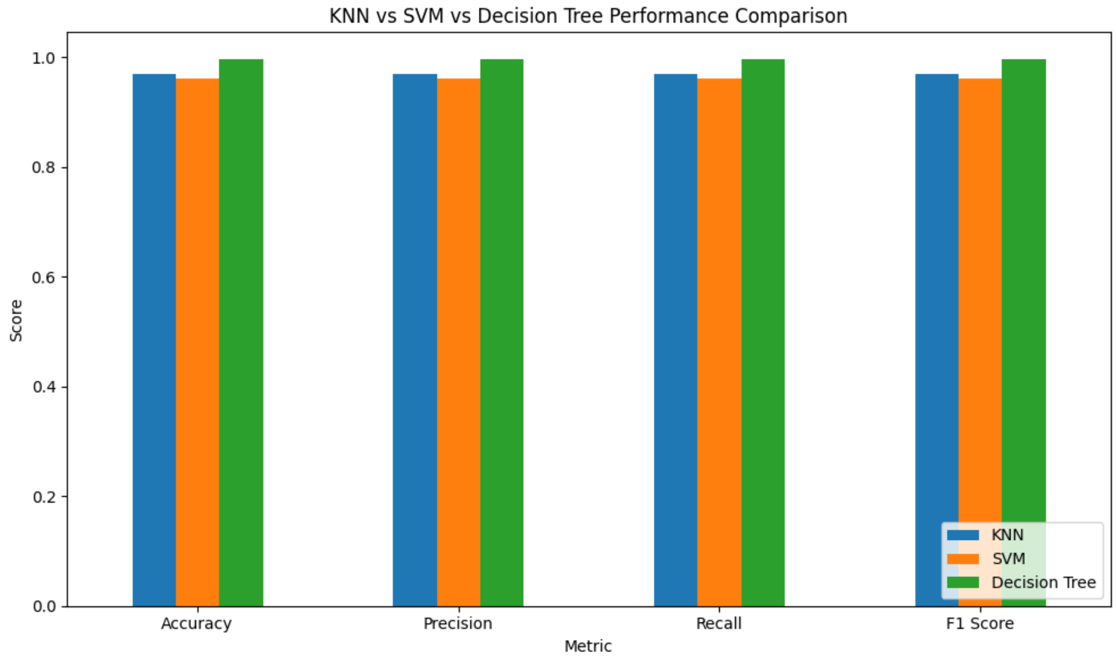
```python
# Create a DataFrame to compare metrics
metrics_df = pd.DataFrame({
    'Metric': ['Accuracy', 'Precision', 'Recall', 'F1 Score'],
    'KNN': [knn_accuracy, knn_precision, knn_recall, knn_f1],
    'SVM': [svm_accuracy, svm_precision, svm_recall, svm_f1],
    'Decision Tree': [dt_accuracy, dt_precision, dt_recall, dt_f1]
})

print("\nModel Performance Comparison:")
print(metrics_df)

# Plot comparison
metrics_df.set_index('Metric').plot(kind='bar', figsize=(10, 6))
plt.title('KNN vs SVM vs Decision Tree Performance Comparison')
plt.ylabel('Score')
plt.xticks(rotation=0)
plt.legend(loc='lower right')
plt.tight_layout()
plt.show()
```

```
Model Performance Comparison:
      Metric       KNN       SVM  Decision Tree
0   Accuracy  0.969697  0.961039       0.995671
1  Precision  0.969882  0.961176       0.995716
2     Recall  0.969697  0.961039       0.995671
3   F1 Score  0.969619  0.960939       0.995674
```



[85]:
```python
# Create a DataFrame to compare metrics
metrics_df = pd.DataFrame({
    'Metric': ['Accuracy', 'Precision', 'Recall', 'F1 Score'],
    'KNN': [knn_accuracy, knn_precision, knn_recall, knn_f1],
    'SVM': [svm_accuracy, svm_precision, svm_recall, svm_f1],
    'Decision Tree': [dt_accuracy, dt_precision, dt_recall, dt_f1]
})

print("\nModel Performance Comparison:")
print(metrics_df.round(4))  # Round to 4 decimal places for clarity

# Plot comparison
metrics_df.set_index('Metric').plot(kind='bar', figsize=(10, 6))
plt.title('KNN vs SVM vs Decision Tree Performance Comparison')
plt.ylabel('Score')
plt.xticks(rotation=0)
plt.legend(loc='lower right')
plt.tight_layout()
plt.show()
```

```
Model Performance Comparison:
      Metric     KNN     SVM  Decision Tree
0   Accuracy  0.9697  0.9610         0.9957
1  Precision  0.9699  0.9612         0.9957
2     Recall  0.9697  0.9610         0.9957
3   F1 Score  0.9696  0.9609         0.9957
```



16 Explainable AI

[134]:
```
from lime.lime_tabular import LimeTabularExplainer
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
import numpy as np

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
 ↪random_state=70)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

print("X_train_scaled:", X_train_scaled.shape)
print("y_train:", y_train.shape)


# Train multiple models
models = {
```

```
    'KNN': KNeighborsClassifier(),
    'Decision Tree': DecisionTreeClassifier(random_state=30),
    'SVM': SVC(random_state=42, probability=True)
}

# Evaluate models
best_model_name = None
best_accuracy = 0
best_model = None

for name, model in models.items():
    model.fit(X_train_scaled, y_train)
    y_pred = model.predict(X_test_scaled)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"{name} Accuracy: {accuracy:.4f}")
    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_model_name = name
        best_model = model

print(f"\nBest Model: {best_model_name} with Accuracy: {best_accuracy:.4f}")

# LIME Explanation
explainer = LimeTabularExplainer(
    X_train_scaled.values if hasattr(X_train_scaled, 'values') else␣
 ↪X_train_scaled,
    feature_names=feature_names,
    class_names=class_names,
    mode='classification'
)

decision_tree = DecisionTreeClassifier(random_state=42)
decision_tree.fit(X_train_scaled, y_train)

# Ensure X_test_scaled is a NumPy array
instance_array = X_test_scaled.values if hasattr(X_test_scaled, 'values') else␣
 ↪X_test_scaled
instance = instance_array[0]

prediction = decision_tree.predict_proba([instance])[0]
print(f"\nPrediction Probabilities for Instance: {prediction}")

exp = explainer.explain_instance(
    instance,
    decision_tree.predict_proba,
    num_features=10
)
```

```
print("\nLIME Explanation for Instance:")
exp.as_list()

exp.show_in_notebook(show_table=True, show_all=False)
import matplotlib.pyplot as plt
fig = exp.as_pyplot_figure()
plt.tight_layout()
plt.show()
```

```
X_train_scaled: (949, 11)
y_train: (949,)
KNN Accuracy: 0.9580
Decision Tree Accuracy: 0.9664
SVM Accuracy: 0.9580

Best Model: Decision Tree with Accuracy: 0.9664

Prediction Probabilities for Instance: [1. 0.]

LIME Explanation for Instance:
```

```
<IPython.core.display.HTML object>
```