FINAL Nicole Smitheman ECS784P coursework

March 18, 2025

[35]:
```
#Importing desired modules
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics

#Load and display first few lines of file
cvd_df = pd.read_csv("heart_disease_risk.csv")
print(cvd_df.head())
```

```
   Chest_Pain  Shortness_of_Breath  Fatigue  Palpitations  Dizziness  \
0         0.0                  0.0      0.0           1.0        0.0
1         0.0                  1.0      0.0           1.0        0.0
2         1.0                  0.0      0.0           1.0        0.0
3         1.0                  1.0      0.0           1.0        0.0
4         0.0                  0.0      1.0           0.0        1.0

   Swelling  Pain_Arms_Jaw_Back  Cold_Sweats_Nausea  High_BP  \
0       0.0                 0.0                 0.0      0.0
1       0.0                 0.0                 0.0      1.0
2       0.0                 0.0                 0.0      1.0
3       0.0                 1.0                 1.0      1.0
4       0.0                 0.0                 0.0      0.0

   High_Cholesterol  Diabetes  Smoking  Obesity  Sedentary_Lifestyle  \
0               0.0       0.0      1.0      0.0                  1.0
1               0.0       0.0      1.0      1.0                  0.0
2               1.0       0.0      1.0      1.0                  1.0
3               0.0       1.0      1.0      0.0                  1.0
4               0.0       1.0      0.0      0.0                  0.0

   Family_History  Chronic_Stress  Gender   Age  Heart_Risk
```

```
0             0.0             0.0     0.0  48.0         0.0
1             0.0             0.0     0.0  46.0         0.0
2             0.0             0.0     1.0  66.0         0.0
3             1.0             1.0     1.0  60.0         1.0
4             0.0             0.0     0.0  69.0         0.0
```

[36]:
```
#Checking for null values - no null values present
print(cvd_df.isnull().sum())
```

```
Chest_Pain             0
Shortness_of_Breath    0
Fatigue                0
Palpitations           0
Dizziness              0
Swelling               0
Pain_Arms_Jaw_Back     0
Cold_Sweats_Nausea     0
High_BP                0
High_Cholesterol       0
Diabetes               0
Smoking                0
Obesity                0
Sedentary_Lifestyle    0
Family_History         0
Chronic_Stress         0
Gender                 0
Age                    0
Heart_Risk             0
dtype: int64
```

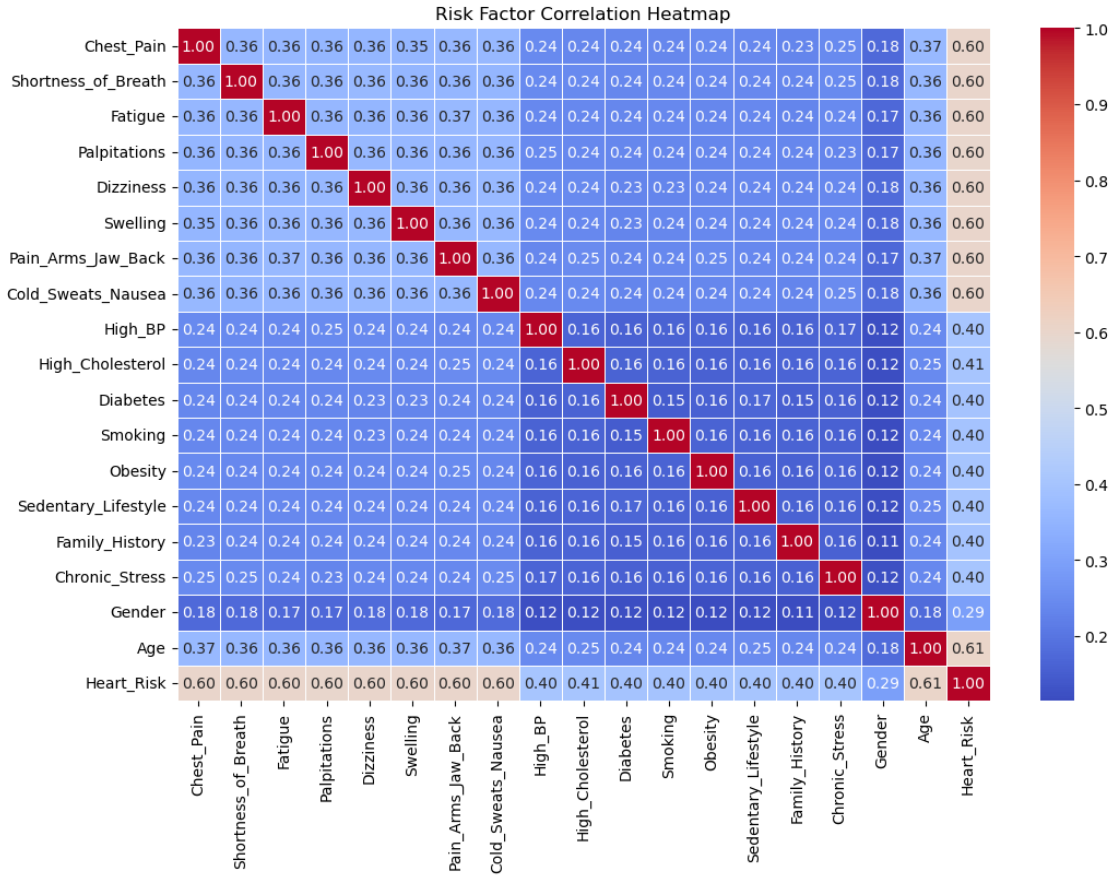
[37]:
```
#Checking the statistics of the information in the dataset - datatype and␣
 ↪statistics
print(cvd_df.describe())
print(cvd_df.info())
```

```
         Chest_Pain  Shortness_of_Breath       Fatigue  Palpitations  \
count  70000.000000         70000.000000  70000.000000  70000.000000
mean       0.499229             0.500586      0.498571      0.498729
std        0.500003             0.500003      0.500002      0.500002
min        0.000000             0.000000      0.000000      0.000000
25%        0.000000             0.000000      0.000000      0.000000
50%        0.000000             1.000000      0.000000      0.000000
75%        1.000000             1.000000      1.000000      1.000000
max        1.000000             1.000000      1.000000      1.000000

          Dizziness      Swelling  Pain_Arms_Jaw_Back  Cold_Sweats_Nausea  \
count  70000.000000  70000.000000        70000.000000        70000.000000
mean       0.501414      0.498929            0.501500            0.502457
std        0.500002      0.500002            0.500001            0.499998
```

```
min        0.000000      0.000000            0.000000            0.000000
25%        0.000000      0.000000            0.000000            0.000000
50%        1.000000      0.000000            1.000000            1.000000
75%        1.000000      1.000000            1.000000            1.000000
max        1.000000      1.000000            1.000000            1.000000

            High_BP  High_Cholesterol      Diabetes       Smoking  \
count  70000.000000      70000.000000  70000.000000  70000.000000
mean       0.497429          0.499214      0.500643      0.502971
std        0.499997          0.500003      0.500003      0.499995
min        0.000000          0.000000      0.000000      0.000000
25%        0.000000          0.000000      0.000000      0.000000
50%        0.000000          0.000000      1.000000      1.000000
75%        1.000000          1.000000      1.000000      1.000000
max        1.000000          1.000000      1.000000      1.000000

            Obesity  Sedentary_Lifestyle  Family_History  Chronic_Stress  \
count  70000.000000         70000.000000    70000.000000    70000.000000
mean       0.499157             0.503543        0.497629        0.499957
std        0.500003             0.499991        0.499998        0.500004
min        0.000000             0.000000        0.000000        0.000000
25%        0.000000             0.000000        0.000000        0.000000
50%        0.000000             1.000000        0.000000        0.000000
75%        1.000000             1.000000        1.000000        1.000000
max        1.000000             1.000000        1.000000        1.000000

             Gender           Age    Heart_Risk
count  70000.000000  70000.000000  70000.000000
mean       0.548929     54.461986      0.500000
std        0.497604     16.410794      0.500004
min        0.000000     20.000000      0.000000
25%        0.000000     45.000000      0.000000
50%        1.000000     56.000000      0.500000
75%        1.000000     67.000000      1.000000
max        1.000000     84.000000      1.000000
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 70000 entries, 0 to 69999
Data columns (total 19 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Chest_Pain           70000 non-null  float64
 1   Shortness_of_Breath  70000 non-null  float64
 2   Fatigue              70000 non-null  float64
 3   Palpitations         70000 non-null  float64
 4   Dizziness            70000 non-null  float64
 5   Swelling             70000 non-null  float64
 6   Pain_Arms_Jaw_Back   70000 non-null  float64
 7   Cold_Sweats_Nausea   70000 non-null  float64
```

```
 8   High_BP                 70000 non-null  float64
 9   High_Cholesterol        70000 non-null  float64
 10  Diabetes                70000 non-null  float64
 11  Smoking                 70000 non-null  float64
 12  Obesity                 70000 non-null  float64
 13  Sedentary_Lifestyle     70000 non-null  float64
 14  Family_History          70000 non-null  float64
 15  Chronic_Stress          70000 non-null  float64
 16  Gender                  70000 non-null  float64
 17  Age                     70000 non-null  float64
 18  Heart_Risk              70000 non-null  float64
dtypes: float64(19)
memory usage: 10.1 MB
None
```

[38]:
```python
#Correlation heatmap
corr_matrix = cvd_df.corr()

plt.figure(figsize=(12, 8))
sns.heatmap(corr_matrix, annot=True, cmap="coolwarm", fmt=".2f", linewidths=0.5)
plt.title("Risk Factor Correlation Heatmap")
plt.show()
```



[39]:
```
#Defining the X and Y axis variables
X = cvd_df.drop('Heart_Risk', axis=1)
y = cvd_df['Heart_Risk']

#Feature scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

[40]:
```
#Splitting data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,␣
 ↪random_state=14)
```

[41]:
```
# Implementing Logistic Regression model and fitting with data
logreg = LogisticRegression(random_state=14)
logreg.fit(X_train, y_train)
```

[41]:
```
LogisticRegression(random_state=14)
```

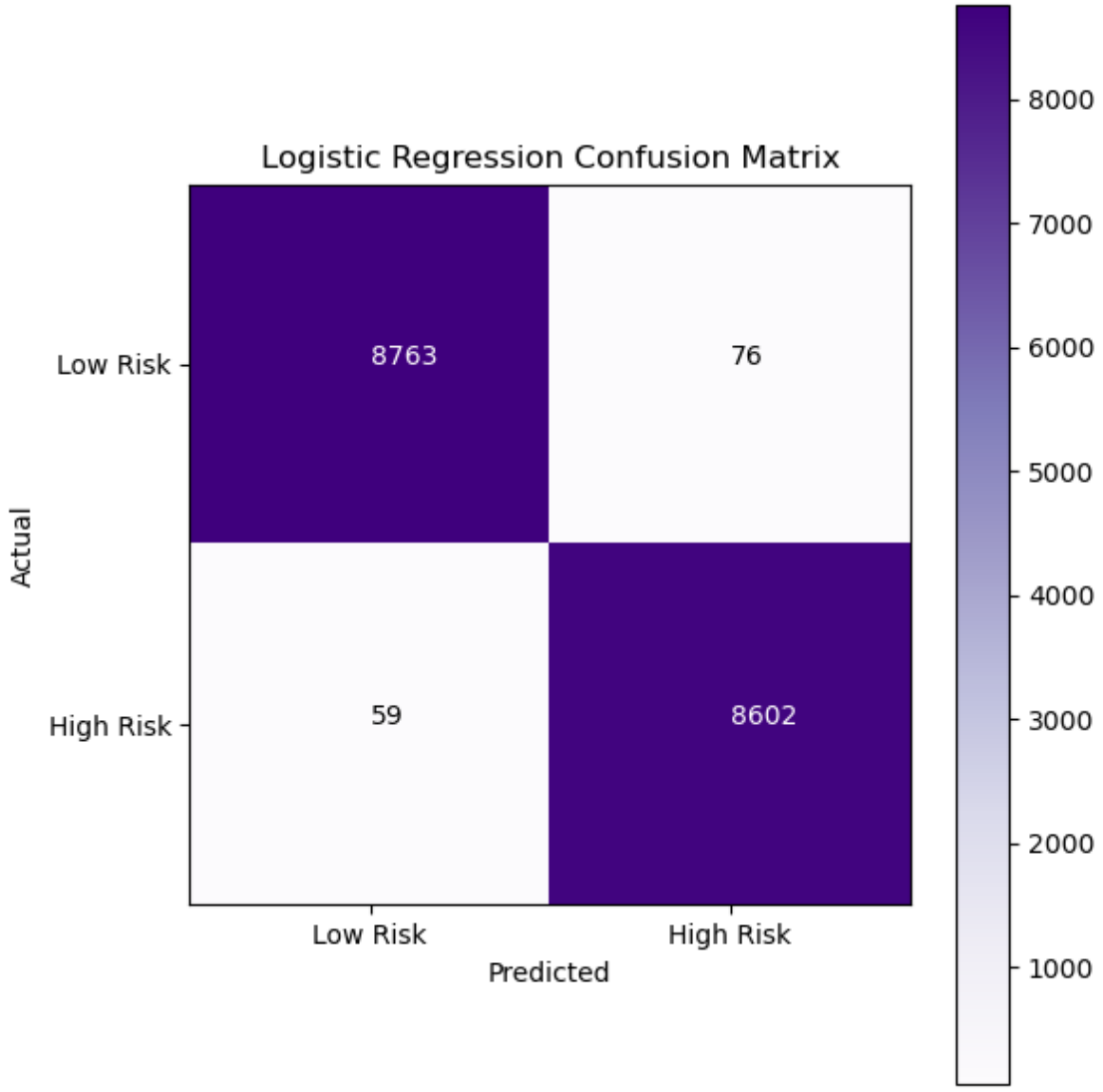
[42]:
```
# Predict probabilities
y_prob_log = logreg.predict_proba(X_test)[:, 1]
# Compute the ROC AUC score
roc = metrics.roc_auc_score(y_test, y_prob_log)

# Print the score
print(f"Logistic Regression ROC: {roc:}")

#Checking for possible model overfitting
print(f"Accuracy of training set: {logreg.score(X_train, y_train):}")
print(f"Accuracy of test set: {logreg.score(X_test, y_test):}")
```

```
Logistic Regression ROC: 0.9996845649167505
Accuracy of training set: 0.9916761904761905
Accuracy of test set: 0.9922857142857143
```

[45]:
```
#Predicting and evaluating model performance
#Using a confusion matrix
y_pred = logreg.predict(X_test)
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
print(cnf_matrix)

# Plotting the matrix
plt.figure(figsize=(6,6))
plt.imshow(cnf_matrix, interpolation='nearest', cmap='Purples')
plt.title("Logistic Regression Confusion Matrix")
plt.xlabel("Predicted")
```

```
plt.ylabel("Actual")
plt.colorbar()

threshold = cnf_matrix.max() / 2.
for i in range(cnf_matrix.shape[0]):
    for j in range(cnf_matrix.shape[1]):
        plt.text(j, i, format(cnf_matrix[i, j]),
                 color="white" if cnf_matrix[i, j] > threshold else "black")

# Set axis ticks
plt.xticks([0, 1], ['Low Risk', 'High Risk'])
plt.yticks([0, 1], ['Low Risk', 'High Risk'])

# Display
plt.tight_layout()
plt.show()
plt.close()

#Classification report
print(metrics.classification_report(y_test, y_pred))
```

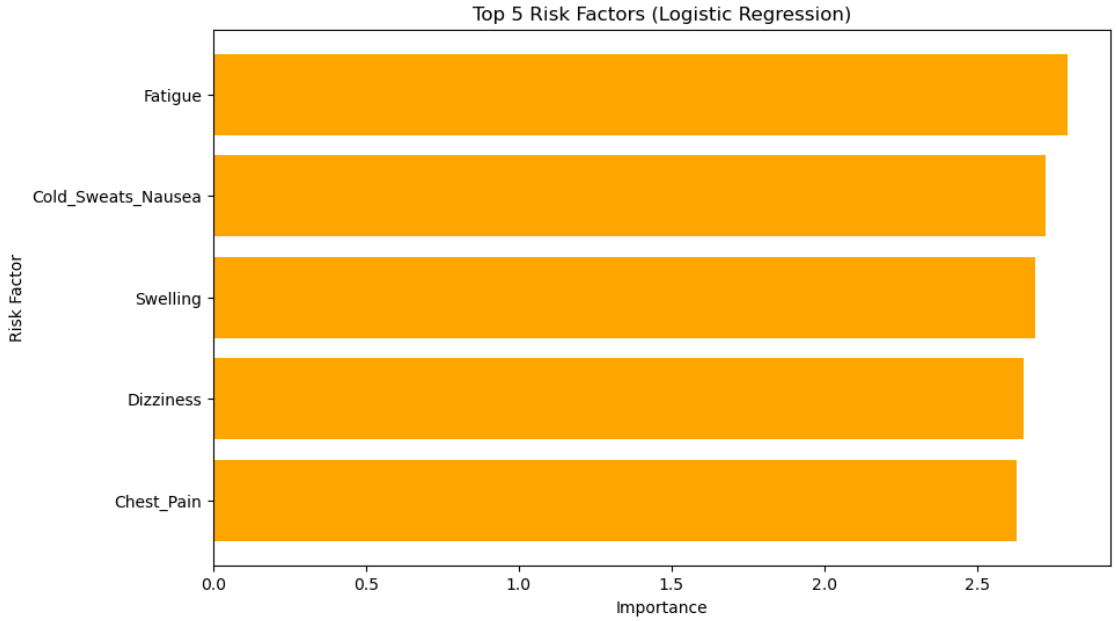
```
[[8763   76]
 [  59 8602]]
```



	precision	recall	f1-score	support
0.0	0.99	0.99	0.99	8839
1.0	0.99	0.99	0.99	8661
accuracy			0.99	17500
macro avg	0.99	0.99	0.99	17500
weighted avg	0.99	0.99	0.99	17500

[46]:
```
#Bar chart of most important factors with Logistic Regression model
# Get absolute coefficients
log_reg_importance = pd.DataFrame({
```

```
    "Risk Factor": X.columns,
    "Importance": np.abs(logreg.coef_[0])
}).sort_values(by="Importance", ascending=False)

# Select the top 5 most important features
top = log_reg_importance[:5]

# Plot the feature importance
plt.figure(figsize=(10, 6))
plt.barh(top["Risk Factor"], top["Importance"], color="orange")
plt.xlabel("Importance")
plt.ylabel("Risk Factor")
plt.title("Top 5 Risk Factors (Logistic Regression)")
plt.gca().invert_yaxis()
plt.show()
```

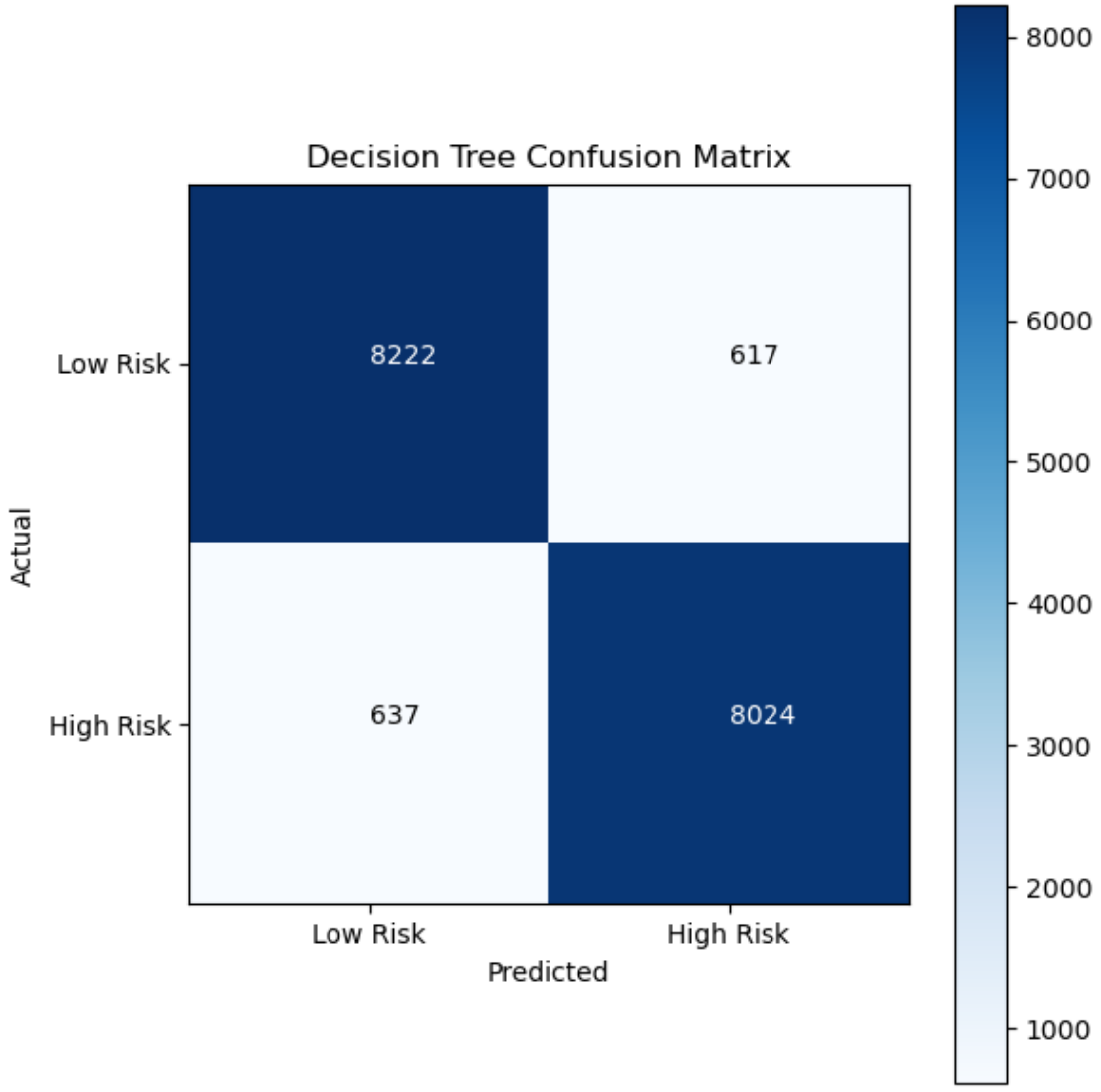


[47]:
```
#Creating a Risk Decision Tree
dtree = DecisionTreeClassifier(max_depth=4, random_state=14)
dtree.fit(X_train, y_train)
y_pred = dtree.predict(X_test)

print("\nDecision Tree Accuracy:", metrics.accuracy_score(y_test, y_pred))
print(metrics.classification_report(y_test, y_pred))

#Using a confusion matrix
cnf_matrix_tree = metrics.confusion_matrix(y_test, y_pred)
```

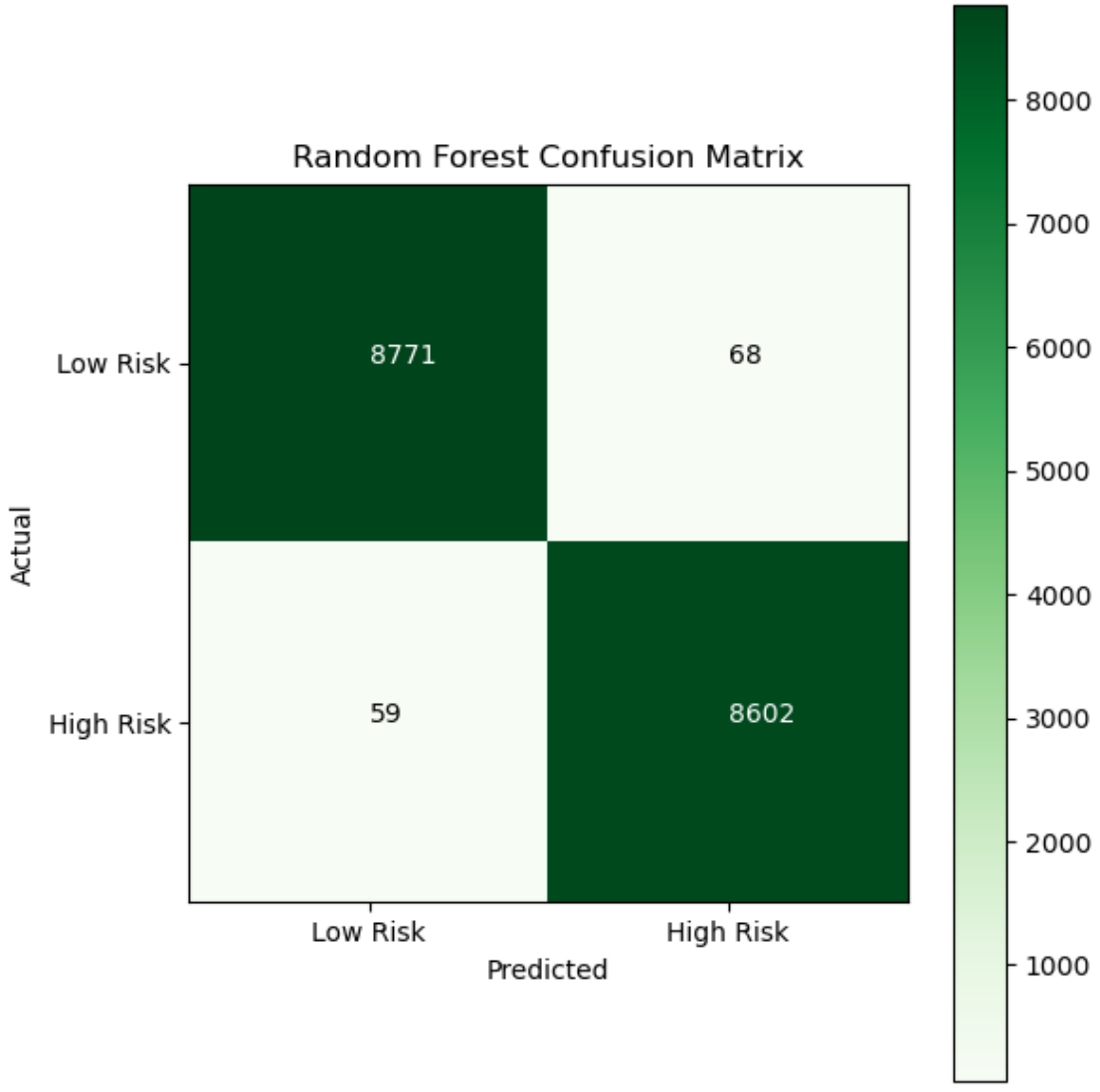
```
plt.figure(figsize=(6,6))
plt.imshow(cnf_matrix_tree, interpolation='nearest', cmap='Blues')
plt.title("Decision Tree Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.colorbar()

threshold = cnf_matrix_tree.max() / 2.
for i in range(cnf_matrix_tree.shape[0]):
    for j in range(cnf_matrix_tree.shape[1]):
        plt.text(j, i, format(cnf_matrix_tree[i, j]),
                 color="white" if cnf_matrix_tree[i, j] > threshold else␣
 ↪"black")

plt.xticks([0, 1], ['Low Risk', 'High Risk'])
plt.yticks([0, 1], ['Low Risk', 'High Risk'])
plt.tight_layout()
plt.show()


#Random Forest
rf = RandomForestClassifier(random_state=14)
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)

print("\nRandom Forest Accuracy:", metrics.accuracy_score(y_test, y_pred))
print(metrics.classification_report(y_test, y_pred))

#Using a confusion matrix
cnf_matrix_rf = metrics.confusion_matrix(y_test, y_pred)

plt.figure(figsize=(6,6))
plt.imshow(cnf_matrix_rf, interpolation='nearest', cmap='Greens')
plt.title("Random Forest Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.colorbar()

threshold = cnf_matrix_rf.max() / 2.
for i in range(cnf_matrix_rf.shape[0]):
    for j in range(cnf_matrix_rf.shape[1]):
        plt.text(j, i, format(cnf_matrix_rf[i, j]),
                 color="white" if cnf_matrix_rf[i, j] > threshold else "black")

plt.xticks([0, 1], ['Low Risk', 'High Risk'])
plt.yticks([0, 1], ['Low Risk', 'High Risk'])
```

```
plt.tight_layout()
plt.show()
```

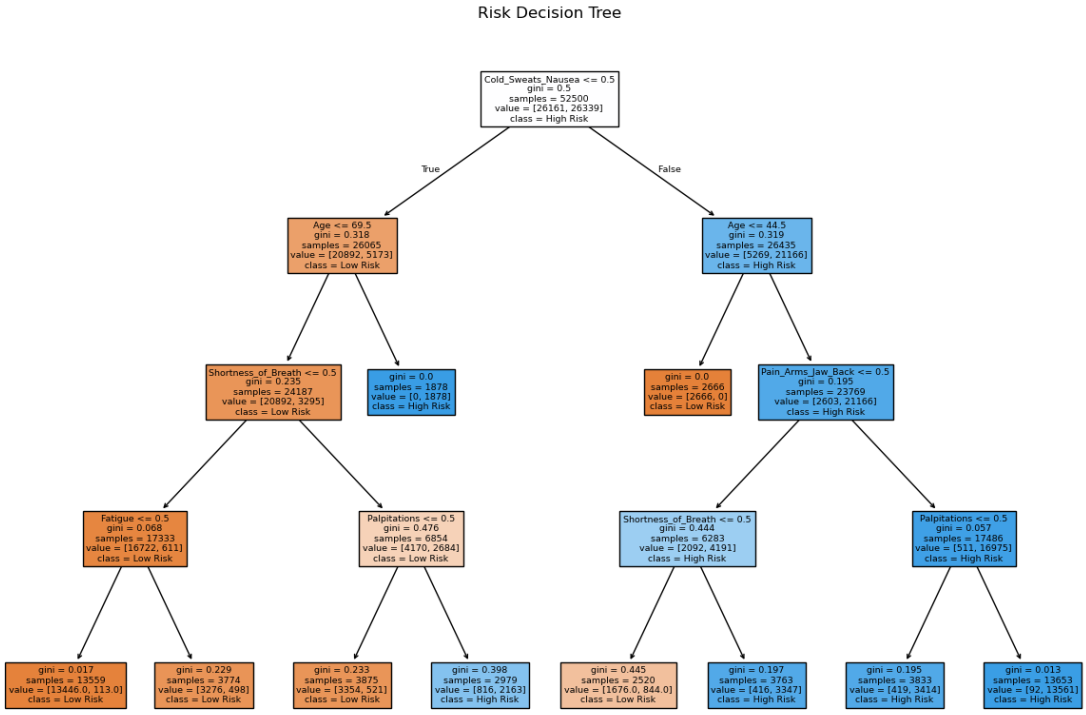
```
Decision Tree Accuracy: 0.9283428571428571
              precision    recall  f1-score   support

         0.0       0.93      0.93      0.93      8839
         1.0       0.93      0.93      0.93      8661

    accuracy                           0.93     17500
   macro avg       0.93      0.93      0.93     17500
weighted avg       0.93      0.93      0.93     17500
```



```
Random Forest Accuracy: 0.9927428571428571
              precision    recall  f1-score   support

         0.0       0.99      0.99      0.99      8839
         1.0       0.99      0.99      0.99      8661

    accuracy                           0.99     17500
   macro avg       0.99      0.99      0.99     17500
weighted avg       0.99      0.99      0.99     17500
```

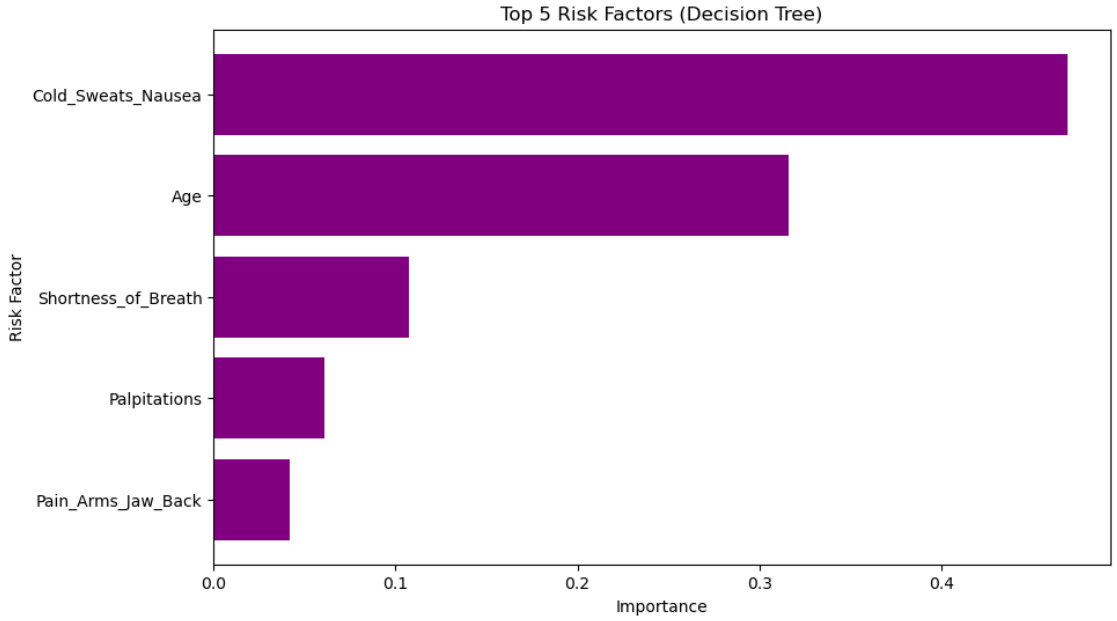


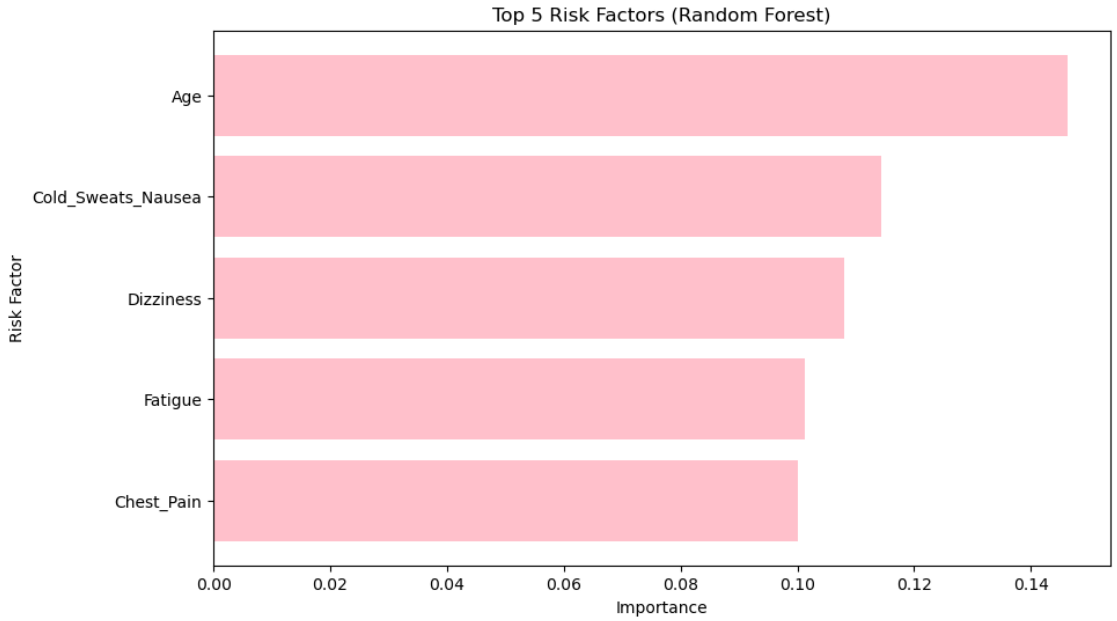
[48]:
```
# Visualising the decision tree
plt.figure(figsize=(15, 10))
plot_tree(dtree, filled=True, feature_names=X.columns, class_names=["Low Risk",␣
 ↪"High Risk"])
plt.title("Risk Decision Tree")
plt.show()
```



[49]:
```
# Compute risk factor importance for Decision Tree
dtree_importance = pd.DataFrame({
    "Risk factor": X.columns,
    "Importance": dtree.feature_importances_
}).sort_values(by="Importance", ascending=False)

# Compute risk factor importance for Random Forest
rf_importance = pd.DataFrame({
    "Risk factor": X.columns,
    "Importance": rf.feature_importances_
}).sort_values(by="Importance", ascending=False)

#decison tree bar chart
plt.figure(figsize=(10, 6))
```

```
plt.barh(dtree_importance["Risk factor"][:5], dtree_importance["Importance"][:
 ↪5], color="purple")
plt.xlabel("Importance")
plt.ylabel("Risk Factor")
plt.title("Top 5 Risk Factors (Decision Tree)")
plt.gca().invert_yaxis()
plt.show()

#random forest bar chart
plt.figure(figsize=(10, 6))
plt.barh(rf_importance["Risk factor"][:5], rf_importance["Importance"][:5],␣
 ↪color="pink")
plt.xlabel("Importance")
plt.ylabel("Risk Factor")
plt.title("Top 5 Risk Factors (Random Forest)")
plt.gca().invert_yaxis()
plt.show()
```




Top 5 Risk Factors (Random Forest)
Age
Cold_Sweats_Nausea
Dizziness
Fatigue
Chest_Pain
Risk Factor
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
Importance