```
# IMPORTANT: RUN THIS CELL IN ORDER TO IMPORT YOUR KAGGLE DATA SOURCES
# TO THE CORRECT LOCATION (/kaggle/input) IN YOUR NOTEBOOK,
# THEN FEEL FREE TO DELETE THIS CELL.
# NOTE: THIS NOTEBOOK ENVIRONMENT DIFFERS FROM KAGGLE'S PYTHON
# ENVIRONMENT SO THERE MAY BE MISSING LIBRARIES USED BY YOUR
# NOTEBOOK.

import os
import sys
from tempfile import NamedTemporaryFile
from urllib.request import urlopen
from urllib.parse import unquote, urlparse
from urllib.error import HTTPError
from zipfile import ZipFile
import tarfile
import shutil

CHUNK_SIZE = 40960
DATA_SOURCE_MAPPING = 'maternal-health-risk-data-set:https%3A%2F%2Fstorage.googleapis.com%2Fkaggle-data-sets%2F2497178%2F4237054%2Fbundle%2Farchive.zip%3FX-Goog-Algorithm%

KAGGLE_INPUT_PATH='/kaggle/input'
KAGGLE_WORKING_PATH='/kaggle/working'
KAGGLE_SYMLINK='kaggle'

!umount /kaggle/input/ 2> /dev/null
shutil.rmtree('/kaggle/input', ignore_errors=True)
os.makedirs(KAGGLE_INPUT_PATH, 0o777, exist_ok=True)
os.makedirs(KAGGLE_WORKING_PATH, 0o777, exist_ok=True)

try:
  os.symlink(KAGGLE_INPUT_PATH, os.path.join("..", 'input'), target_is_directory=True)
except FileExistsError:
  pass
try:
  os.symlink(KAGGLE_WORKING_PATH, os.path.join("..", 'working'), target_is_directory=True)
except FileExistsError:
  pass

for data_source_mapping in DATA_SOURCE_MAPPING.split(','):
    directory, download_url_encoded = data_source_mapping.split(':')
    download_url = unquote(download_url_encoded)
    filename = urlparse(download_url).path
    destination_path = os.path.join(KAGGLE_INPUT_PATH, directory)
    try:
        with urlopen(download_url) as fileres, NamedTemporaryFile() as tfile:
            total_length = fileres.headers['content-length']
            print(f'Downloading {directory}, {total_length} bytes compressed')
            dl = 0
            data = fileres.read(CHUNK_SIZE)
            while len(data) > 0:
                dl += len(data)
                tfile.write(data)
                done = int(50 * dl / int(total_length))
                sys.stdout.write(f"\r[{'=' * done}{' ' * (50-done)}] {dl} bytes downloaded")
                sys.stdout.flush()
                data = fileres.read(CHUNK_SIZE)
            if filename.endswith('.zip'):
              with ZipFile(tfile) as zfile:
                zfile.extractall(destination_path)
            else:
              with tarfile.open(tfile.name) as tarfile:
                tarfile.extractall(destination_path)
            print(f'\nDownloaded and uncompressed: {directory}')
    except HTTPError as e:
        print(f'Failed to load (likely expired) {download_url} to path {destination_path}')
        continue
    except OSError as e:
        print(f'Failed to load {download_url} to path {destination_path}')
        continue

print('Data source import complete.')
```

```
Downloading maternal-health-risk-data-set, 2734 bytes compressed
[==================================================] 2734 bytes downloaded
Downloaded and uncompressed: maternal-health-risk-data-set
Data source import complete.
```

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
```

```
data = pd.read_csv("/kaggle/input/maternal-health-risk-data-set/Maternal_Risk.csv")
data.head()
```

	Age	SystolicBP	DiastolicBP	BS	BodyTemp	HeartRate	RiskLevel
0	25	130	80	15.0	98.0	86	high risk
1	35	140	90	13.0	98.0	70	high risk
2	29	90	70	8.0	100.0	80	high risk
3	30	140	85	7.0	98.0	70	high risk
4	35	120	60	6.1	98.0	76	low risk

Next steps: Generate code with data View recommended plots

```
data.shape
```

```
(808, 7)
```

```
data.dtypes
```

```
Age              int64
SystolicBP       int64
DiastolicBP      int64
BS             float64
BodyTemp       float64
HeartRate        int64
RiskLevel       object
dtype: object
```

```
data.describe()
```

	Age	SystolicBP	DiastolicBP	BS	BodyTemp	HeartRate
count	808.000000	808.000000	808.000000	808.000000	808.000000	808.000000
mean	30.585396	112.972772	77.500000	9.264839	98.640347	74.297030
std	13.922075	19.924312	14.772207	3.617635	1.386501	8.822686
min	10.000000	70.000000	49.000000	6.000000	98.000000	7.000000
25%	19.000000	90.000000	65.000000	7.000000	98.000000	70.000000
50%	27.000000	120.000000	80.000000	7.500000	98.000000	76.000000
75%	40.000000	120.000000	90.000000	11.000000	98.000000	80.000000
max	70.000000	160.000000	100.000000	19.000000	103.000000	90.000000

```
data.columns
```

```
Index(['Age', 'SystolicBP', 'DiastolicBP', 'BS', 'BodyTemp', 'HeartRate',
       'RiskLevel'],
      dtype='object')
```

Pre processing

Double-click (or enter) to edit

```
print('Count of Null values')
data.isnull().sum()
```

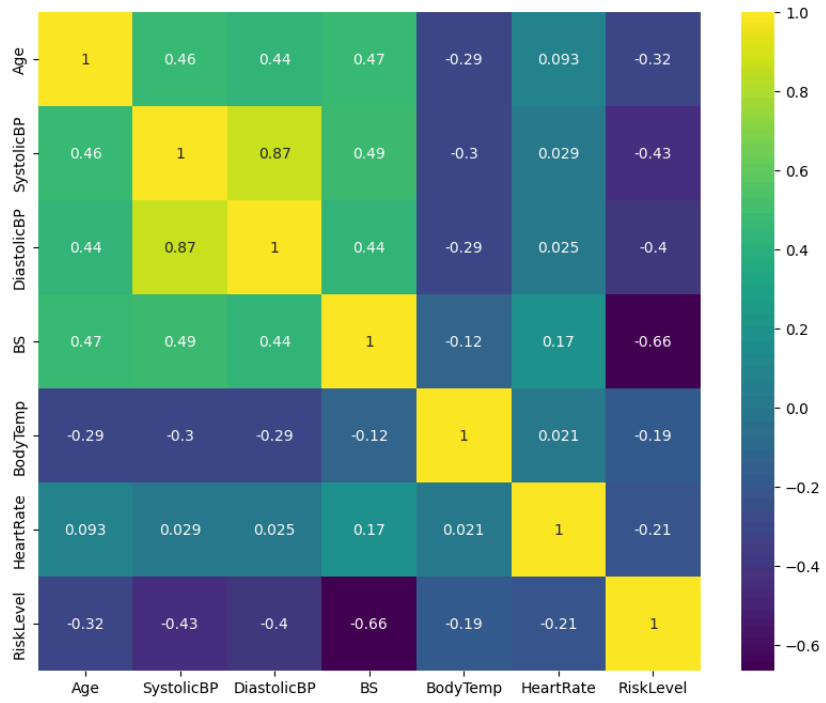
```
Count of Null values
Age            0
SystolicBP     0
DiastolicBP    0
BS             0
BodyTemp       0
HeartRate      0
RiskLevel      0
dtype: int64
```

```
print('Count of unique values')
for i in data.columns:
    print(f'{i} : {len(data[i].unique())}')
```

```
Count of unique values
Age : 48
SystolicBP : 18
DiastolicBP : 16
BS : 29
BodyTemp : 8
HeartRate : 15
RiskLevel : 2
```

```
from sklearn.preprocessing import LabelEncoder
scaler = LabelEncoder()
df_label = data.copy()
df_label['RiskLevel'] = scaler.fit_transform(df_label['RiskLevel'])
corr = df_label.corr()
```

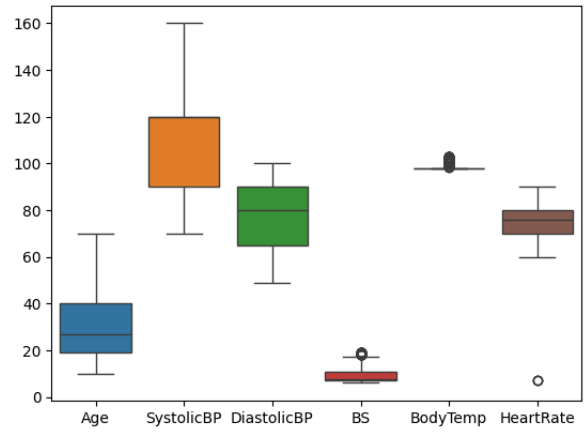
```
plt.figure(figsize = (10, 8))
sns.heatmap(corr , annot = True , cmap = 'viridis')
plt.show()
```



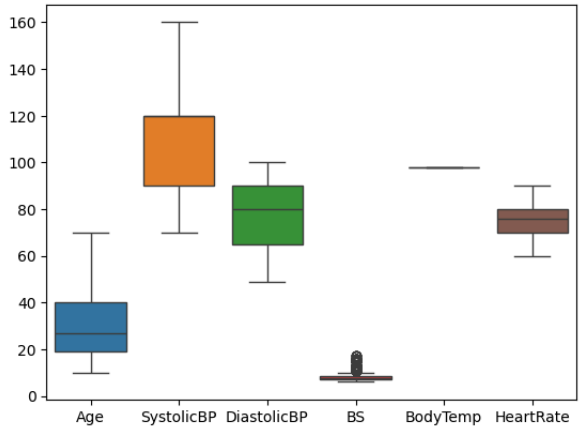
Outlier Detection

```
sns.boxplot(data.drop('RiskLevel' , axis = 1))
```

```
<Axes: >
```




```
from sklearn.impute import SimpleImputer

train = data.drop('RiskLevel' , axis = 1)
def impute_outliers(data, column, factor):
    q1 = data[column].quantile(0.25)
    q3 = data[column].quantile(0.75)
    iqr = q3 - q1
    lower_bound = q1 - factor * iqr
    upper_bound = q3 + factor * iqr

    data_copy = data.copy()
    data_copy[column] = np.where(data_copy[column] < lower_bound, np.nan, data_copy[column])
    data_copy[column] = np.where(data_copy[column] > upper_bound, np.nan, data_copy[column])

    imputer = SimpleImputer(strategy="mean")
    data_imputed = imputer.fit_transform(data_copy)

    return pd.DataFrame(data_imputed, columns=data.columns)

for column in train.columns:
    train = impute_outliers(train, column, 1.5)

df = train
sns.boxplot(df)
plt.show()
```




```
print('Age less than 18:')
df['Age'].where(df['Age'] < 18).value_counts()
```

```
Age less than 18:
15.0    66
17.0    59
12.0    27
16.0    13
13.0     7
14.0     3
10.0     2
Name: Age, dtype: int64
```

```
print('Age less than 18:')
mean_age = df['Age'].mean()
df.loc[df['Age'] < 18, 'Age'] = mean_age
df['Age'].where(df['Age'] < 18).value_counts().sum()
```

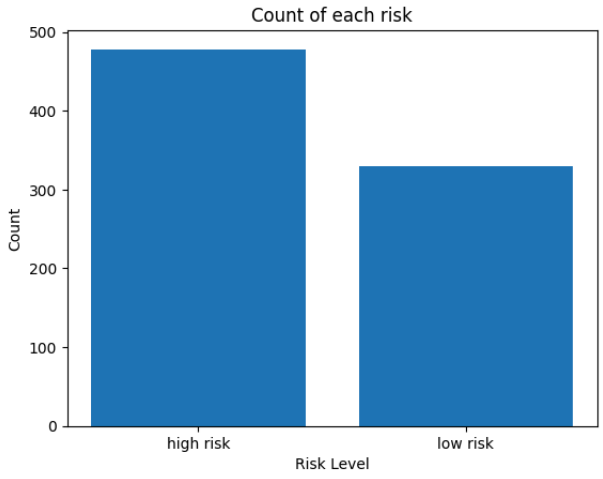
```
Age less than 18:
0
```

```
df['RiskLevel'] = data['RiskLevel']
df.head()
```

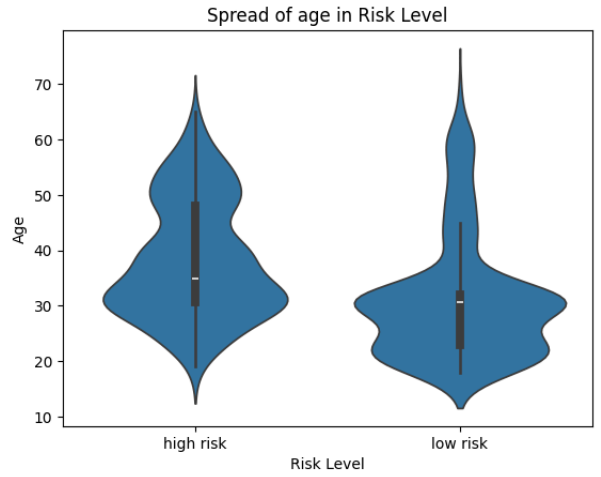
	Age	SystolicBP	DiastolicBP	BS	BodyTemp	HeartRate	RiskLevel
0	25.0	130.0	80.0	15.0	98.0	86.0	high risk
1	35.0	140.0	90.0	13.0	98.0	70.0	high risk
2	29.0	90.0	70.0	8.0	98.0	80.0	high risk
3	30.0	140.0	85.0	7.0	98.0	70.0	high risk
4	35.0	120.0	60.0	6.1	98.0	76.0	low risk

Next steps: Generate code with `df` View recommended plots

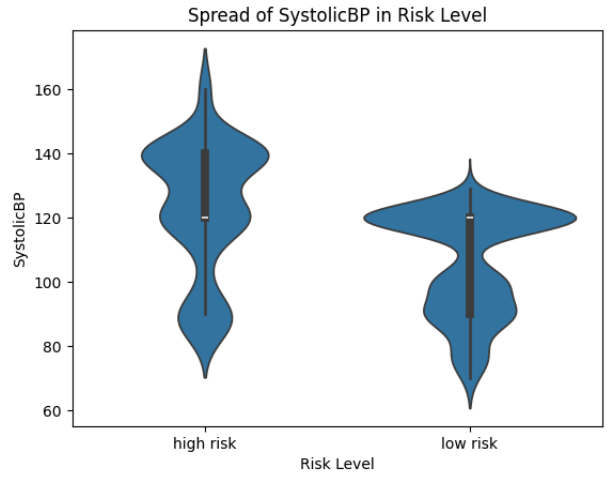
```
plt.bar(df['RiskLevel'].unique() , df['RiskLevel'].value_counts())
plt.xlabel('Risk Level')
plt.ylabel('Count')
plt.title('Count of each risk')
plt.show()
```



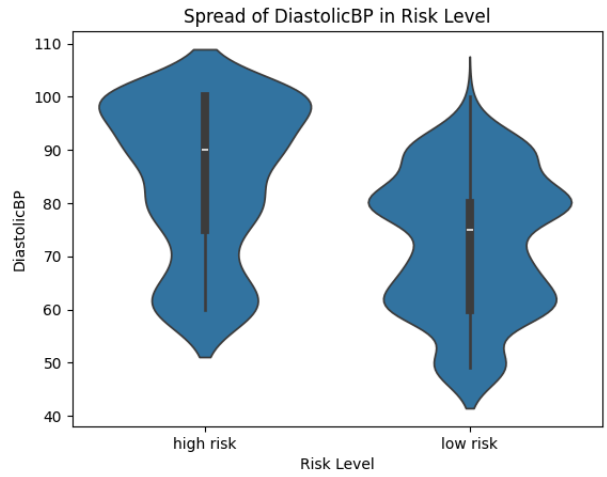

```
sns.violinplot(x="RiskLevel", y="Age", data=df)
plt.xlabel('Risk Level')
plt.ylabel('Age')
plt.title('Spread of age in Risk Level')
plt.show()
```



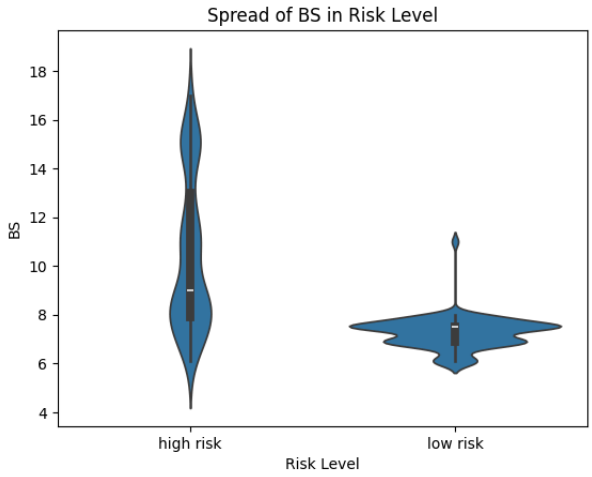

```
sns.violinplot(x="RiskLevel", y="SystolicBP", data=df)
plt.xlabel('Risk Level')
plt.ylabel('SystolicBP')
plt.title('Spread of SystolicBP in Risk Level')
plt.show()
```



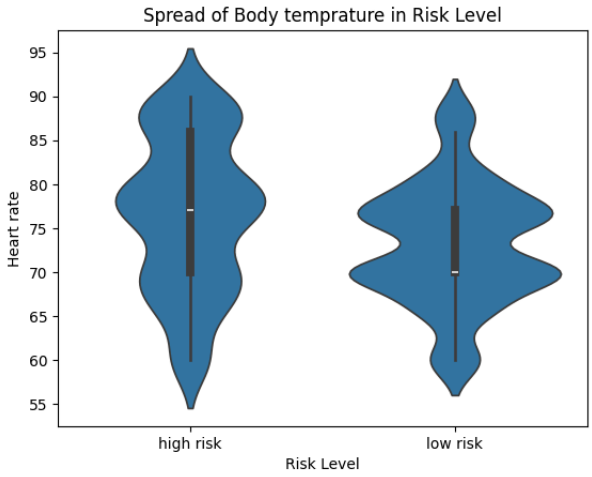

```
sns.violinplot(x="RiskLevel", y="DiastolicBP", data=df)
plt.xlabel('Risk Level')
plt.ylabel('DiastolicBP')
plt.title('Spread of DiastolicBP in Risk Level')
plt.show()
```




```
sns.violinplot(x="RiskLevel", y="BS", data=df)
plt.xlabel('Risk Level')
plt.ylabel('BS')
plt.title('Spread of BS in Risk Level')
plt.show()
```




```
sns.violinplot(x="RiskLevel", y="HeartRate", data=df)
plt.xlabel('Risk Level')
plt.ylabel('Heart rate')
plt.title('Spread of Body temprature in Risk Level')
plt.show()
```



Univariate analysis

```
df_highrisk = df.where(df['RiskLevel'] == 'high risk').dropna()
df_highrisk.head()
```

	Age	SystolicBP	DiastolicBP	BS	BodyTemp	HeartRate	RiskLevel
0	25.0	130.0	80.0	15.00	98.0	86.0	high risk
1	35.0	140.0	90.0	13.00	98.0	70.0	high risk
2	29.0	90.0	70.0	8.00	98.0	80.0	high risk
3	30.0	140.0	85.0	7.00	98.0	70.0	high risk
5	23.0	140.0	80.0	7.01	98.0	70.0	high risk

Next steps: Generate code with df_highrisk View recommended plots

```
df_lowrisk = df.where(df['RiskLevel'] == 'low risk').dropna()
df_lowrisk.head()
```

	Age	SystolicBP	DiastolicBP	BS	BodyTemp	HeartRate	RiskLevel
4	35.000000	120.0	60.0	6.10	98.0	76.0	low risk
8	23.000000	90.0	60.0	7.01	98.0	76.0	low risk
10	25.000000	110.0	89.0	7.01	98.0	77.0	low risk
11	30.585396	120.0	80.0	7.01	98.0	70.0	low risk
14	30.585396	70.0	50.0	6.90	98.0	70.0	low risk

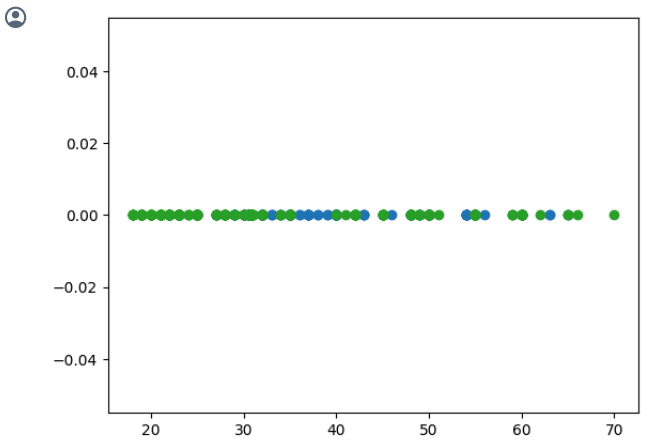
Next steps: Generate code with df_lowrisk View recommended plots

```
df_midrisk = df.where(df['RiskLevel'] == 'mid risk').dropna()
df_midrisk.head()
```

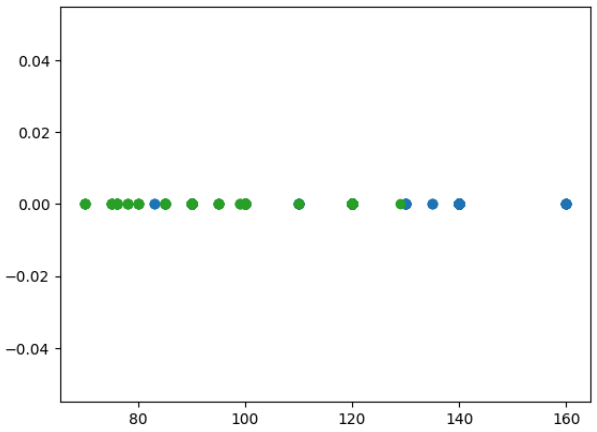
	Age	SystolicBP	DiastolicBP	BS	BodyTemp	HeartRate	RiskLevel

```
plt.plot(df_highrisk['Age'] , np.zeros_like(df_highrisk['Age']) , 'o')
```

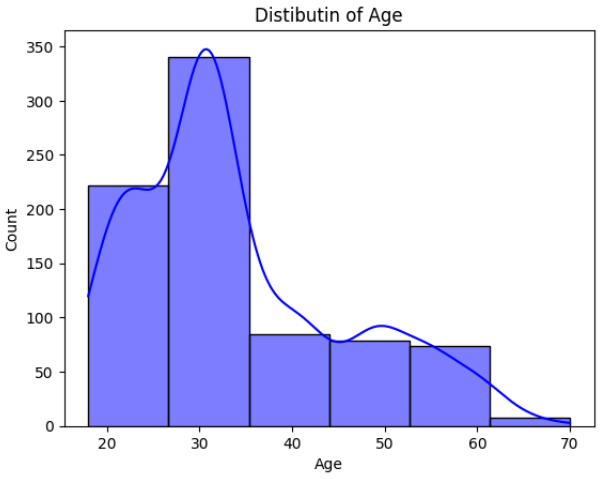
```
plt.plot(df_midrisk['Age'] , np.zeros_like(df_midrisk['Age']) , 'o')
plt.plot(df_lowrisk['Age'] , np.zeros_like(df_lowrisk['Age']) , 'o')
plt.show()
```



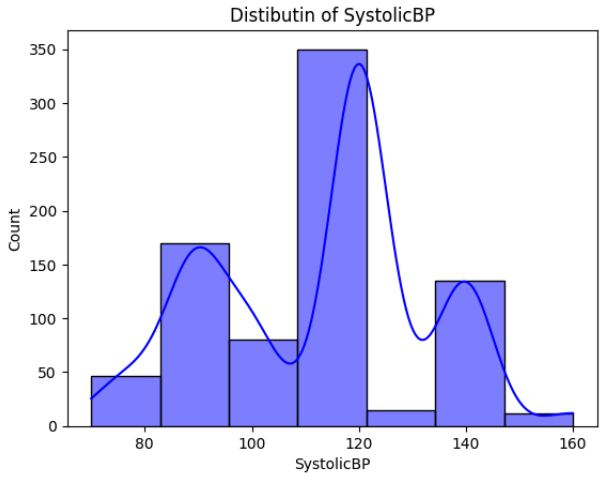

```
plt.plot(df_highrisk['SystolicBP'] , np.zeros_like(df_highrisk['SystolicBP']), 'o')
plt.plot(df_midrisk['SystolicBP'] , np.zeros_like(df_midrisk['SystolicBP']), 'o')
plt.plot(df_lowrisk['SystolicBP'] , np.zeros_like(df_lowrisk['SystolicBP']) , 'o')
plt.show()
```



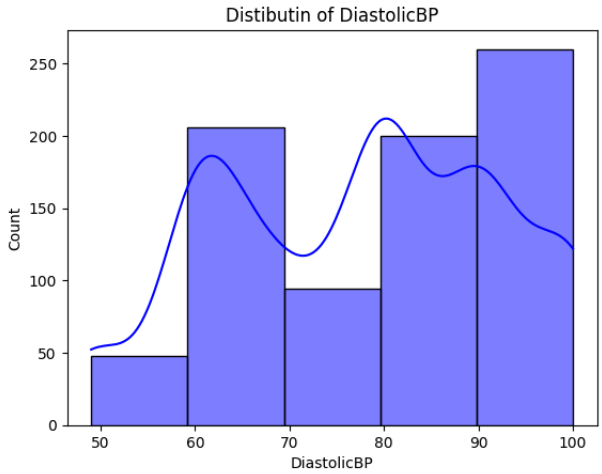

```
sns.histplot(df['Age'], bins=6, kde=True, color='blue', edgecolor='k')
plt.title('Distibutin of Age')
plt.show()
```



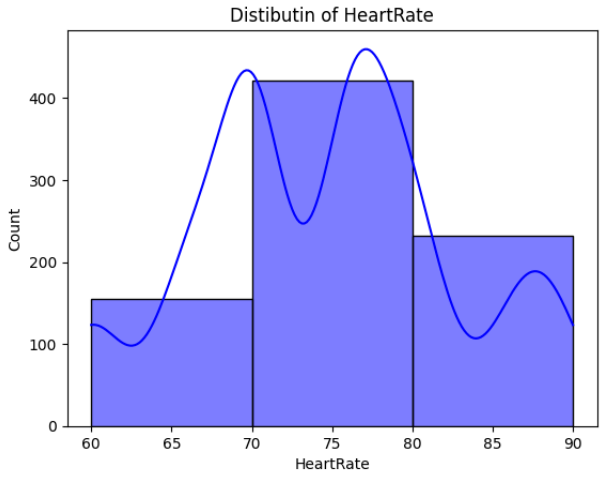

```
sns.histplot(df['SystolicBP'], bins=7, kde=True, color='blue', edgecolor='k')
plt.title('Distibutin of SystolicBP')
plt.show()
```



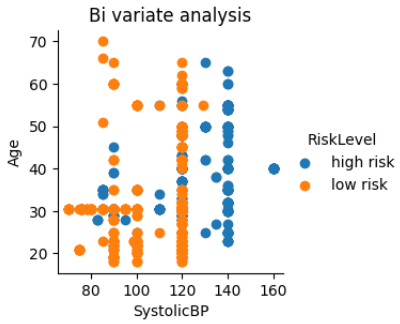

```
sns.histplot(df['DiastolicBP'], bins=5, kde=True, color='blue', edgecolor='k')
plt.title('Distibutin of DiastolicBP')
plt.show()
```



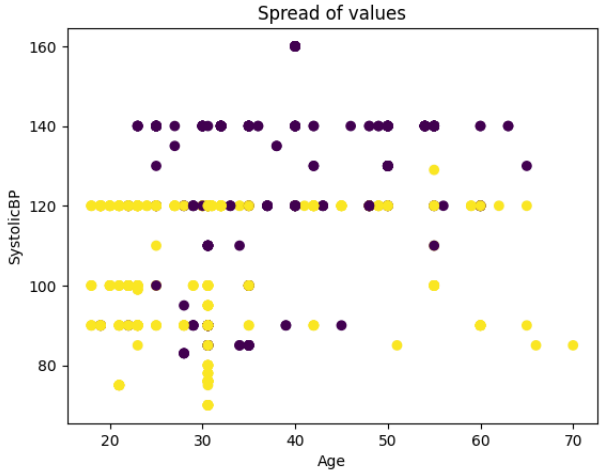

```
sns.histplot(df['HeartRate'], bins=3, kde=True, color='blue', edgecolor='k')
plt.title('Distibutin of HeartRate')
plt.show()
```



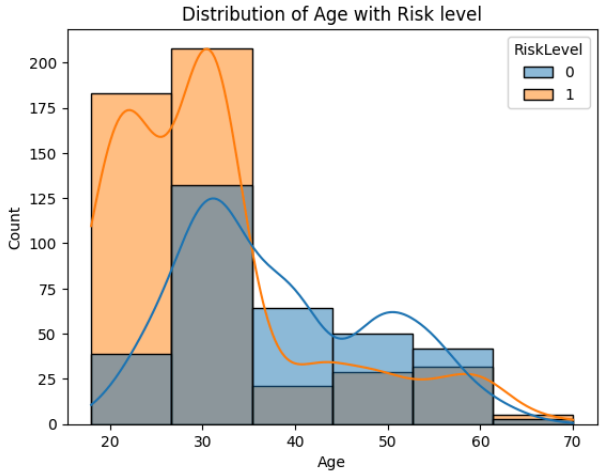

```
sns.FacetGrid(df, hue = "RiskLevel").map(plt.scatter , "SystolicBP" , "Age" ).add_legend();
plt.title('Bi variate analysis')
plt.show()
```



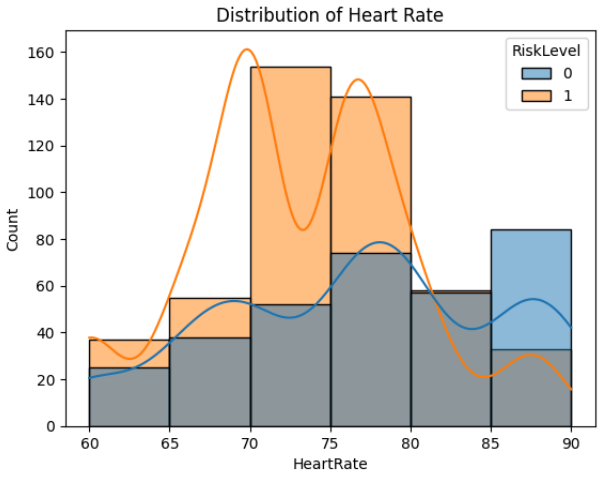

```
df['RiskLevel'] = scaler.fit_transform(df['RiskLevel'])
plt.scatter(data=df, x='Age', y='SystolicBP', c = 'RiskLevel')
plt.xlabel('Age')
plt.ylabel('SystolicBP')
plt.title('Spread of values')
plt.show()
```




```
sns.histplot(data=df, x='Age', hue='RiskLevel', kde=True , bins = 6)
plt.title('Distribution of Age with Risk level')
plt.show()
```

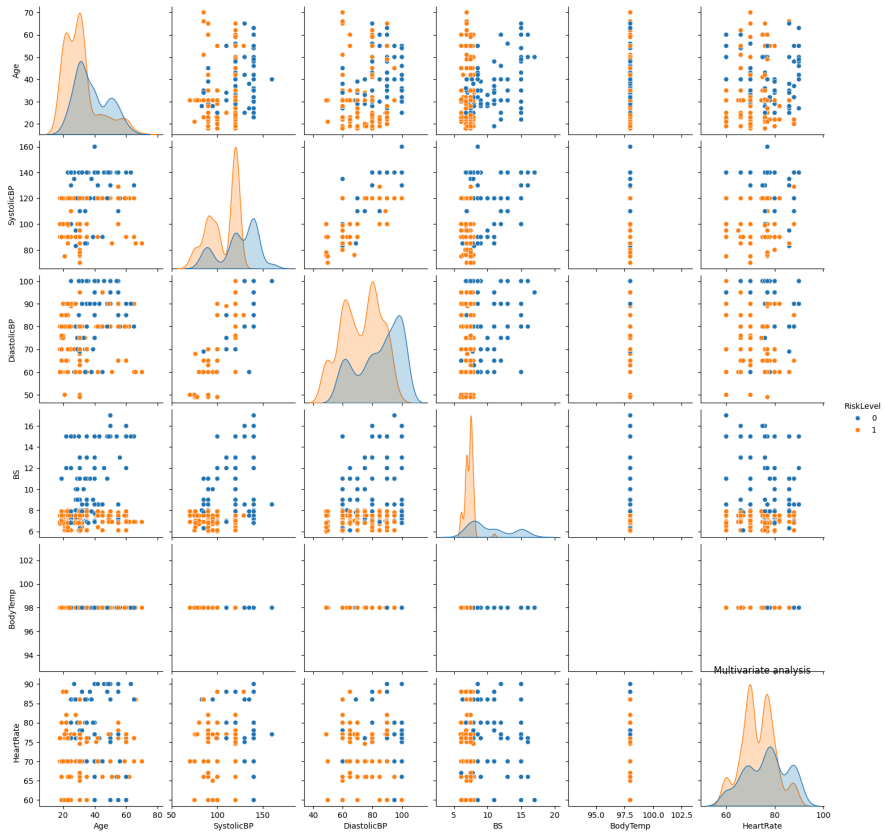



```
sns.histplot(data=df, x='HeartRate', hue='RiskLevel', kde=True , bins = 6)
plt.title('Distribution of Heart Rate')
plt.show()
```



Multi variate analysis

```
sns.pairplot(df , hue = 'RiskLevel')
plt.title('Multivariate analysis')
plt.show()
```



Classification

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score , classification_report
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier

X_train , X_test , y_train , y_test = train_test_split(df.drop('RiskLevel' , axis = 1) , df['RiskLevel'] , test_size=0.3, random_state=42)

print(f'X train : {X_train.shape}')
print(f'X test : {X_test.shape}')
print(f'Y train : {y_train.shape}')
print(f'Y test : {y_test.shape}')
```

```
X train : (565, 6)
X test : (243, 6)
Y train : (565,)
Y test : (243,)
```

```
print('DecisionTreeClassifier:')
model_dec = DecisionTreeClassifier(random_state = 42)
model_dec.fit(X_train , y_train)

y_predict = model_dec.predict(X_test)

accuracy = accuracy_score(y_predict , y_test)
print(f'Accuracy : {accuracy:.2f}')

print(classification_report(y_test , y_predict))
```

```
DecisionTreeClassifier:
Accuracy : 0.96
              precision    recall  f1-score   support

           0       0.94      0.98      0.96       104
           1       0.99      0.95      0.97       139

    accuracy                           0.96       243
   macro avg       0.96      0.97      0.96       243
weighted avg       0.96      0.96      0.96       243
```

```
print('RandomForestClassifier:')
model = RandomForestClassifier(random_state = 42)
```