In [30]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from matplotlib.patches import Ellipse
import warnings
warnings.filterwarnings("ignore", message=".*the default behavior of `mode
```

In [31]:

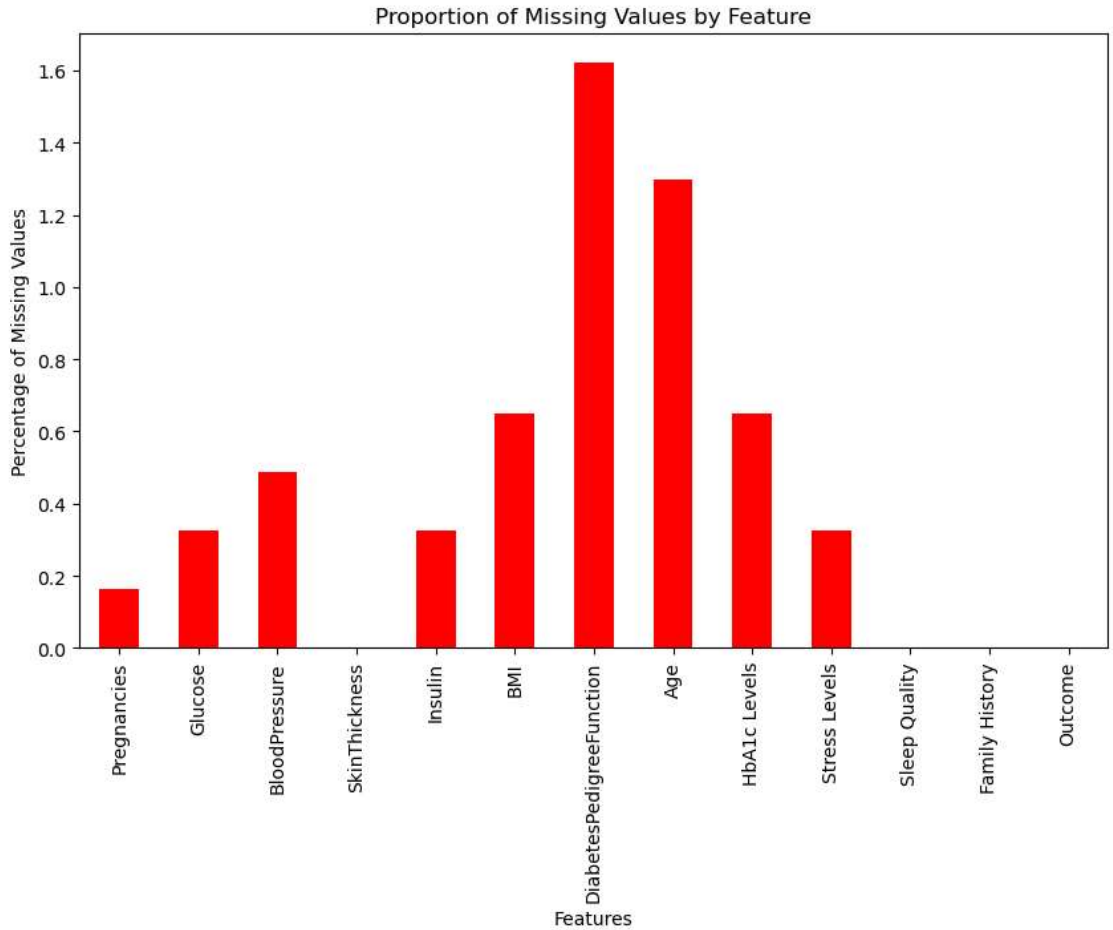
```
data = pd.read_csv("dataset.csv")
print(data.head())
print(data.describe())
print("info")
print(data.info())
print(data.isnull().sum())
print(data.duplicated().sum())
```

```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI
\
0          6.0    148.0           72.0             35      0.0   NaN
1          1.0     85.0           66.0             29      0.0  26.6
2          8.0    183.0           64.0              0      0.0  23.3
3          1.0     89.0           66.0             23     94.0  28.1
4          0.0    137.0           40.0             35    168.0  43.1

   DiabetesPedigreeFunction   Age  HbA1c Levels  Stress Levels  Sleep
Quality  \
0                     0.627  50.0           6.4            3.0
8
1                     0.351  31.0           6.1            2.0
6
2                     0.672  32.0           7.2            5.0
7
3                     0.167  21.0           6.5            3.0
6
4                     2.288  33.0           5.8            4.0
```

In [32]:

```
missing_values = data.isnull().sum()
total_missing = missing_values.sum()
missing_percent = (missing_values / data.shape[0]) * 100
print("Missing values:\n", missing_values)
print("Missing values percentage:\n", missing_percent)


plt.figure(figsize=(10, 6))
missing_percent.plot(kind='bar', color='red')
plt.title('Proportion of Missing Values by Feature')
plt.xlabel('Features')
plt.ylabel('Percentage of Missing Values')
plt.show()

print("Duplicates:", data.duplicated().sum())
```

```
Missing values:
 Pregnancies                 1
Glucose                     2
BloodPressure               3
SkinThickness               0
Insulin                     2
BMI                         4
DiabetesPedigreeFunction   10
Age                         8
HbA1c Levels                4
Stress Levels               2
Sleep Quality               0
Family History              0
Outcome                     0
dtype: int64
Missing values percentage:
 Pregnancies                 0.162075
Glucose                     0.324149
BloodPressure               0.486224
SkinThickness               0.000000
Insulin                     0.324149
BMI                         0.648298
DiabetesPedigreeFunction    1.620746
Age                         1.296596
HbA1c Levels                0.648298
Stress Levels               0.324149
Sleep Quality               0.000000
Family History              0.000000
Outcome                     0.000000
dtype: float64
```




```
Duplicates: 536
```

In [33]:

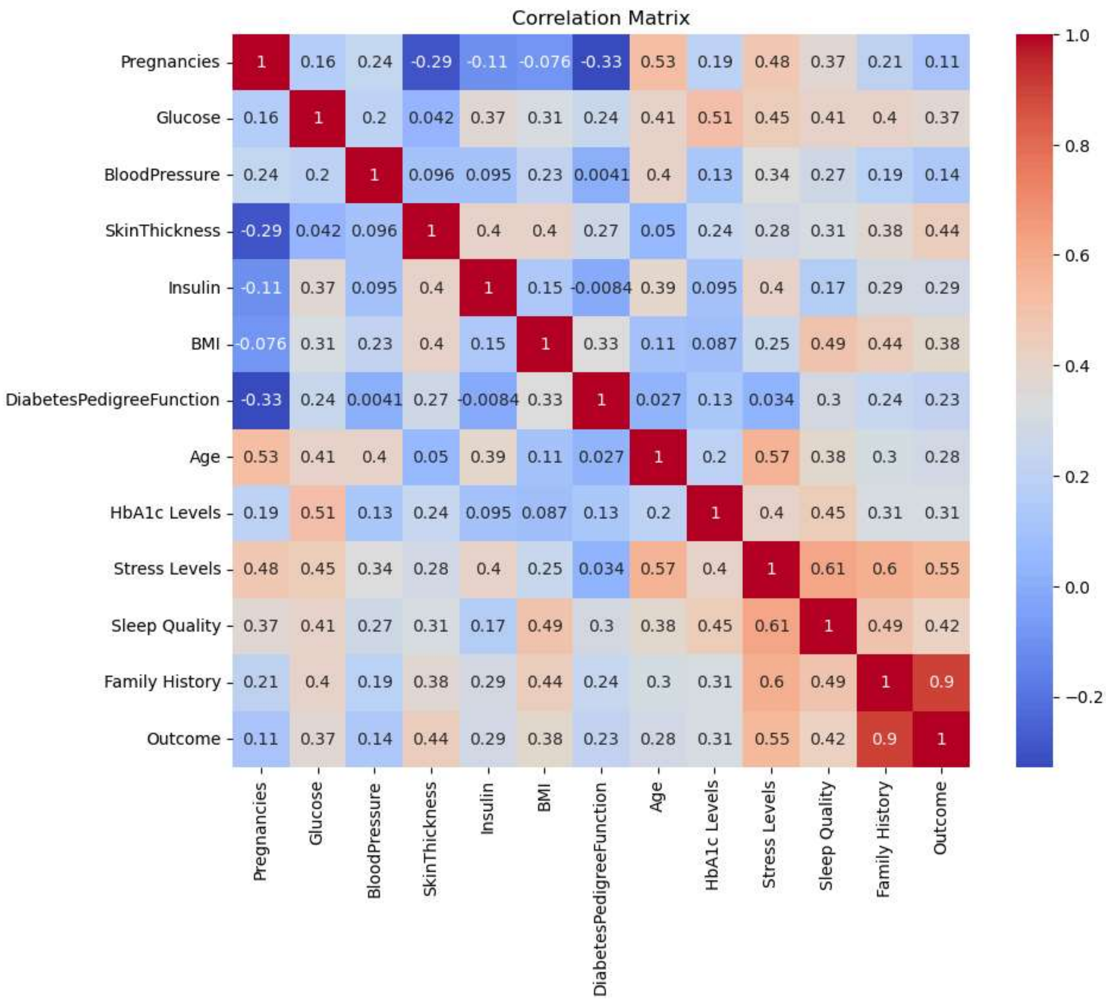
```
data.fillna(data.mean(), inplace=True)


data.drop_duplicates(inplace=True)
```

In [34]:
```
from scipy.stats import zscore


numeric_columns = ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
                   'DiabetesPedigreeFunction', 'Age', 'HbA1c Levels', 'Str
                   'Sleep Quality', 'Family History']
z_scores = np.abs(zscore(data[numeric_columns]))
print("Z-scores:\n", z_scores)
threshold = 3
outliers = np.where(z_scores > threshold)
data_cleaned = data[(z_scores < threshold).all(axis=1)]

print("Original data shape:", data.shape)
print("Cleaned data shape:", data_cleaned.shape)


#scaler = StandardScaler()
#data_cleaned[numeric_columns] = scaler.fit_transform(data_cleaned[numeric

#print("Normalized data:\n", data_cleaned)
```

```
Z-scores:
      Glucose  BloodPressure  SkinThickness   Insulin       BMI  \
0    0.587063       0.158442       0.951645  0.551267  0.075143
1    1.224046       0.147983       0.574095  0.551267  0.543798
2    1.593235       0.250124       1.250734  0.551267  0.930706
3    1.109055       0.147983       0.196544  0.151654  0.367930
4    0.270838       1.475825       0.951645  0.705017  1.390743
..        ...            ...            ...       ...       ...
447  0.032673       0.965116       0.306857  0.026709  0.825185
484  0.903289       0.250124       1.250734  0.551267  0.450002
485  1.506992       0.147983       1.203346  0.551267  1.261774
499  1.626514       0.056301       0.448244  0.551267  0.379655
508  0.529568       0.658691       1.250734  0.551267  0.180338

     DiabetesPedigreeFunction       Age  HbA1c Levels  Stress Levels  \
0                    0.138874  1.357392      1.030893       0.522738
1                    0.504850  0.387766      1.722939       1.556818
2                    0.243830  0.295916      0.814562       1.545422
3                    0.934000  1.306271      0.800211       0.522738
4                    4.012882  0.204066      2.414984       0.511342
..                        ...       ...           ...            ...
447                  0.096691  0.009355      0.338847       0.522738
484                  0.042084  0.009355      0.800211       0.511342
485                  0.042084  0.938869      2.198653       0.522738
499                  0.042084  1.214420      1.261575       0.522738
508                  0.042084  0.571467      0.569529       1.556818

     Sleep Quality  Family History
0         1.103440        1.063757
1         1.646672        0.940064
2         0.271616        1.063757
3         1.646672        0.940064
4         1.103440        1.063757
..             ...             ...
447       0.271616        0.940064
484       0.271616        0.940064
485       1.103440        1.063757
499       0.271616        0.940064
508       1.646672        0.940064

[81 rows x 11 columns]
Original data shape: (81, 13)
Cleaned data shape: (71, 13)
```

In [35]:

```
correlations = data.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlations, cmap='coolwarm', annot=True)
plt.title('Correlation Matrix')
plt.show()


plt.figure(figsize=(10, 6))
```

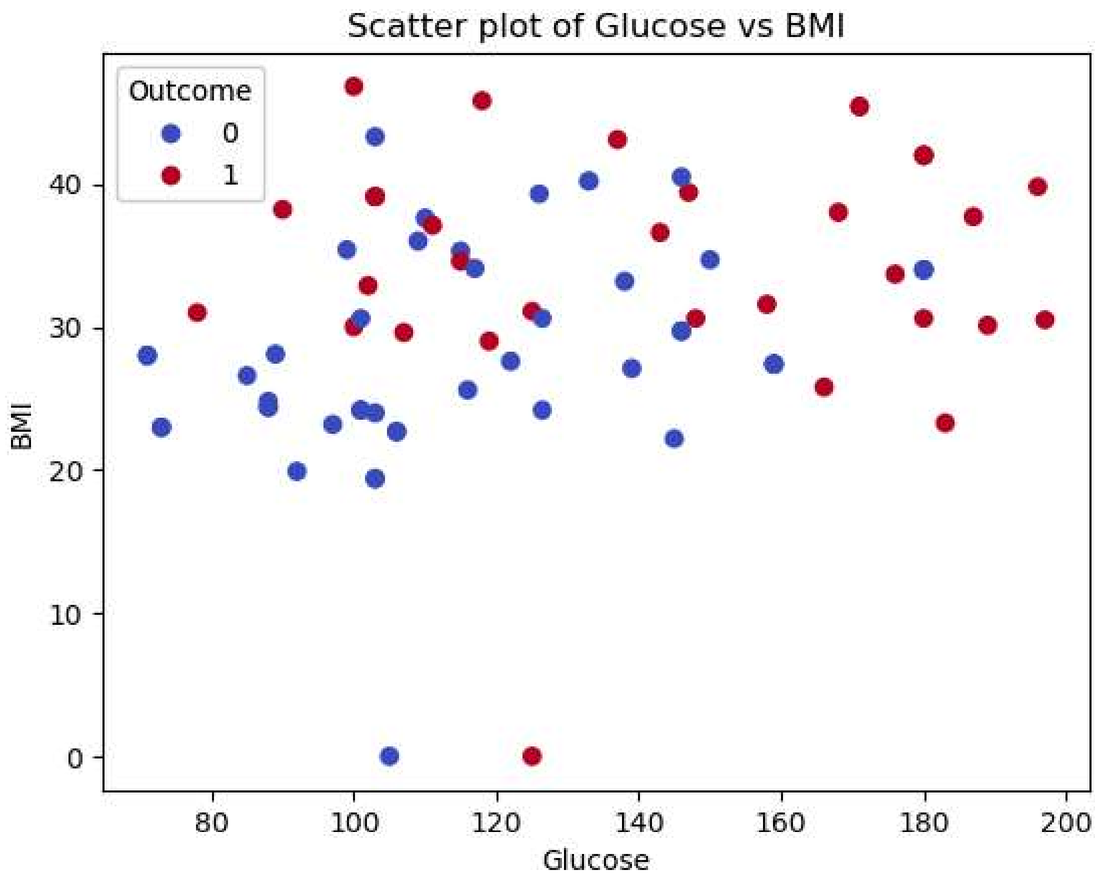


Out[35]: <Figure size 1000x600 with 0 Axes>

<Figure size 1000x600 with 0 Axes>

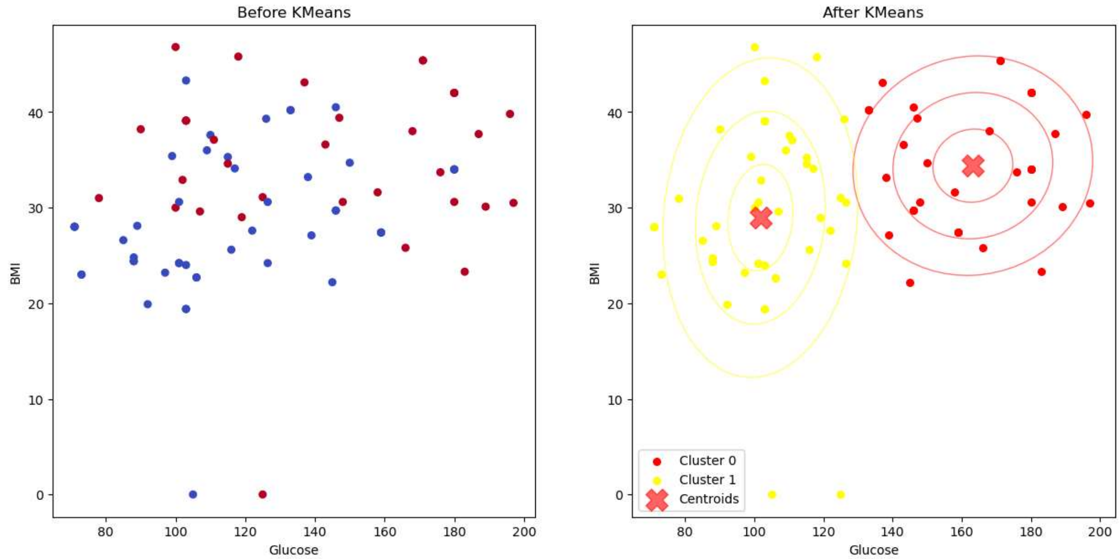
In [36]:

```
# Create scatter plot with different colors for each outcome
scatter = plt.scatter(data['Glucose'], data['BMI'], c=data['Outcome'], cma
plt.xlabel('Glucose')
plt.ylabel('BMI')
plt.title('Scatter plot of Glucose vs BMI')

# Create a custom legend
legend1 = plt.legend(*scatter.legend_elements(), title='Outcome')
plt.gca().add_artist(legend1)
plt.show()
```



In [37]:

```
np.random.seed(42)

kmeans = KMeans(n_clusters=2, random_state=42)
data['Cluster'] = kmeans.fit_predict(data[['Glucose', 'BMI']])
centers = kmeans.cluster_centers_

# Function to draw an ellipse around clusters
def draw_ellipse(position, covariance, ax=None, **kwargs):
    ax = ax or plt.gca()
    if covariance.shape == (2, 2):
        U, s, Vt = np.linalg.svd(covariance)
        angle = np.degrees(np.arctan2(U[1, 0], U[0, 0]))
        width, height = 1.2 * np.sqrt(s)
    else:
        angle = 0
        width, height = 1.2 * np.sqrt(covariance)

    for nsig in range(1, 4):
        ax.add_patch(Ellipse(position, nsig * width, nsig * height,
                             angle,facecolor='none', **kwargs))

# Plotting
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 7))

# Before KMeans
ax1.scatter(data['Glucose'], data['BMI'], c=data['Outcome'],cmap='coolwarm
ax1.set_title('Before KMeans')
ax1.set_xlabel('Glucose')
ax1.set_ylabel('BMI')


# After KMeans
colors = ['red','yellow']
for i in range(2):
    points = data[data['Cluster'] == i]
    ax2.scatter(points['Glucose'], points['BMI'], s=30, color=colors[i], l
    draw_ellipse(centers[i], np.cov(points[['Glucose', 'BMI']].values.T),

ax2.scatter(centers[:, 0], centers[:, 1], c='red', s=300, alpha=0.6, marke
ax2.set_title('After KMeans')
ax2.set_xlabel('Glucose')
ax2.set_ylabel('BMI')
ax2.legend()
plt.show()
```



In [38]:

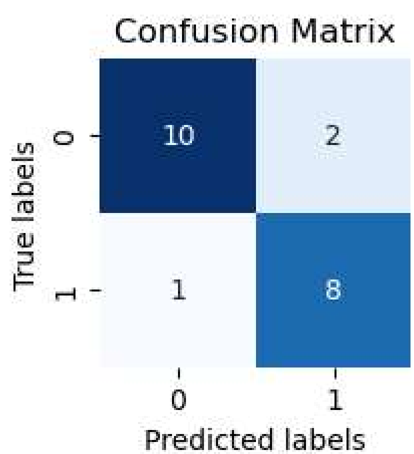
```
X = data.drop(['Outcome', 'Pregnancies',], axis=1)
y = data['Outcome']

from sklearn.feature_selection import mutual_info_classif

# Calculate mutual information
mutual_info = mutual_info_classif(X, y)
mutual_info_series = pd.Series(mutual_info, index=X.columns)

# Sort and display mutual information scores
mutual_info_series = mutual_info_series.sort_values(ascending=False)
print("Mutual information scores:\n", mutual_info_series)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
```

```
Mutual information scores:
 Family History              0.523271
BMI                         0.236591
SkinThickness               0.189547
Age                         0.180113
HbA1c Levels                0.125514
Stress Levels               0.105741
Sleep Quality               0.101695
BloodPressure               0.061102
Insulin                     0.058033
DiabetesPedigreeFunction    0.054335
Cluster                     0.046899
Glucose                     0.015636
dtype: float64
```

In [39]:

```
log_reg_model = LogisticRegression(max_iter=1000)  # Increase max_iter
log_reg_model.fit(X_train, y_train)
log_reg_accuracy = log_reg_model.score(X_test, y_test)
log_reg_pred = log_reg_model.predict(X_test)
log_reg_cm = metrics.confusion_matrix(y_test, log_reg_pred)
print("Logistic regression accuracy",log_reg_accuracy)
print("confusion matrix of logistic regression",log_reg_cm)
print("prediction of logistic regression",log_reg_pred)
print("Intercept: ", log_reg_model.intercept_)
print("Coefficients: ", log_reg_model.coef_)
plt.figure(figsize=(2, 2))
sns.heatmap(log_reg_cm, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.xlabel("Predicted labels")
plt.ylabel("True labels")
plt.title("Confusion Matrix")
plt.show()
```

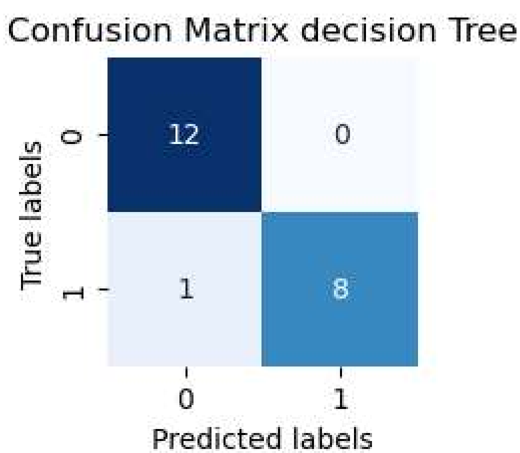
```
Logistic regression accuracy 0.8571428571428571
confusion matrix of logistic regression [[10  2]
 [ 1  8]]
prediction of logistic regression [1 1 1 1 1 0 0 1 1 0 0 0 0 0 1 1 1 0 0
0 0]
Intercept:  [-7.61590177]
Coefficients:  [[ 0.02046365 -0.03147864  0.05585701 -0.00270719  0.0871
8083 -0.33242102
   0.0550702  -0.14273518  0.8026947  -0.34932055  2.40770665  0.8953890
2]]
```



In [40]:

```
# Decision Tree
dt_model = DecisionTreeClassifier()
dt_model.fit(X_train, y_train)
dt_accuracy = dt_model.score(X_test, y_test)
dt_pred = dt_model.predict(X_test)
dt_cm = metrics.confusion_matrix(y_test, dt_pred)
print("confusion matrix of Decision Tree",dt_cm)
print("Prediction of Decision Tree",dt_pred)
print("accuracy of decision tree",dt_accuracy)
plt.figure(figsize=(2, 2))
sns.heatmap(dt_cm, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.xlabel("Predicted labels")
plt.ylabel("True labels")
plt.title("Confusion Matrix decision Tree")
plt.show()
```

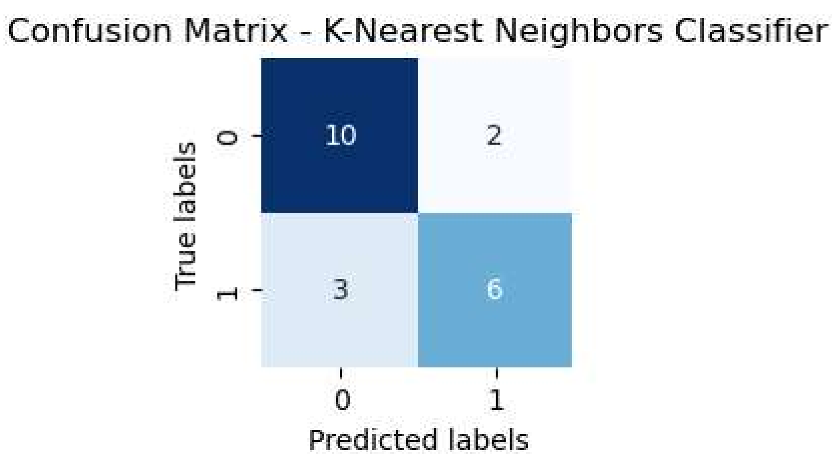
```
confusion matrix of Decision Tree [[12  0]
 [ 1  8]]
Prediction of Decision Tree [0 1 1 1 0 0 0 1 1 0 0 0 0 0 1 1 1 0 0 0 0]
accuracy of decision tree 0.9523809523809523
```



In [41]:

```
import warnings
warnings.filterwarnings("ignore", message=".*the default behavior of `mode
knn_model = KNeighborsClassifier()

knn_model.fit(X_train, y_train)
knn_accuracy = knn_model.score(X_test, y_test)
knn_pred = knn_model.predict(X_test)
knn_cm = metrics.confusion_matrix(y_test, knn_pred)
print("confusion matrix of KNeighour Classification")
print(knn_cm)
plt.figure(figsize=(2, 2))
print("accuracy of KNeighour Classification",knn_accuracy)
print("KNeighour Classification prediction",knn_pred)
sns.heatmap(knn_cm, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.xlabel("Predicted labels")
plt.ylabel("True labels")
plt.title("Confusion Matrix - K-Nearest Neighbors Classifier")
plt.show()
```

```
confusion matrix of KNeighour Classification
[[10  2]
 [ 3  6]]
accuracy of KNeighour Classification 0.7619047619047619
KNeighour Classification prediction [0 0 0 1 0 1 0 1 1 0 0 0 0 1 1 1 1 0
0 0 0]
```



In [42]:

```
models_accuracy = {'Logistic Regression': log_reg_accuracy,
                   'Decision Tree': dt_accuracy,
                   'KNN': knn_accuracy}
print(models_accuracy)
```

```
{'Logistic Regression': 0.8571428571428571, 'Decision Tree': 0.952380952
3809523, 'KNN': 0.7619047619047619}
```

In [43]:

```
import warnings
warnings.filterwarnings("ignore", message=".*the default behavior of `mode

# Assuming the models and accuracies are already defined somewhere earlier
X_test = X_test[X_train.columns]
best_model_name = max(models_accuracy, key=models_accuracy.get)
print(f"Best Model: {best_model_name}")

# Select the best model
if best_model_name == 'Logistic Regression':
    best_model = log_reg_model
elif best_model_name == 'Decision Tree':
    best_model = dt_model
else:
    best_model = knn_model

# Print confusion matrix
""""print(f"Confusion Matrix for {best_model_name}:")
if best_model_name == 'Logistic Regression':
    print(log_reg_cm)
elif best_model_name == 'Decision Tree':
    print(dt_cm)
else:
    print(knn_cm)"""



def risk_level(glucose):
    if glucose > 140:
        return "High"
    elif 100 < glucose <= 140:
        return "Medium"
    else:
        return "Low"
    data['Risk_Level'] = data['Glucose'].apply(risk_level)

# Define function to predict diabetes
def predict_diabetes(model):
    Pregnancies = float(input("Enter Pregnancies: "))
    glucose = float(input("Enter Glucose level: "))
    blood_pressure = float(input("Enter Blood Pressure: "))
    skin_thickness = float(input("Enter Skin Thickness: "))
    insulin = float(input("Enter Insulin: "))
    bmi = float(input("Enter BMI: "))
    diabetes_pedigree = float(input("Enter Diabetes Pedigree Function: "))
    age = float(input("Enter Age: "))
    hba1c_levels = float(input("Enter HbA1c Levels: "))
    stress_levels = float(input("Enter Stress Levels: "))
    sleep_quality = float(input("Enter Sleep Quality: "))
    family_history = float(input("Enter Family History: "))

    new_data = [[Pregnancies,glucose, blood_pressure, skin_thickness, insu

    prediction = model.predict(new_data)
    risk = risk_level(glucose)

    if prediction[0] == 1:
```

```
        print("Prediction: Person has diabetes")
        print("Risk Level:", risk)
    else:
        print("Prediction: Person does not have diabetes")

import warnings
warnings.filterwarnings("ignore", message=".*the default behavior of `mode


predict_diabetes(best_model)

X_test['Predicted_Diabetes'] = best_model.predict(X_test)
predicted_data = pd.concat([X_test, y_test], axis=1)
predicted_data.to_csv("dm_project_final_predicted.csv", index=False)
```

```
Best Model: Decision Tree
Enter Pregnancies: 1
Enter Glucose level: 140
Enter Blood Pressure: 90
Enter Skin Thickness: 40
Enter Insulin: 0
Enter BMI: 40
Enter Diabetes Pedigree Function: 0.790
Enter Age: 50
Enter HbA1c Levels: 7.8
Enter Stress Levels: 8
Enter Sleep Quality: 9
Enter Family History: 1
Prediction: Person has diabetes
Risk Level: Medium

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\base.py:450: UserWarn
ing: X does not have valid feature names, but DecisionTreeClassifier was
fitted with feature names
  warnings.warn(
```