```
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.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neural_network import MLPClassifier
```

```
df = pd.read_csv('dataset.csv')
```

```
df.head()
```

	Age	Gender	Smoking	Hx Smoking	Hx Radiothreapy	Thyroid Function	Physical Examination	Adenop
0	27	F	No	No	No	Euthyroid	Single nodular goiter-left	
1	34	F	No	Yes	No	Euthyroid	Multinodular goiter	
2	30	F	No	No	No	Euthyroid	Single nodular goiter-right	
3	62	F	No	No	No	Euthyroid	Single nodular goiter-right	
4	62	F	No	No	No	Euthyroid	Multinodular goiter	

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

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 383 entries, 0 to 382
Data columns (total 17 columns):
 #   Column                Non-Null Count  Dtype 
---  ------                --------------  ----- 
 0   Age                   383 non-null    int64 
 1   Gender                383 non-null    object
 2   Smoking               383 non-null    object
 3   Hx Smoking            383 non-null    object
 4   Hx Radiothreapy       383 non-null    object
 5   Thyroid Function      383 non-null    object
 6   Physical Examination  383 non-null    object
 7   Adenopathy            383 non-null    object
 8   Pathology             383 non-null    object
 9   Focality              383 non-null    object
 10  Risk                  383 non-null    object
 11  T                     383 non-null    object
 12  N                     383 non-null    object
 13  M                     383 non-null    object
 14  Stage                 383 non-null    object
 15  Response              383 non-null    object
 16  Recurred              383 non-null    object
dtypes: int64(1), object(16)
memory usage: 51.0+ KB
```

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

	0
Age	0
Gender	0
Smoking	0
Hx Smoking	0
Hx Radiothreapy	0
Thyroid Function	0
Physical Examination	0
Adenopathy	0
Pathology	0
Focality	0
Risk	0
T	0
N	0
M	0
Stage	0
Response	0
Recurred	0

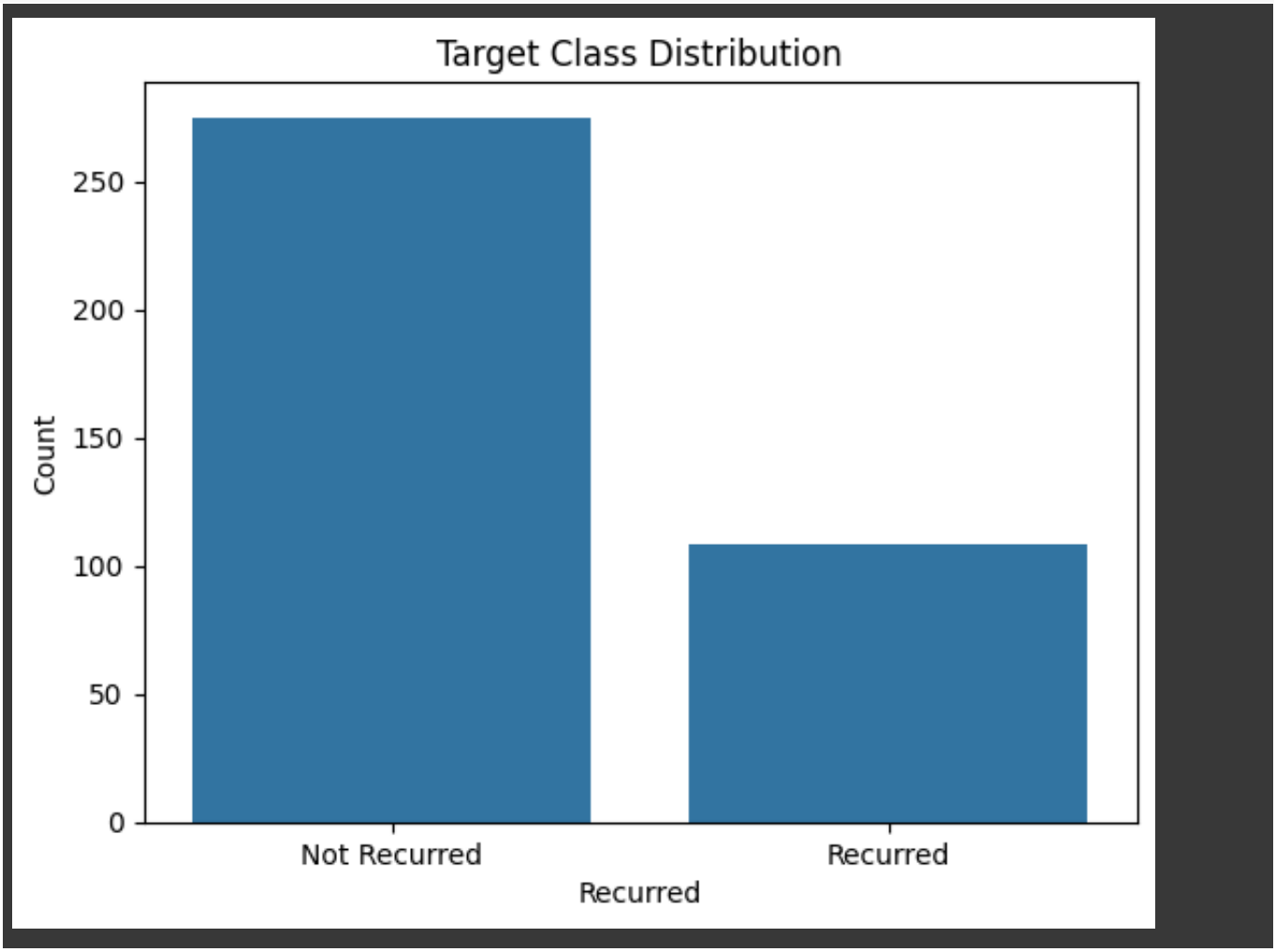
dtype: int64

```
labelencoder = LabelEncoder()

for col in df.columns:
    if df[col].dtype == 'object':
        df[col] = labelencoder.fit_transform(df[col])
```

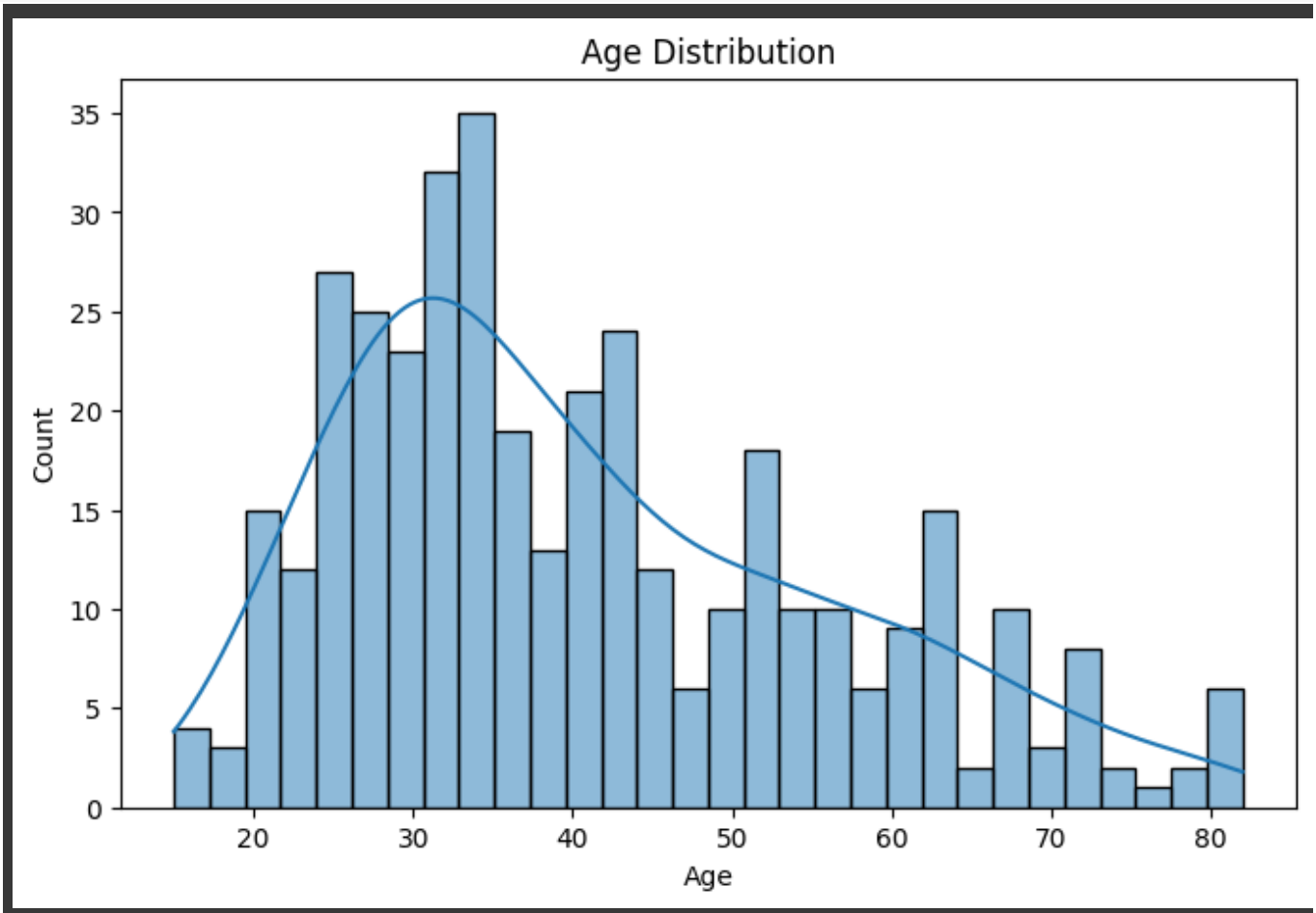
```
df.describe()
```

	Age	Gender	Smoking	Hx Smoking	Hx Radiothreapy	Thyroid Function	Ex
count	383.000000	383.000000	383.000000	383.000000	383.000000	383.000000	
mean	40.866841	0.185379	0.127937	0.073107	0.018277	1.950392	
std	15.134494	0.389113	0.334457	0.260653	0.134126	0.630917	
min	15.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	29.000000	0.000000	0.000000	0.000000	0.000000	2.000000	
50%	37.000000	0.000000	0.000000	0.000000	0.000000	2.000000	
75%	51.000000	0.000000	0.000000	0.000000	0.000000	2.000000	
max	82.000000	1.000000	1.000000	1.000000	1.000000	4.000000	

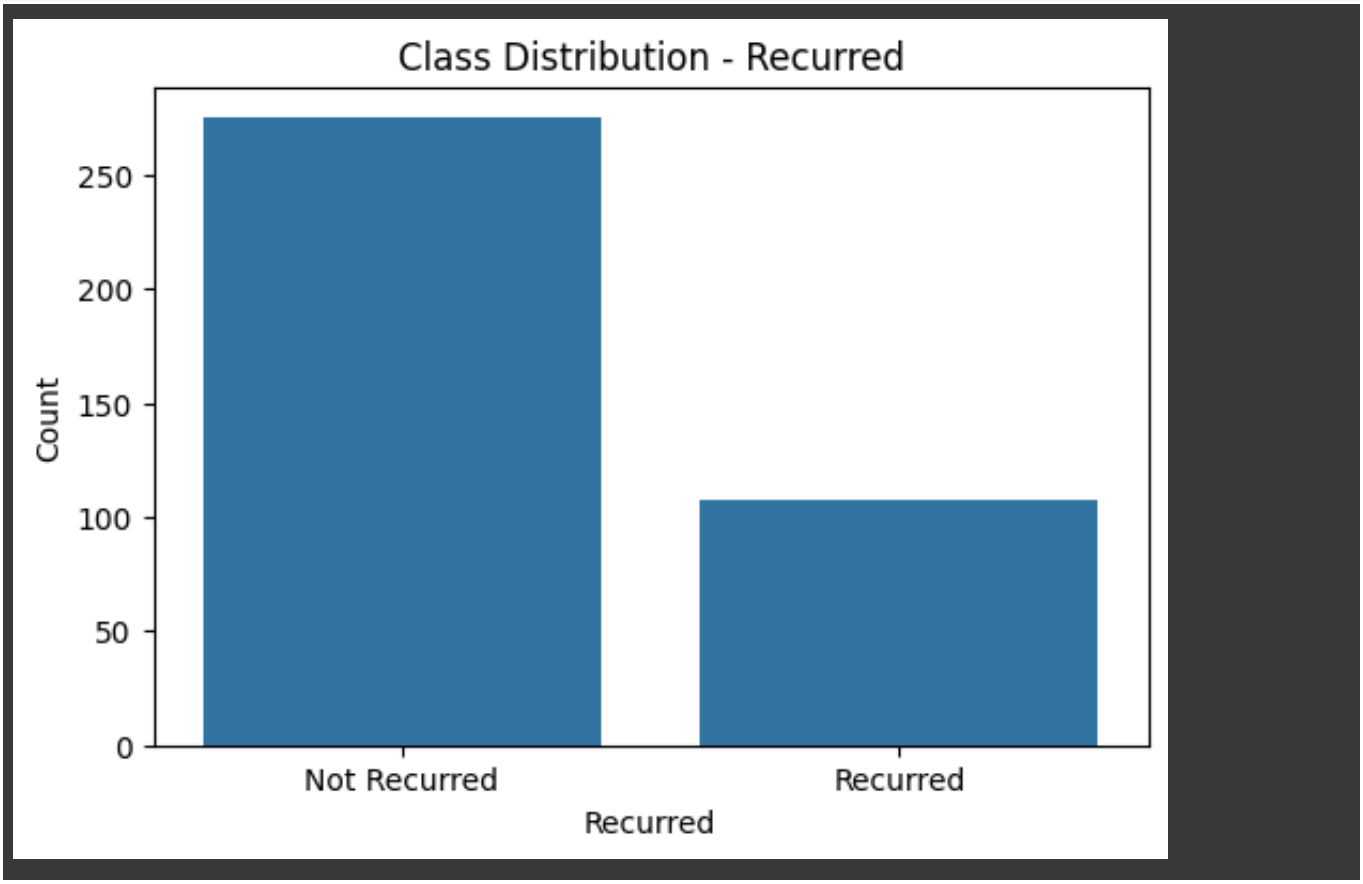
```
sns.countplot(x='Recurred', data=df)
plt.xticks([0, 1], ['Not Recurred', 'Recurred'])
plt.title('Target Class Distribution')
plt.ylabel('Count')
plt.show()
```



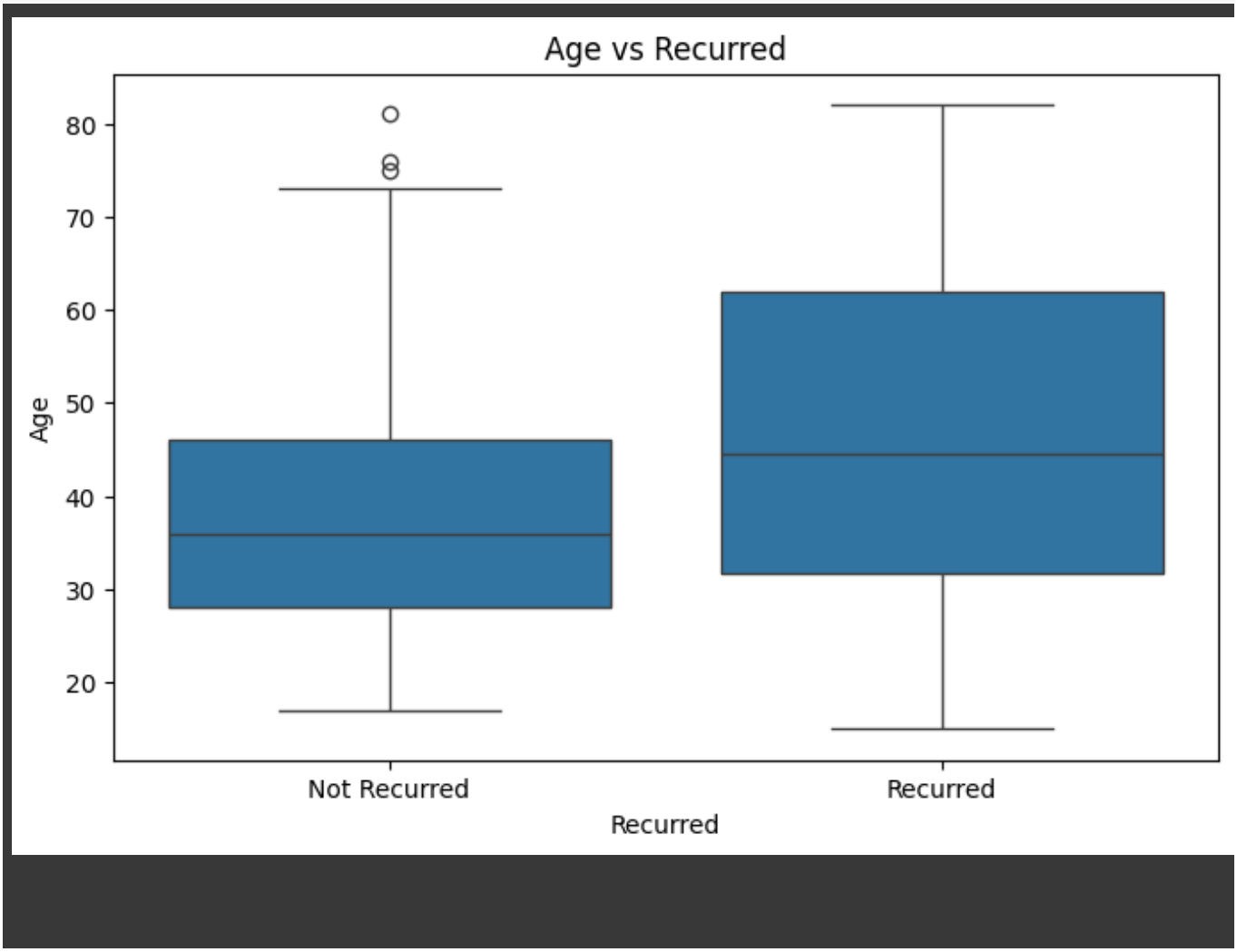
```
plt.figure(figsize=(8, 5))
sns.histplot(df['Age'], bins=30, kde=True)
plt.title('Age Distribution')
plt.xlabel('Age')
plt.ylabel('Count')
plt.show()
```



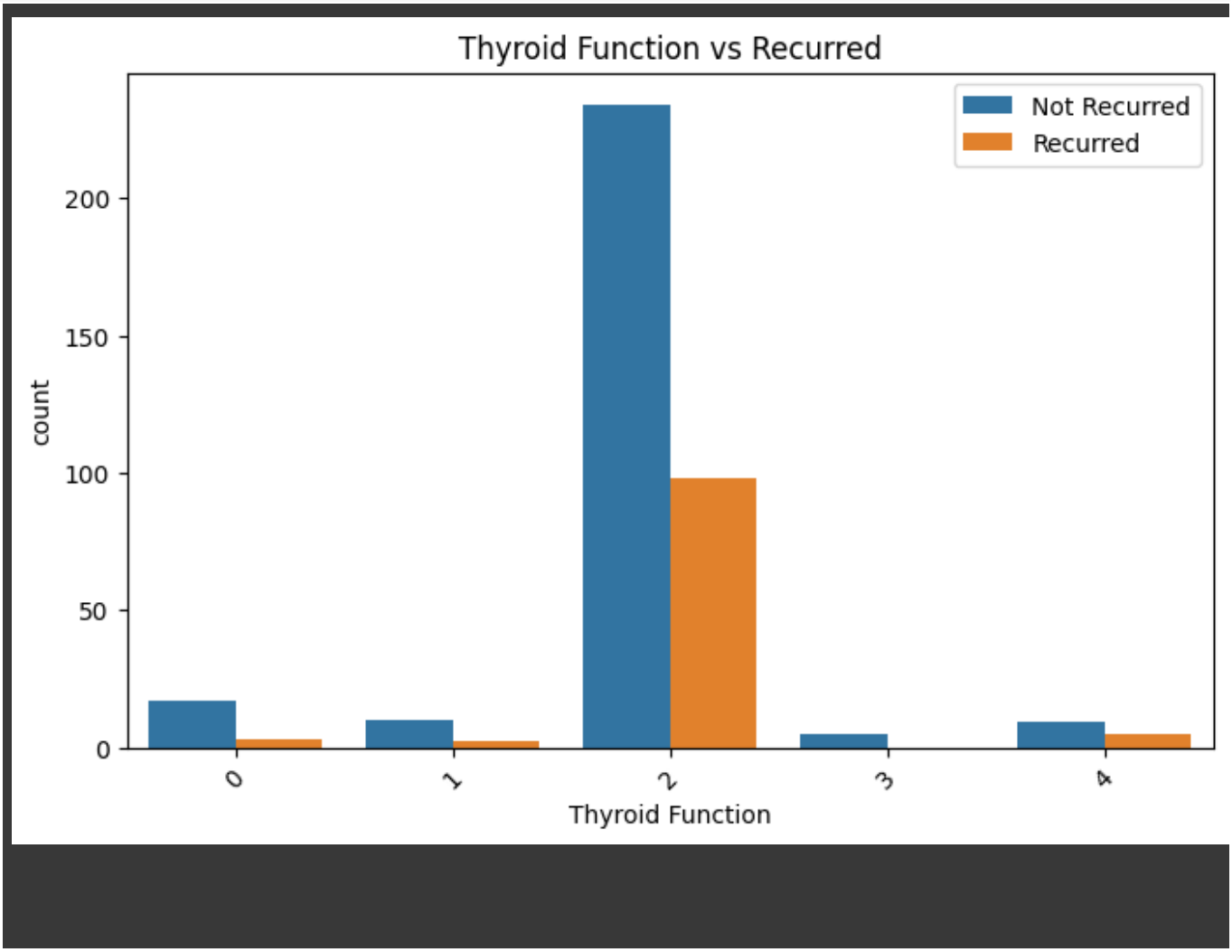
```
plt.figure(figsize=(6, 4))
sns.countplot(x='Recurred', data=df)
plt.title('Class Distribution - Recurred')
plt.xticks([0, 1], ['Not Recurred', 'Recurred'])
plt.ylabel('Count')
plt.show()
```



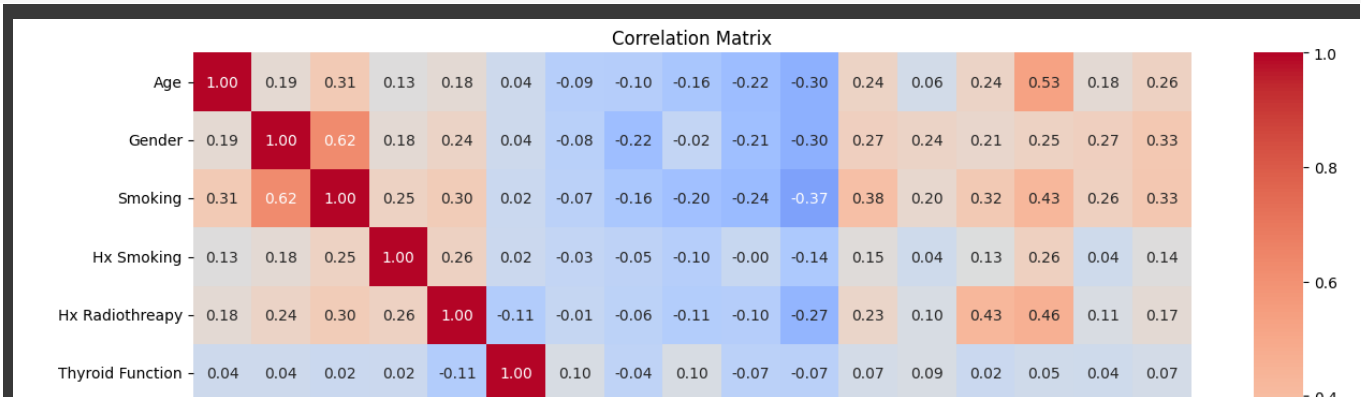
```
plt.figure(figsize=(8, 5))
sns.boxplot(x='Recurred', y='Age', data=df)
plt.title('Age vs Recurred')
plt.xticks([0, 1], ['Not Recurred', 'Recurred'])
plt.show()
```

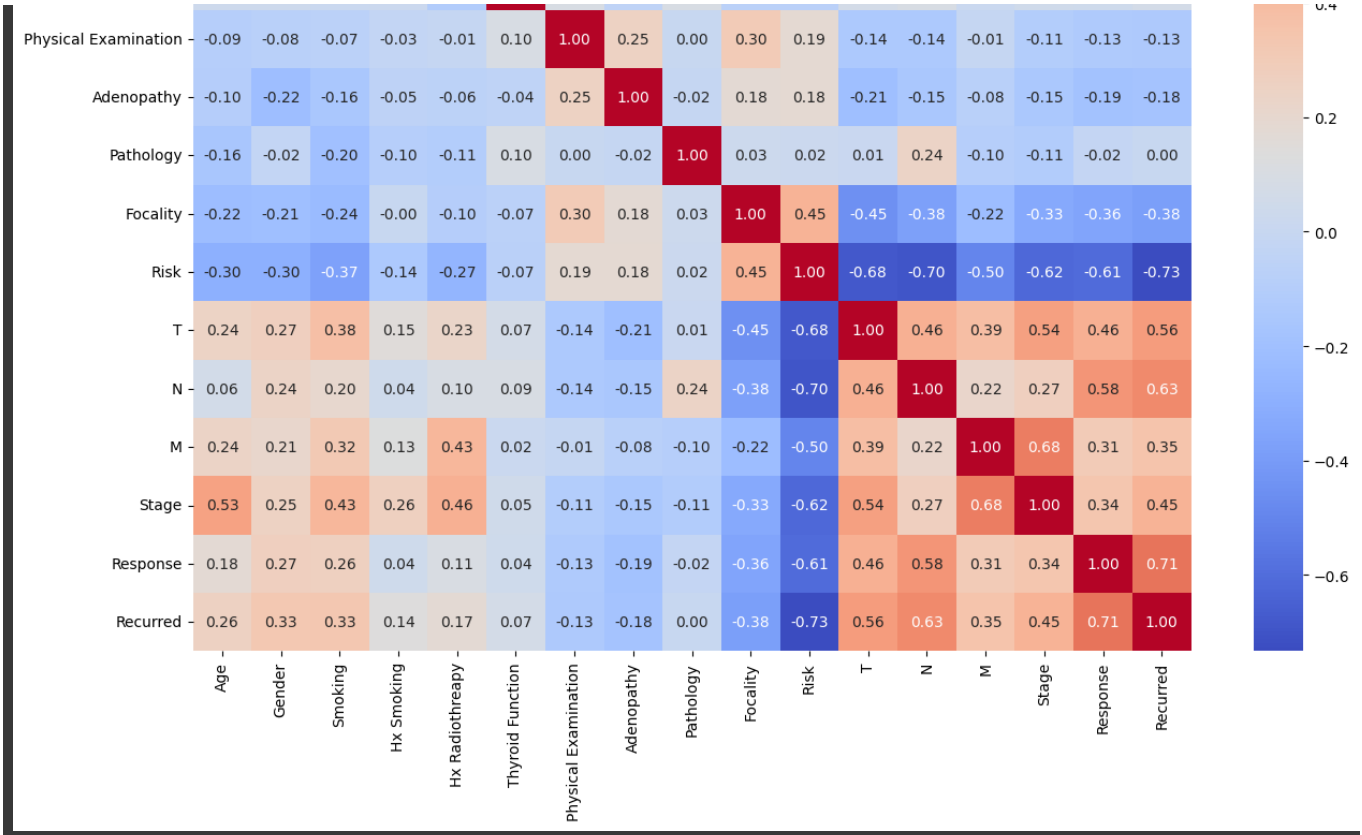


```
plt.figure(figsize=(8, 5))
sns.countplot(x='Thyroid Function', data=df, hue='Recurred')
plt.title('Thyroid Function vs Recurred')
plt.legend(['Not Recurred', 'Recurred'])
plt.xticks(rotation=45)
plt.show()
```




```
plt.figure(figsize=(15, 12))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix')
plt.show()
```






```
X = df.drop('Recurred', axis=1)
y = df['Recurred']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random
```

```
def perform_classification(model):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    print(classification_report(y_test, y_pred))
```

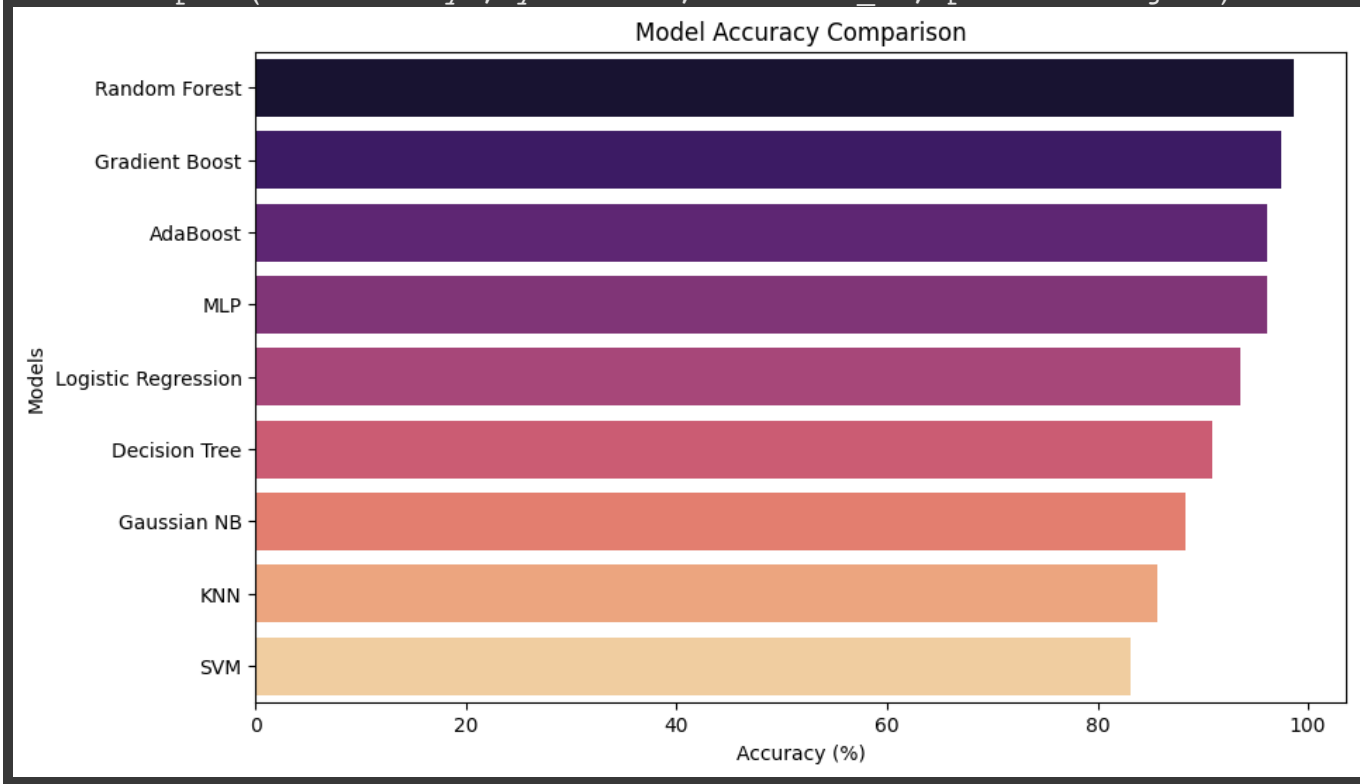
```
def perform_classification(model, name):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    print(f"\n{name} Classification Report:\n")
    print(classification_report(y_test, y_pred))
    return accuracy_score(y_test, y_pred) * 100

random_forest = RandomForestClassifier()
decision_tree = DecisionTreeClassifier()
logistic_regression = LogisticRegression(max_iter=1000)
gradient_boost = GradientBoostingClassifier()
ada_boost = AdaBoostClassifier()
svm = SVC()
knn = KNeighborsClassifier()
mlp = MLPClassifier(max_iter=1000)
gaussian = GaussianNB()

models = {
    'Random Forest': random_forest,
    'Decision Tree': decision_tree,
    'Logistic Regression': logistic_regression,
    'Gradient Boost': gradient_boost,
    'AdaBoost': ada_boost,
    'SVM': svm,
    'KNN': knn,
    'MLP': mlp,
    'Gaussian NB': gaussian
}

results = {}
for name, model in models.items():
    acc = perform_classification(model, name)
    results[name] = acc

sorted_models = sorted(results.items(), key=lambda item: item[1], reverse=True)

print("\n--- Model Accuracy Comparison ---")
for name, acc in sorted_models:
    print(f"{name}: {acc:.2f}%")
```

```
SVM Classification Report:

              precision    recall  f1-score   support

           0       0.82      1.00      0.90        58
           1       1.00      0.32      0.48        19

    accuracy                           0.83        77
   macro avg       0.91      0.66      0.69        77
weighted avg       0.86      0.83      0.80        77


KNN Classification Report:

              precision    recall  f1-score   support

           0       0.85      0.98      0.91        58
           1       0.90      0.47      0.62        19

    accuracy                           0.86        77
   macro avg       0.88      0.73      0.77        77
weighted avg       0.86      0.86      0.84        77


MLP Classification Report:

              precision    recall  f1-score   support

           0       0.95      1.00      0.97        58
           1       1.00      0.84      0.91        19

    accuracy                           0.96        77
   macro avg       0.98      0.92      0.94        77
weighted avg       0.96      0.96      0.96        77


Gaussian NB Classification Report:

              precision    recall  f1-score   support

           0       0.88      0.98      0.93        58
           1       0.92      0.58      0.71        19

    accuracy                           0.88        77
   macro avg       0.90      0.78      0.82        77
weighted avg       0.89      0.88      0.87        77


--- Model Accuracy Comparison ---
Random Forest: 98.70%
Gradient Boost: 97.40%
AdaBoost: 96.10%
MLP: 96.10%
```

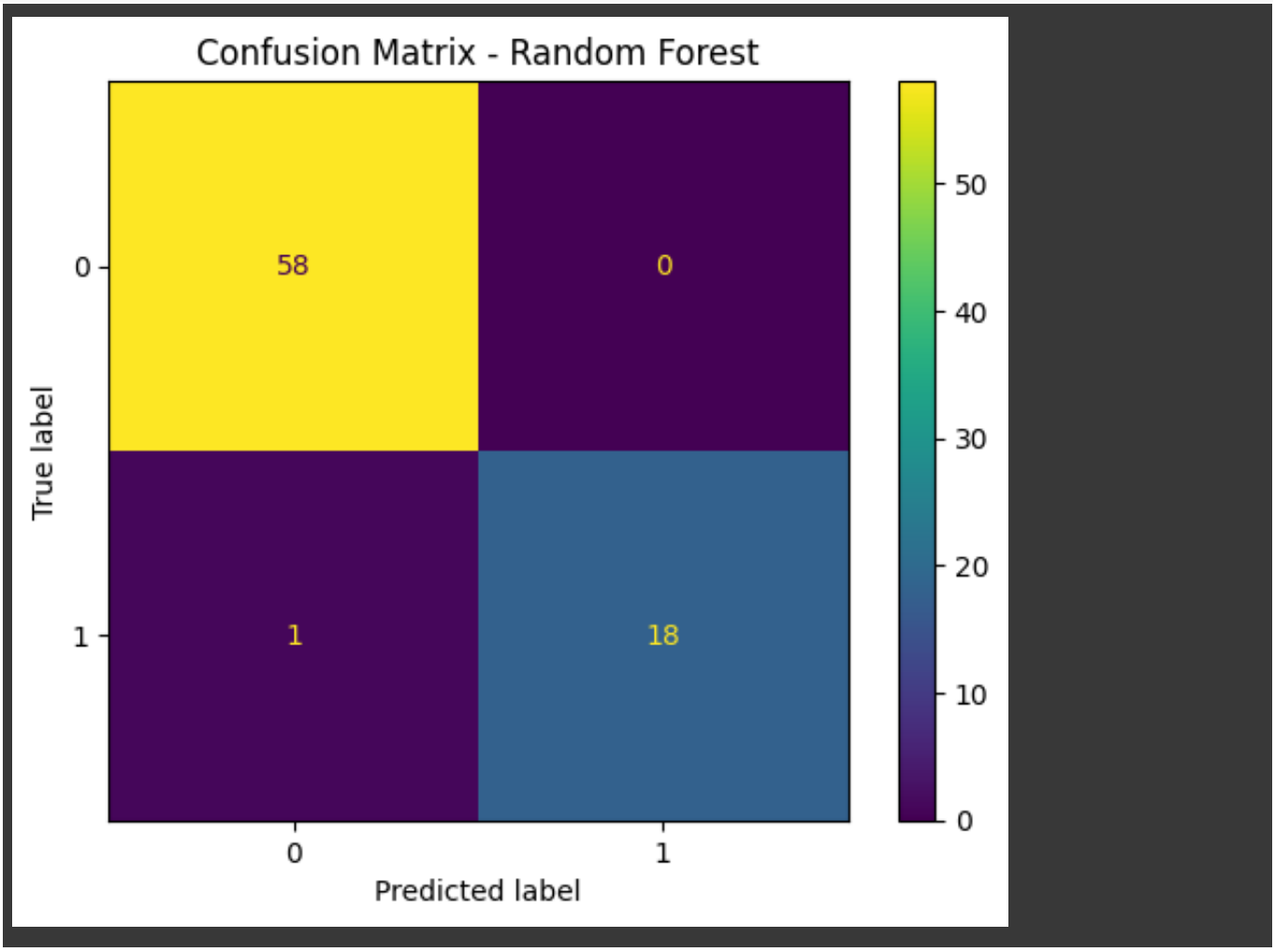
```
Logistic Regression: 93.51%
Decision Tree: 90.91%
Gaussian NB: 88.31%
```

```
acc_df = pd.DataFrame(list(results.items()), columns=['Model', 'Accuracy'])
acc_df.sort_values(by='Accuracy', ascending=False, inplace=True)

plt.figure(figsize=(10, 6))
sns.barplot(x='Accuracy', y='Model', data=acc_df, palette='magma')
plt.title('Model Accuracy Comparison')
plt.xlabel('Accuracy (%)')
plt.ylabel('Models')
plt.show()
```

```
/tmp/ipython-input-4075718528.py:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed

  sns.barplot(x='Accuracy', y='Model', data=acc_df, palette='magma')
```



```
from sklearn.metrics import ConfusionMatrixDisplay

best_model = RandomForestClassifier()
best_model.fit(X_train, y_train)
y_pred = best_model.predict(X_test)

ConfusionMatrixDisplay.from_predictions(y_test, y_pred)
plt.title('Confusion Matrix - Random Forest')
plt.show()
```



```
importances = best_model.feature_importances_
features = X.columns
indices = np.argsort(importances)

plt.figure(figsize=(10, 6))
plt.title('Feature Importances - Random Forest')
plt.barh(range(len(indices)), importances[indices], color='teal')
plt.yticks(range(len(indices)), [features[i] for i in indices])
plt.xlabel('Relative Importance')
plt.tight_layout()
plt.show()
```