In [1]:
```python
from sklearn.datasets import load_breast_cancer
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```python
df = load_breast_cancer(as_frame=True)
```

In [3]:
```python
X = df.data
y = df.target
```

In [4]:
```python
X
```

Out[4]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concav poin
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.147
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.070
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.127
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.105
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.104
...	...	...	...	...	...	...	...	
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.138
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.097
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.053
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.152
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.000

569 rows × 30 columns

In [5]:
```python
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
import numpy as np
from sklearn.model_selection import GridSearchCV
```

In [6]:
```python
knn_pipe = Pipeline([
    ("scaler", StandardScaler()),
    ("knn", KNeighborsClassifier())
])
```

In [7]:
```
knn_grid = {
    "knn__n_neighbors": np.arange(1,31,2)
}
```

In [8]:
```
knn = GridSearchCV(
    knn_pipe,
    knn_grid,
    cv = 5,
    n_jobs = -1
)
```

In [9]:
```
lr_pipe = Pipeline([
    ("scaler", StandardScaler()),
    ("lr", LogisticRegression(max_iter=200_000))
])
```

In [10]:
```
lr_grid = {
    "lr__C":np.logspace(-3, 3, 50)
}
```

In [11]:
```
lr = GridSearchCV(
    lr_pipe,
    lr_grid,
    cv = 5,
    n_jobs = -1
)
```

In [12]:
```
dt_pipe = Pipeline([
    ("scaler", StandardScaler()),
    ("dt", DecisionTreeClassifier())
])
```

In [13]:
```
DecisionTreeClassifier()
```

Out[13]:

▾ DecisionTreeClassifier

▶ Parameters

In [14]:
```
dt_grid = {
    "dt__criterion": ["gini", "entropy"],
    "dt__max_depth": [None, 3, 5, 8, 12]
}
```

In [15]:
```
dt = GridSearchCV(
    dt_pipe,
    dt_grid,
    cv = 5,
    n_jobs = -1
)
```

In [16]:
```python
nb_pipe = Pipeline([
    ("scaler", StandardScaler()),
    ("nb", GaussianNB())
])
```

In [17]:
```python
nb_grid = {
    "nb__var_smoothing": np.logspace(-10, -8, 20)
}
```

In [18]:
```python
nb = GridSearchCV(
    nb_pipe,
    nb_grid,
    cv = 5,
    n_jobs = -1
)
```

In [19]:
```python
rf_pipe = Pipeline([
    ("scaler", StandardScaler()),
    ("rf", RandomForestClassifier())
])
```

In [20]:
```python
RandomForestClassifier()
```

Out[20]:

```
▾ RandomForestClassifier
▶ Parameters
```

In [21]:
```python
rf_grid = {
    "rf__n_estimators": [50, 100, 200],
    "rf__max_depth": [5, 10, 15, None],
    "rf__min_samples_split": [2, 5, 10]
}
```

In [22]:
```python
rf = GridSearchCV(
    rf_pipe,
    rf_grid,
    cv = 5,
    n_jobs = -1
)
```

In [23]:
```python
from sklearn.model_selection import StratifiedKFold
import pandas as pd
from sklearn.metrics import (accuracy_score, precision_score, recall_score,
    f1_score, confusion_matrix, roc_curve, auc
)
import matplotlib.pyplot as plt
```

In [24]:
```python
def cv5_report(name, model, X, y, outer_splits=5):

    skf = StratifiedKFold(n_splits = outer_splits, random_state = 42, shuffl

    folds = []
```

```python
aucs = []

plt.figure(figsize=(6,5))
for fold, (train_index, test_index) in enumerate(skf.split(X,y),1):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    model.fit(X_train, y_train)

    if hasattr(model, "best_estimator_"):
        best_model = model.best_estimator_
    else:
        best_model = model

    y_test_pred = best_model.predict(X_test)
    y_test_prob = best_model.predict_proba(X_test)[:,1]

    #Test Metrics
    acc = accuracy_score(y_test, y_test_pred)
    pre = precision_score(y_test, y_test_pred)
    rec = recall_score(y_test, y_test_pred)
    f1 = f1_score(y_test, y_test_pred)

    #ROC_curve
    fpr, tpr, _ = roc_curve(y_test, y_test_prob)
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, label=f"AUC = {roc_auc:.2f}")
    plt.xlabel("False Positive Rate (FPR)")
    plt.ylabel("True Positive Rate (TPR)")
    plt.title(f"{name} ROC Curve")
    plt.grid()
    plt.legend()


    aucs.append(roc_auc)

    folds.append(
        {"Model":name, "Fold":fold, "Accuracy":acc, "Precision":pre,
        "Recall":rec, "F1":f1, "AUC": roc_auc}
    )

plt.show()

folds_df = pd.DataFrame(folds)

row_mean = {
        "Model":name,
        "Fold":"Mean",
        "Accuracy":folds_df["Accuracy"].mean(),
        "Precision":folds_df["Precision"].mean(),
        "Recall":folds_df["Recall"].mean(),
        "F1":folds_df["F1"].mean(),
        "AUC":folds_df["AUC"].mean(),
}

results = pd.concat([folds_df, pd.DataFrame([row_mean])])
```

```python
    return results
```

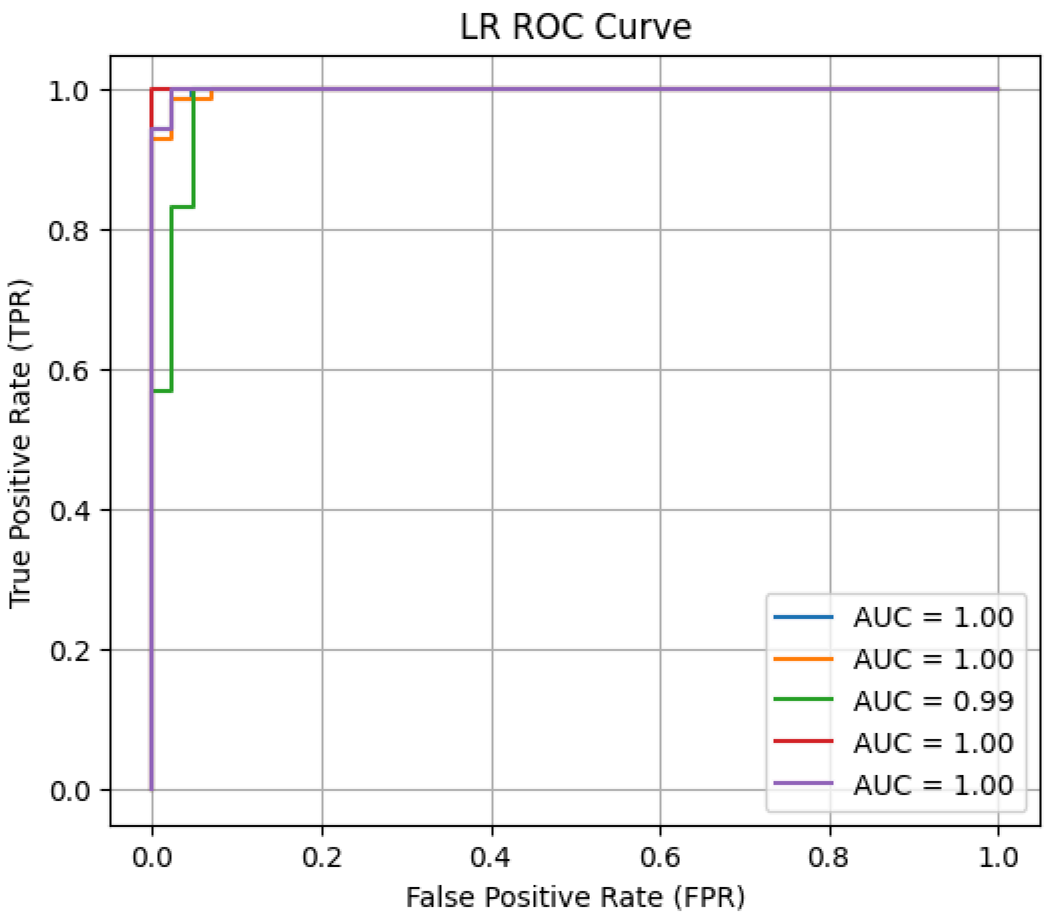
In [25]:
```python
from sklearn.svm import SVC
```

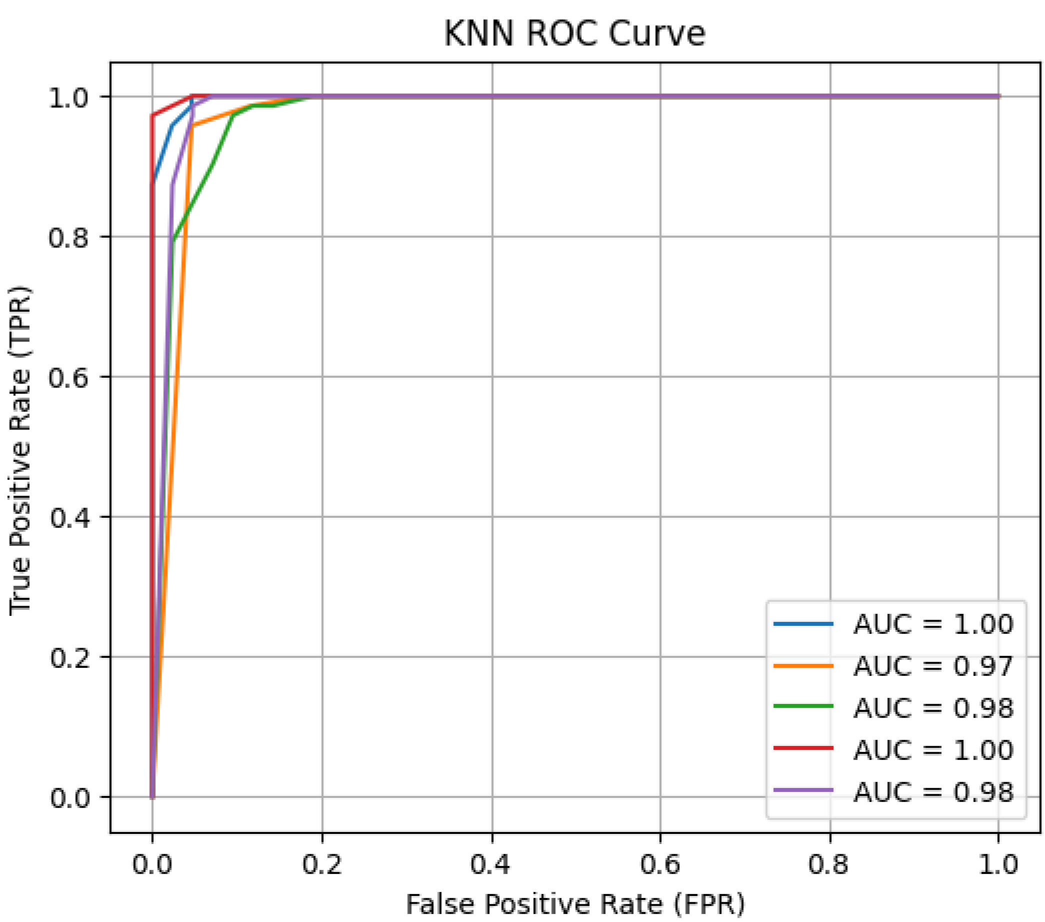
In [26]:
```python
svm_pipe = Pipeline([
    ("scaler", StandardScaler()),
    ("svm", SVC(probability = True))
])
```

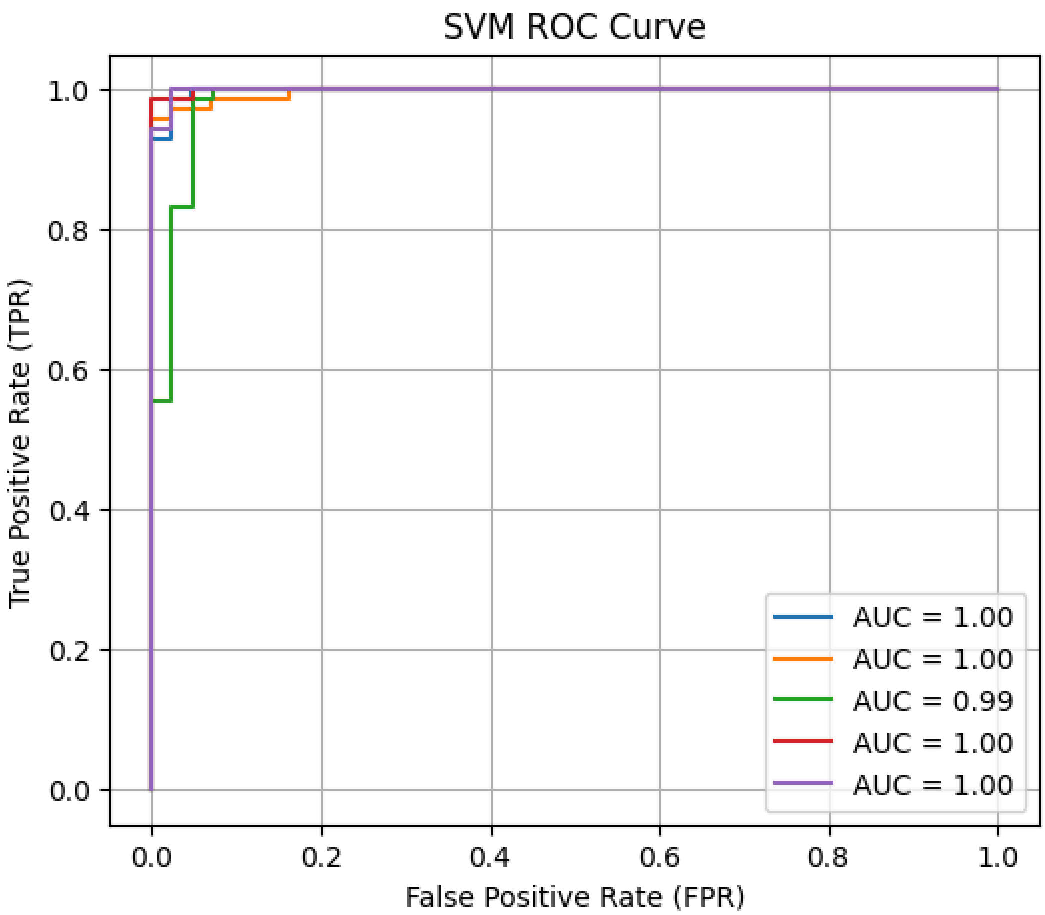
In [27]:
```python
svm_grid = {
    "svm__C": np.logspace(-2,2,50),
    "svm__kernel" : ["linear"]
}
```

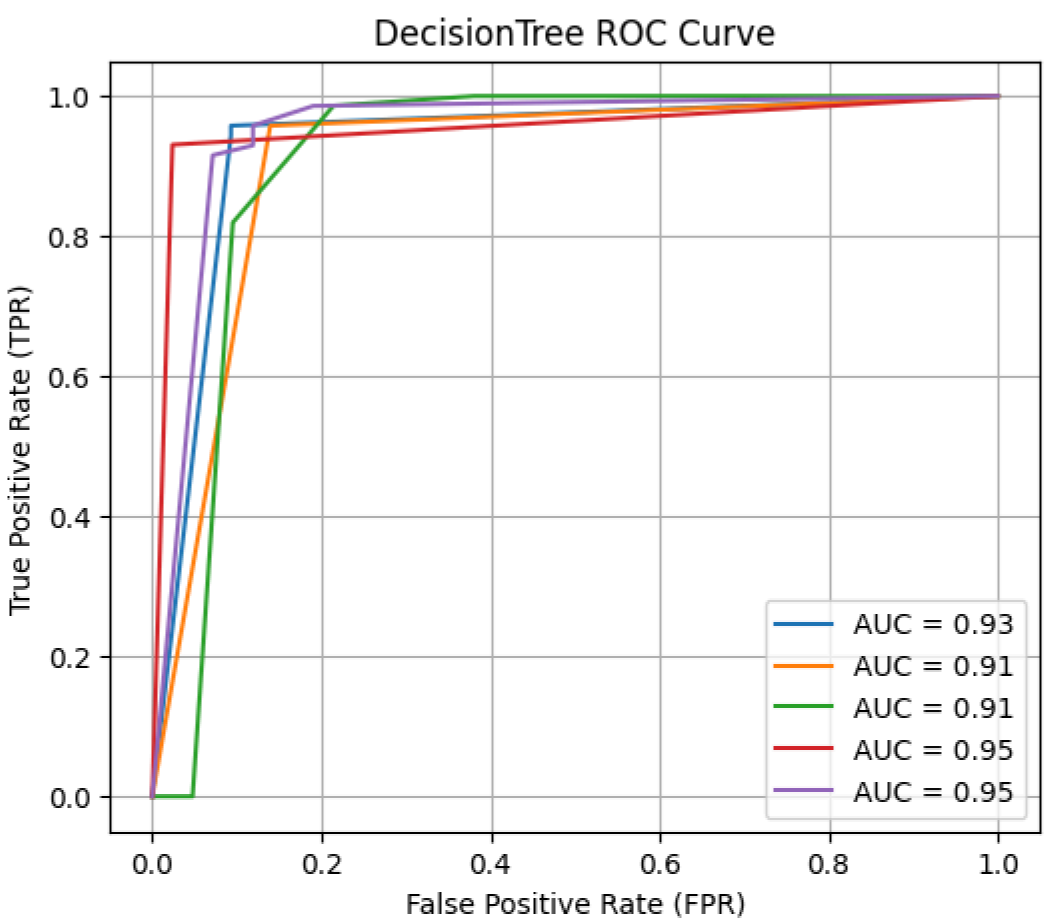
In [28]:
```python
svm = GridSearchCV(
    svm_pipe,
    svm_grid,
    cv = 5,
    n_jobs = -1
)
```

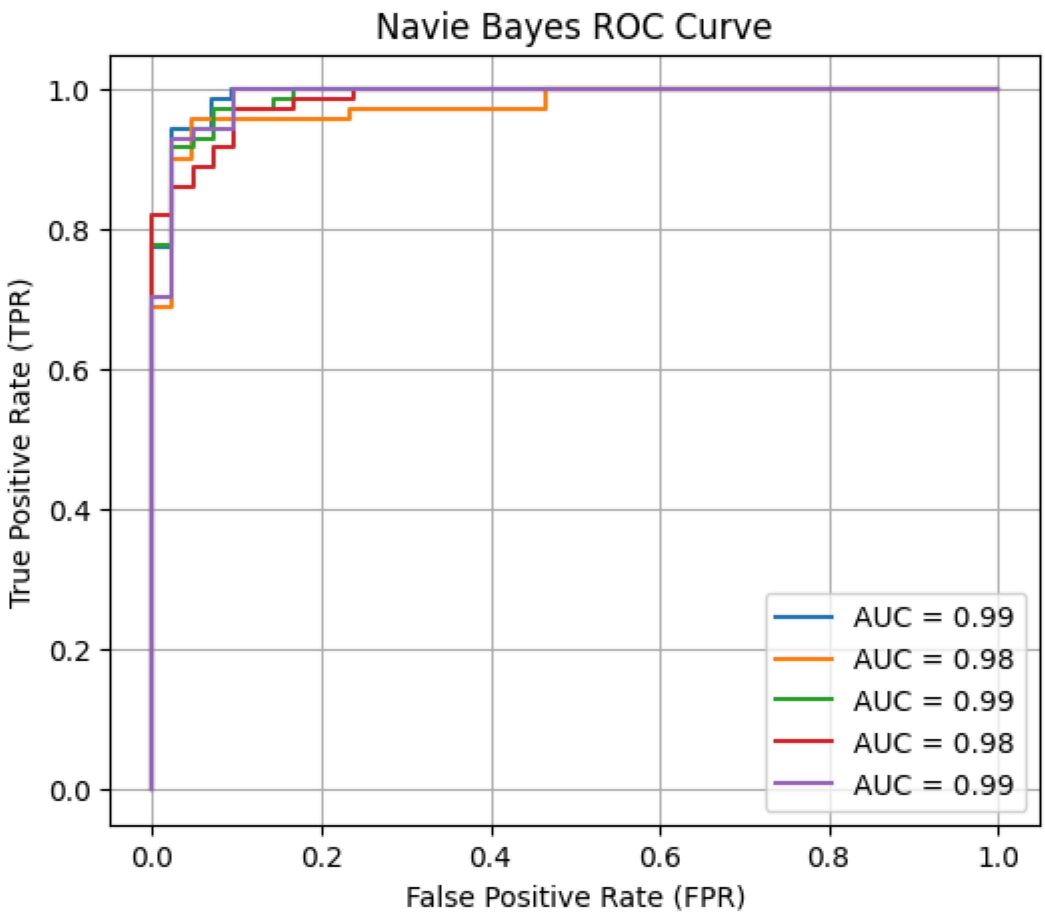
In [29]:
```python
pd.concat([
    cv5_report("LR", lr, X, y, 5),
    cv5_report("KNN", knn, X, y, 5),
    cv5_report("SVM", svm, X, y, 5),
    cv5_report("DecisionTree", dt, X, y, 5),
    cv5_report("Navie Bayes", nb, X, y, 5),
    cv5_report("Random Forest", rf, X, y, 5)
]).set_index(["Model"])
```

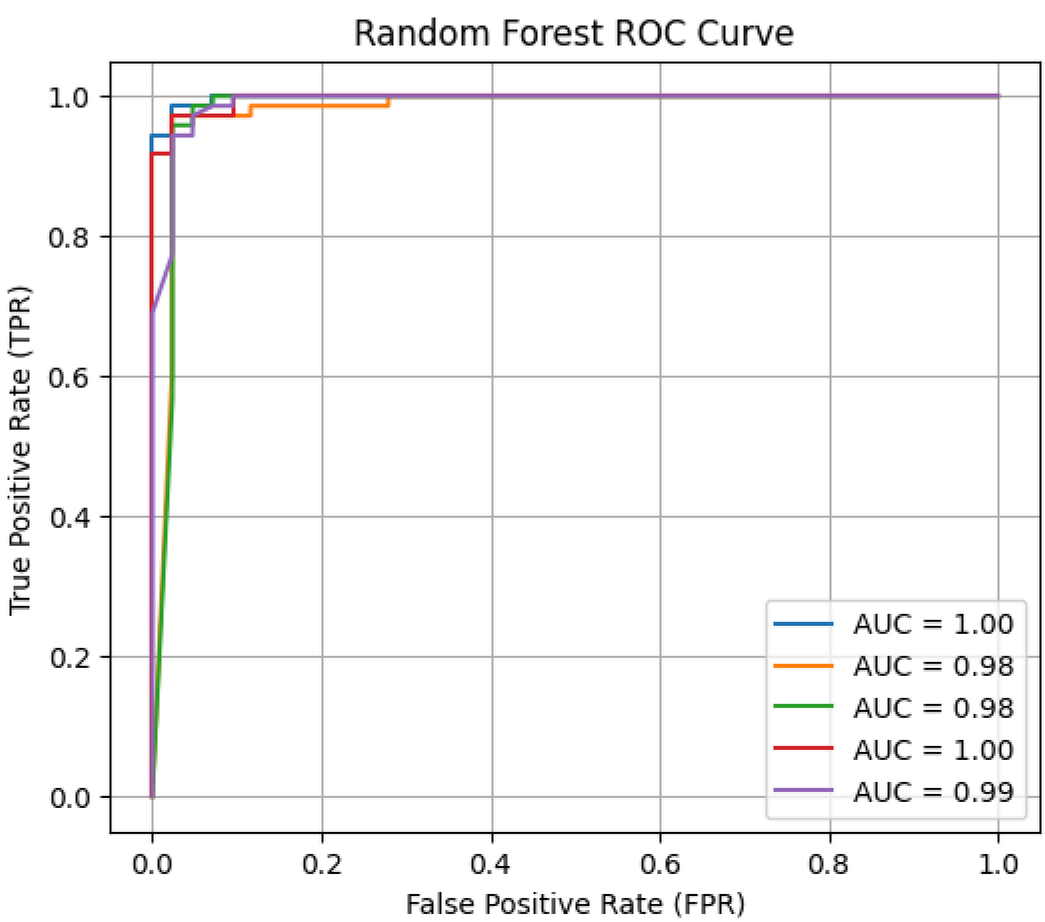

LR ROC Curve
True Positive Rate (TPR)
False Positive Rate (FPR)
0.0
0.2
0.4
0.6
0.8
1.0
AUC = 1.00
AUC = 1.00
AUC = 0.99
AUC = 1.00
AUC = 1.00


KNN ROC Curve
True Positive Rate (TPR)
False Positive Rate (FPR)
0.0
0.2
0.4
0.6
0.8
1.0
AUC = 1.00
AUC = 0.97
AUC = 0.98
AUC = 1.00
AUC = 0.98


SVM ROC Curve
True Positive Rate (TPR)
False Positive Rate (FPR)
0.0
0.2
0.4
0.6
0.8
1.0
AUC = 1.00
AUC = 1.00
AUC = 0.99
AUC = 1.00
AUC = 1.00


DecisionTree ROC Curve
True Positive Rate (TPR)
False Positive Rate (FPR)
0.0
0.2
0.4
0.6
0.8
1.0
AUC = 0.93
AUC = 0.91
AUC = 0.91
AUC = 0.95
AUC = 0.95


Navie Bayes ROC Curve
True Positive Rate (TPR)
False Positive Rate (FPR)
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
AUC = 0.99
AUC = 0.98
AUC = 0.99
AUC = 0.98
AUC = 0.99


Random Forest ROC Curve
True Positive Rate (TPR)
False Positive Rate (FPR)
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
AUC = 1.00
AUC = 0.98
AUC = 0.98
AUC = 1.00
AUC = 0.99

Out[29]:

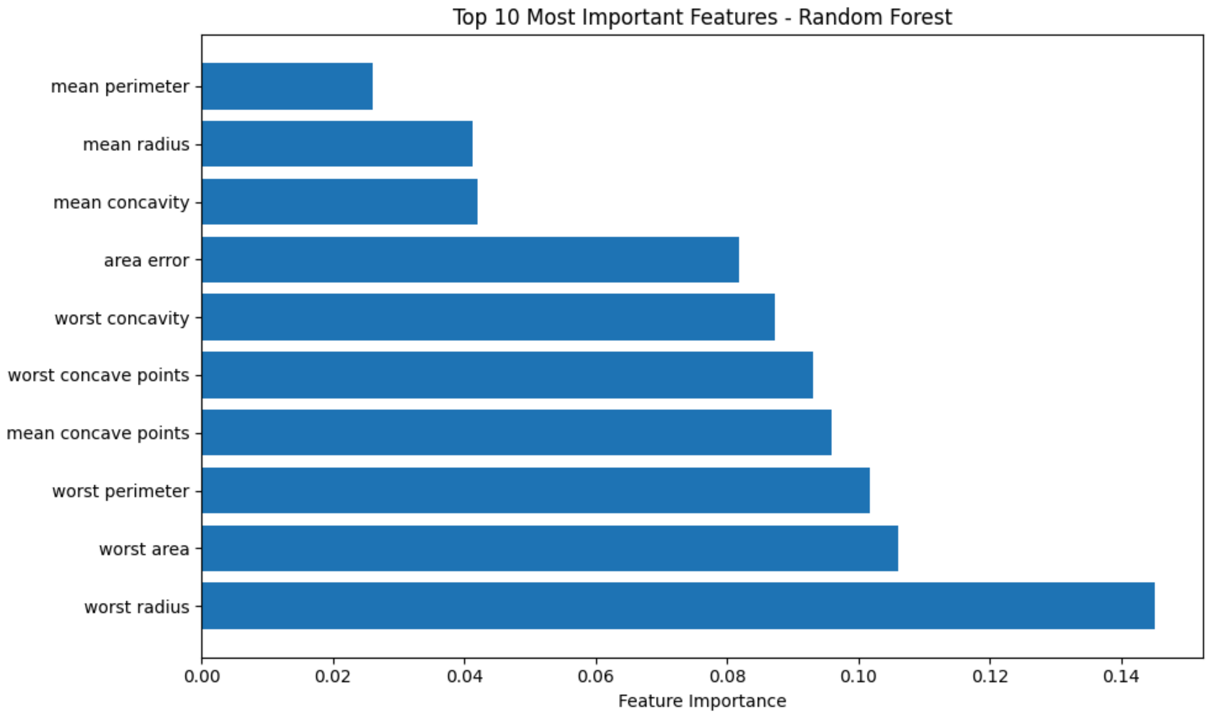
Model	Fold	Accuracy	Precision	Recall	F1	AUC
LR	1	0.973684	0.972222	0.985915	0.979021	0.998362
LR	2	0.938596	0.910256	1.000000	0.953020	0.997707
LR	3	0.964912	0.947368	1.000000	0.972973	0.985780
LR	4	0.991228	1.000000	0.986111	0.993007	1.000000
LR	5	0.991150	0.986111	1.000000	0.993007	0.998659
LR	Mean	0.971914	0.963192	0.994405	0.978206	0.996102
KNN	1	0.982456	0.972603	1.000000	0.986111	0.997380
KNN	2	0.947368	0.933333	0.985915	0.958904	0.973305
KNN	3	0.947368	0.934211	0.986111	0.959459	0.975694
KNN	4	0.982456	1.000000	0.972222	0.985915	0.999339
KNN	5	0.973451	0.959459	1.000000	0.979310	0.984574
KNN	Mean	0.966620	0.959921	0.988850	0.973940	0.986058
SVM	1	0.982456	0.985915	0.985915	0.985915	0.998035
SVM	2	0.956140	0.945946	0.985915	0.965517	0.996397
SVM	3	0.964912	0.947368	1.000000	0.972973	0.985119
SVM	4	0.991228	1.000000	0.986111	0.993007	0.999339
SVM	5	0.991150	0.986111	1.000000	0.993007	0.998659
SVM	Mean	0.977177	0.973068	0.991588	0.982084	0.995510
DecisionTree	1	0.938596	0.944444	0.957746	0.951049	0.931707
DecisionTree	2	0.921053	0.918919	0.957746	0.937931	0.909106
DecisionTree	3	0.912281	0.887500	0.986111	0.934211	0.911541
DecisionTree	4	0.947368	0.985294	0.930556	0.957143	0.953373
DecisionTree	5	0.920354	0.897436	0.985915	0.939597	0.949866
DecisionTree	Mean	0.927930	0.926719	0.963615	0.943986	0.931118
Navie Bayes	1	0.956140	0.958333	0.971831	0.965035	0.991811
Navie Bayes	2	0.912281	0.906667	0.957746	0.931507	0.976089
Navie Bayes	3	0.929825	0.910256	0.986111	0.946667	0.988757
Navie Bayes	4	0.903509	0.969231	0.875000	0.919708	0.984788
Navie Bayes	5	0.946903	0.945205	0.971831	0.958333	0.988598
Navie Bayes	Mean	0.929731	0.937939	0.952504	0.944250	0.986009
Random Forest	1	0.964912	0.985507	0.957746	0.971429	0.998035

Model	Fold	Accuracy	Precision	Recall	F1	AUC
Random Forest	2	0.921053	0.897436	0.985915	0.939597	0.978873
Random Forest	3	0.973684	0.960000	1.000000	0.979592	0.981647
Random Forest	4	0.947368	0.985294	0.930556	0.957143	0.996032
Random Forest	5	0.964602	0.946667	1.000000	0.972603	0.991449
Random Forest	Mean	0.954324	0.954981	0.974844	0.964073	0.989207

In [30]:
```
knn.best_params_
```
Out[30]:
```
{'knn__n_neighbors': np.int64(7)}
```
In [31]:
```
lr.best_params_
```
Out[31]:
```
{'lr__C': np.float64(0.655128556859551)}
```
In [32]:
```
svm.best_params_
```
Out[32]:
```
{'svm__C': np.float64(1.325711365590108), 'svm__kernel': 'linear'}
```
In [33]:
```
nb.best_params_
```
Out[33]:
```
{'nb__var_smoothing': np.float64(1e-10)}
```
In [34]:
```
rf.best_params_
```
Out[34]:
```
{'rf__max_depth': 15, 'rf__min_samples_split': 2, 'rf__n_estimators': 50}
```
In [35]:
```
# Get feature importances from the best model

best_rf = rf.best_estimator_
feature_importance_rf = best_rf.named_steps["rf"].feature_importances_

# Create a DataFrame to visualize feature importance

importance_df_rf = pd.DataFrame({
        'Feature': X.columns,
        'Importance': feature_importance_rf
}).sort_values('Importance', ascending=False)
print("\nTop 10 Most Important Features - Random Forest:")
print(importance_df_rf.head(10))


# Plot
plt.figure(figsize=(10, 6))
plt.barh(importance_df_rf['Feature'].head(10), importance_df_rf['Importance'
plt.xlabel('Feature Importance')
plt.title('Top 10 Most Important Features - Random Forest')
```

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

```
Top 10 Most Important Features - Random Forest:
                 Feature  Importance
20          worst radius    0.145214
23            worst area    0.106068
22       worst perimeter    0.101685
7    mean concave points    0.095882
27  worst concave points    0.093164
26       worst concavity    0.087323
13            area error    0.081751
6         mean concavity    0.042094
0            mean radius    0.041257
2         mean perimeter    0.026013
```



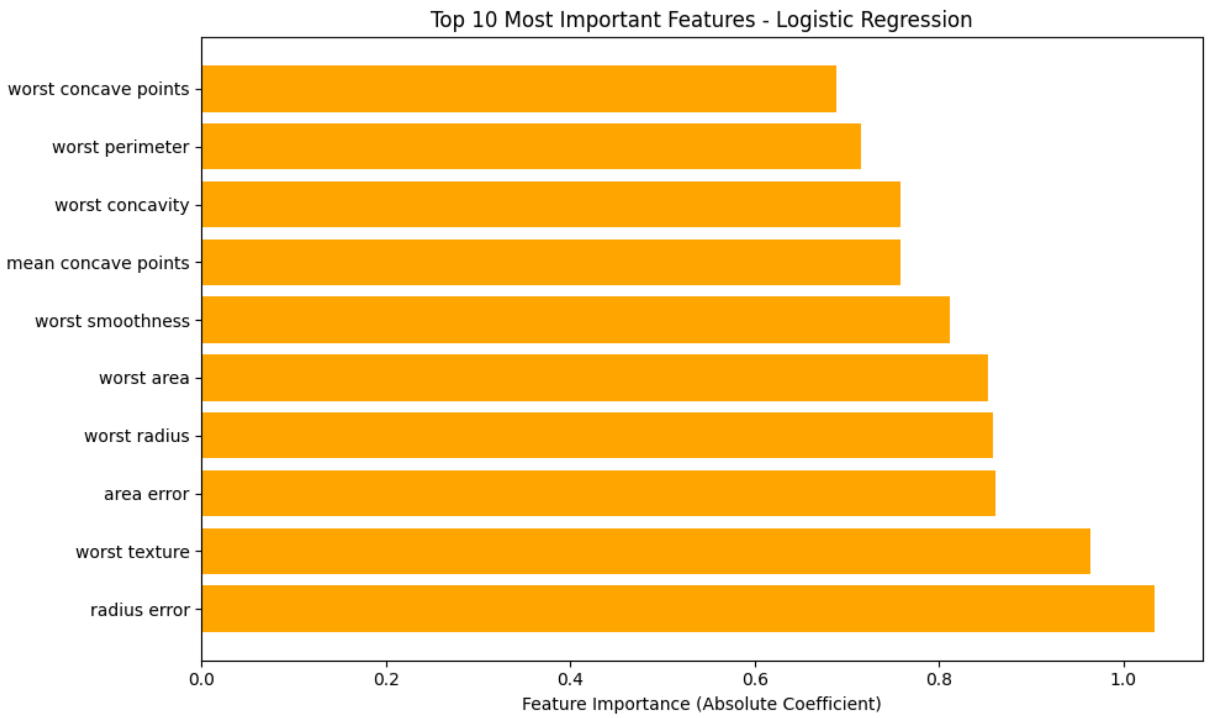
In [ ]:

In [36]:

```
# Get feature importances from the best model
lr_best = lr.best_estimator_
lr_clf = lr_best.named_steps["lr"]

# Coefficients -> importance (absolute value)
coef = lr_clf.coef_          # shape: (1, n_features) for binary classific
imp = np.abs(coef).flatten()

# Create DataFrame
lr_importance_df = pd.DataFrame({
    'Feature': X.columns,
    'Importance': imp
}).sort_values('Importance', ascending=False)

print("\nTop 10 Most Important Features - Logistic Regression:")
print(lr_importance_df.head(10))
```

```python
# Plot
plt.figure(figsize=(10, 6))
plt.barh(lr_importance_df['Feature'].head(10), lr_importance_df['Importance'
plt.xlabel('Feature Importance (Absolute Coefficient)')
plt.title('Top 10 Most Important Features - Logistic Regression')
plt.tight_layout()
plt.show()
```

```
Top 10 Most Important Features - Logistic Regression:
                 Feature  Importance
10          radius error    1.034049
21         worst texture    0.963975
13            area error    0.861386
20          worst radius    0.858908
23            worst area    0.852374
24      worst smoothness    0.812187
7     mean concave points    0.758627
26       worst concavity    0.758144
22       worst perimeter    0.715426
27  worst concave points    0.688695
```



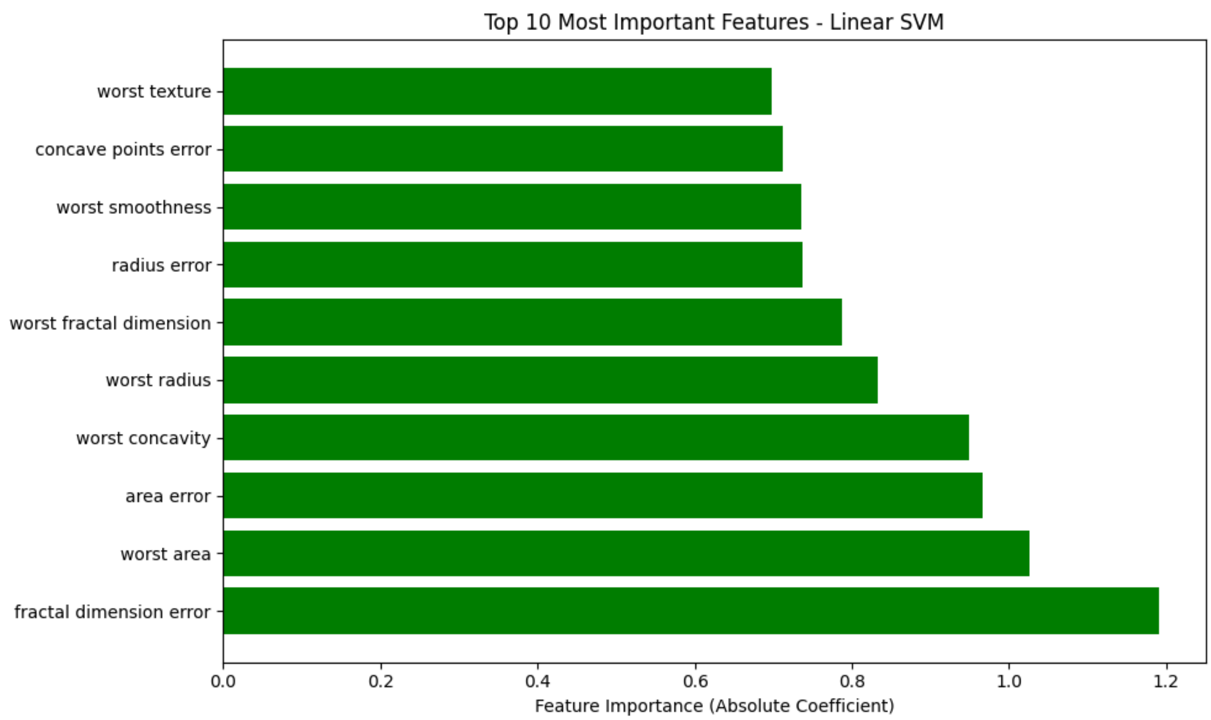
In [ ]:

In [37]:

```python
# Get feature importances from the best model
svm_best = svm.best_estimator_
svm_clf = svm_best.named_steps["svm"]

# Coefficients -> importance (absolute value)
svm_coef = svm_clf.coef_             # shape: (1, n_features) for binary cl
svm_imp = np.abs(svm_coef).flatten()


# Create DataFrame
svm_importance_df = pd.DataFrame({
    'Feature': X.columns,
```

```
    'Importance': svm_imp
}).sort_values('Importance', ascending=False)

print("\nTop 10 Most Important Features - Linear SVM:")
print(svm_importance_df.head(10))

# Plot
plt.figure(figsize=(10, 6))
plt.barh(svm_importance_df['Feature'].head(10), svm_importance_df['Importanc
plt.xlabel('Feature Importance (Absolute Coefficient)')
plt.title('Top 10 Most Important Features - Linear SVM')
plt.tight_layout()
plt.show()
```

```
Top 10 Most Important Features - Linear SVM:
                   Feature  Importance
19  fractal dimension error    1.190984
23               worst area    1.026678
13               area error    0.966771
26          worst concavity    0.948895
20             worst radius    0.832388
29  worst fractal dimension    0.787419
10             radius error    0.736903
24          worst smoothness    0.735840
17     concave points error    0.712115
21            worst texture    0.698103
```



In [38]:
```
# Clinical Interpretation:
# - Tumor SIZE (radius, area, perimeter) = strongest predictor
```

In [ ]:

In [ ]:

In [ ]: