In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_breast_cancer
```

In [4]:
```
# Step 1: Load and Preprocess Data
data = load_breast_cancer()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target
```

In [8]:
```
df
```

Out[8]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	sy
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	
...	...	...	...	...	...	...	...	...	
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	

569 rows × 31 columns

In [6]:
```
# Checking for missing values
print("Missing values:")
print(df.isnull().sum())  # No missing values
```

```
Missing values:
mean radius                0
mean texture               0
mean perimeter             0
mean area                  0
mean smoothness            0
mean compactness           0
mean concavity             0
mean concave points        0
mean symmetry              0
mean fractal dimension     0
radius error               0
texture error              0
perimeter error            0
area error                 0
smoothness error           0
compactness error          0
concavity error            0
concave points error       0
symmetry error             0
fractal dimension error    0
worst radius               0
worst texture              0
worst perimeter            0
worst area                 0
worst smoothness           0
worst compactness          0
worst concavity            0
worst concave points       0
worst symmetry             0
worst fractal dimension    0
target                     0
dtype: int64
```

In [10]:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

In [12]:

```
# Splitting the dataset into train and test
X = df.drop(columns=['target'])
y = df['target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

In [14]:

```
# Feature scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [16]:

```
# Step 2: Classification Algorithm Implementation
models = {
```

```
    "Logistic Regression": LogisticRegression(),
    "Decision Tree": DecisionTreeClassifier(),
    "Random Forest": RandomForestClassifier(),
    "Support Vector Machine": SVC(),
    "k-Nearest Neighbors": KNeighborsClassifier()
}
results = {}
```

In [18]:

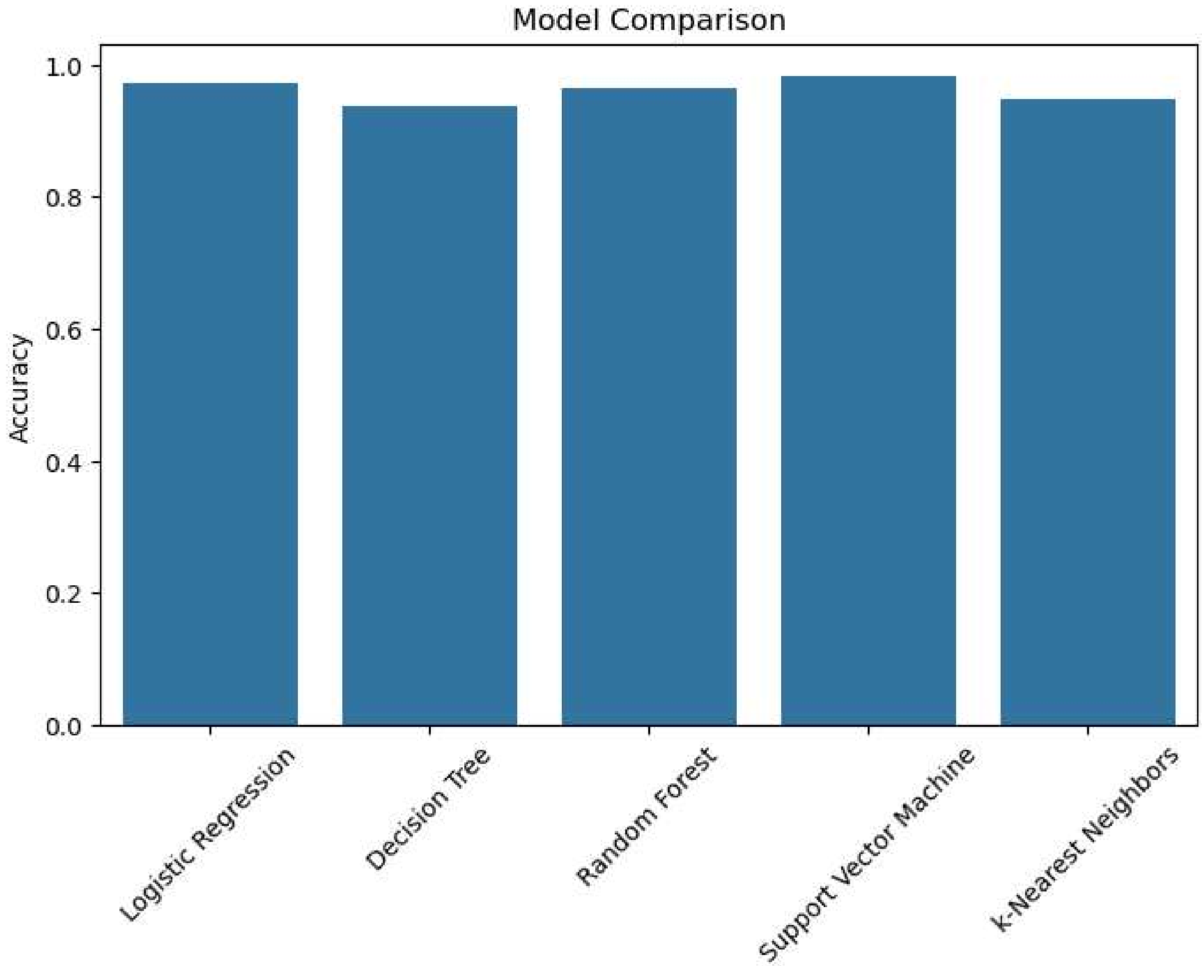
```
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    results[name] = acc
    print(f"{name} Accuracy: {acc:.4f}")
    print(classification_report(y_test, y_pred))
    print("Confusion Matrix:")
    print(confusion_matrix(y_test, y_pred))
    print("-" * 50)
```

```
Logistic Regression Accuracy: 0.9737
              precision    recall  f1-score   support

           0       0.98      0.95      0.96        43
           1       0.97      0.99      0.98        71

    accuracy                           0.97       114
   macro avg       0.97      0.97      0.97       114
weighted avg       0.97      0.97      0.97       114

Confusion Matrix:
[[41  2]
 [ 1 70]]
--------------------------------------------------
Decision Tree Accuracy: 0.9386
              precision    recall  f1-score   support

           0       0.91      0.93      0.92        43
           1       0.96      0.94      0.95        71

    accuracy                           0.94       114
   macro avg       0.93      0.94      0.93       114
weighted avg       0.94      0.94      0.94       114

Confusion Matrix:
[[40  3]
 [ 4 67]]
--------------------------------------------------
Random Forest Accuracy: 0.9649
              precision    recall  f1-score   support

           0       0.98      0.93      0.95        43
           1       0.96      0.99      0.97        71

    accuracy                           0.96       114
   macro avg       0.97      0.96      0.96       114
weighted avg       0.97      0.96      0.96       114

Confusion Matrix:
[[40  3]
 [ 1 70]]
--------------------------------------------------
Support Vector Machine Accuracy: 0.9825
              precision    recall  f1-score   support

           0       1.00      0.95      0.98        43
           1       0.97      1.00      0.99        71

    accuracy                           0.98       114
   macro avg       0.99      0.98      0.98       114
weighted avg       0.98      0.98      0.98       114

Confusion Matrix:
[[41  2]
 [ 0 71]]
--------------------------------------------------
```

```
k-Nearest Neighbors Accuracy: 0.9474
              precision    recall  f1-score   support

           0       0.93      0.93      0.93        43
           1       0.96      0.96      0.96        71

    accuracy                           0.95       114
   macro avg       0.94      0.94      0.94       114
weighted avg       0.95      0.95      0.95       114

Confusion Matrix:
[[40  3]
 [ 3 68]]
--------------------------------------------------
```

In [20]:

```
# Step 3: Model Comparison
plt.figure(figsize=(8, 5))
sns.barplot(x=list(results.keys()), y=list(results.values()))
plt.xticks(rotation=45)
plt.ylabel("Accuracy")
plt.title("Model Comparison")
plt.show()

best_model = max(results, key=results.get)
worst_model = min(results, key=results.get)
print(f"Best Performing Model: {best_model}")
print(f"Worst Performing Model: {worst_model}")
```




```
Best Performing Model: Support Vector Machine
Worst Performing Model: Decision Tree
```