```
# breast Cancer Dataset - Explainable AI Notebook
# Sections separated as notebook cells (use in Jupyter or VSCode
interactive)

# %%
# 0. Install required packages (uncomment if running in a fresh env)
!pip install pandas numpy scikit-learn matplotlib seaborn imbalanced-
learn lime shap xgboost lightgbm catboost pdpbox

# %%
# 1. Imports
import warnings
warnings.filterwarnings('ignore')

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.preprocessing import LabelEncoder, OneHotEncoder,
StandardScaler
from sklearn.impute import SimpleImputer
from imblearn.over_sampling import SMOTE

from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier

from sklearn.ensemble import BaggingClassifier, AdaBoostClassifier,
GradientBoostingClassifier, VotingClassifier, StackingClassifier,
ExtraTreesClassifier, HistGradientBoostingClassifier
import xgboost as xgb
import lightgbm as lgb
from catboost import CatBoostClassifier

from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.inspection import PartialDependenceDisplay

# XAI packages
import lime
from lime import lime_tabular
import shap

# %%
# 2. Load dataset
path = '/content/sample_data/archive (10).csv'
```

```
df = pd.read_csv(path)
print('Shape:', df.shape)
# quick peek
print(df.head())

# %%
# 3. Pre-processing
# 3.1 Handle nulls
print('\nMissing values per column:')
print(df.isnull().sum())

# For simplicity: numeric -> mean, categorical -> mode
num_cols =
df.select_dtypes(include=['int64','float64']).columns.tolist()
cat_cols =
df.select_dtypes(include=['object','category']).columns.tolist()

num_imputer = SimpleImputer(strategy='mean')
cat_imputer = SimpleImputer(strategy='most_frequent')

df[num_cols] = num_imputer.fit_transform(df[num_cols])
df[cat_cols] = cat_imputer.fit_transform(df[cat_cols])

print('\nMissing values after imputation:')
print(df.isnull().sum())

# 3.2 Encoding categorical features
# If target column named 'class' or 'LUNG_CANCER' adjust accordingly;
try to infer
possible_targets = ['diagnosis']
cols_lower = [c.lower() for c in df.columns]
# assume last column is target if none of above
if any(t in cols_lower for t in possible_targets):
    # find the matching column
    for c in df.columns:
        if c.lower() in possible_targets:
            target_col = c
            break
else:
    target_col = df.columns[-1]

print('\nTarget column chosen:', target_col)

# Encode target if needed
if df[target_col].dtype == 'object':
    le_target = LabelEncoder()
    df[target_col] = le_target.fit_transform(df[target_col])

# Encode categorical features using OneHot for low-cardinality,
LabelEncoder for binary
```

```
features = df.drop(columns=[target_col])

cat_features =
features.select_dtypes(include=['object','category']).columns.tolist()
print('Categorical features:', cat_features)

# If many categories, use one-hot
for c in cat_features:
    if df[c].nunique() <= 2:
        df[c] = LabelEncoder().fit_transform(df[c])
    else:
        df = pd.get_dummies(df, columns=[c], drop_first=True)

# 3.3 Feature/target split
X = df.drop(columns=[target_col])
y = df[target_col]

# 3.4 Correlation matrix
# compute correlation and show heatmap
corr = pd.concat([X, y], axis=1).corr()
plt.figure(figsize=(12,10))
sns.heatmap(corr, cmap='coolwarm', center=0)
plt.title('Correlation heatmap (features + target)')
plt.show()

# 3.5 Normalize / scale numeric features
scaler = StandardScaler()
X_scaled = pd.DataFrame(scaler.fit_transform(X), columns=X.columns)

# 3.6 Handle class imbalance with SMOTE
print('\nClass distribution before SMOTE:')
print(y.value_counts())

sm = SMOTE(random_state=42)
X_res, y_res = sm.fit_resample(X_scaled, y)
print('\nShape after SMOTE:', X_res.shape)
print('Class distribution after SMOTE:')
print(pd.Series(y_res).value_counts())

# 3.7 Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_res, y_res,
test_size=0.2, random_state=42, stratify=y_res)

# %%
# 4. Visualizations (pick any 3) - Scatter, Histogram, Boxplot
# scatter between first two numeric features
num_features = X.columns.tolist()
if len(num_features) >= 2:
    plt.figure(figsize=(6,4))
    plt.scatter(X_scaled[num_features[0]], X_scaled[num_features[1]],
```

```
c=y, alpha=0.6)
    plt.xlabel(num_features[0])
    plt.ylabel(num_features[1])
    plt.title('Scatter: {} vs {}'.format(num_features[0],
num_features[1]))
    plt.show()

# histogram of first numeric feature
plt.figure(figsize=(6,4))
plt.hist(X_scaled[num_features[0]], bins=30)
plt.title('Histogram: {}'.format(num_features[0]))
plt.show()

# boxplot of first numeric feature by target (use resampled data)
plt.figure(figsize=(6,4))
sns.boxplot(x=y_res, y=X_res[num_features[0]]) # Changed X_res[:,0] to
X_res[num_features[0]]
plt.title('Boxplot of {} by target'.format(num_features[0]))
plt.xlabel('Target')
plt.show()

# %%
# 5. Machine Learning models (5): Train models - no metrics printed as
requested
models_ml = {
    'LogisticRegression': LogisticRegression(max_iter=1000),
    'DecisionTree': DecisionTreeClassifier(),
    'KNN': KNeighborsClassifier(),
    'SVM': SVC(probability=True),
    'NeuralNet_MLP': MLPClassifier(hidden_layer_sizes=(64,32),
max_iter=500)
}

trained_ml = {}
for name, model in models_ml.items():
    model.fit(X_train, y_train)
    trained_ml[name] = model
    print(f'{name} trained')

# %%
# 6. Ensembles (10)
ensembles = {
    'Bagging': BaggingClassifier(n_estimators=50),
    'AdaBoost': AdaBoostClassifier(n_estimators=50),
    'GradientBoosting': GradientBoostingClassifier(n_estimators=50),
    'XGBoost': xgb.XGBClassifier(use_label_encoder=False,
eval_metric='logloss', verbosity=0),
    'LightGBM': lgb.LGBMClassifier(),
    'CatBoost': CatBoostClassifier(verbose=0),
    'Voting_hard': VotingClassifier(estimators=[('lr',
```

```
models_ml['LogisticRegression']), ('dt', models_ml['DecisionTree']),
('knn', models_ml['KNN'])], voting='hard'),
    'Stacking': StackingClassifier(estimators=[('lr',
models_ml['LogisticRegression']), ('dt', models_ml['DecisionTree'])],
final_estimator=MLPClassifier(max_iter=300)),
    'ExtraTrees': ExtraTreesClassifier(n_estimators=100),
    'HistGradientBoosting': HistGradientBoostingClassifier()
}

trained_ensembles = {}
for name, ens in ensembles.items():
    ens.fit(X_train, y_train)
    trained_ensembles[name] = ens
    print(f'{name} trained')

# %%
# 7. Deep Learning models (5 variants using Keras)
from tensorflow import keras
from tensorflow.keras import layers

# prepare tf datasets
X_train_np = np.array(X_train)
X_test_np = np.array(X_test)

def build_model_1(input_dim):
    model = keras.Sequential([
        layers.Input(shape=(input_dim,)),
        layers.Dense(64, activation='relu'),
        layers.Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
    return model

def build_model_2(input_dim):
    model = keras.Sequential([
        layers.Input(shape=(input_dim,)),
        layers.Dense(128, activation='relu'),
        layers.Dropout(0.3),
        layers.Dense(64, activation='relu'),
        layers.Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
    return model

def build_model_3(input_dim):
    model = keras.Sequential([
        layers.Input(shape=(input_dim,)),
        layers.Dense(32, activation='tanh'),
```

```
        layers.Dense(16, activation='tanh'),
        layers.Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer='sgd', loss='binary_crossentropy',
metrics=['accuracy'])
    return model

def build_model_4(input_dim):
    model = keras.Sequential([
        layers.Input(shape=(input_dim,)),
        layers.Dense(256, activation='relu'),
        layers.Dense(128, activation='relu'),
        layers.Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
    return model

def build_model_5(input_dim):
    model = keras.Sequential([
        layers.Input(shape=(input_dim,)),
        layers.Dense(64, activation='relu'),
        layers.Dense(64, activation='relu'),
        layers.Dense(32, activation='relu'),
        layers.Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
    return model

input_dim = X_train_np.shape[1]
models_dl = {
    'DL_model_1': build_model_1(input_dim),
    'DL_model_2': build_model_2(input_dim),
    'DL_model_3': build_model_3(input_dim),
    'DL_model_4': build_model_4(input_dim),
    'DL_model_5': build_model_5(input_dim)
}

# Fit briefly (few epochs) - adjust epochs as needed
for name, m in models_dl.items():
    print('Training', name)
    m.fit(X_train_np, np.array(y_train), epochs=5, batch_size=32,
validation_split=0.1, verbose=0)
    print(name, 'trained')

# %%
# 8. XAI: LIME for one black-box model (use SVM or NeuralNet_MLP)

explainer = lime_tabular.LimeTabularExplainer(
```

```
    training_data=np.array(X_train),
    feature_names=X_train.columns.tolist(),
    class_names=['No', 'Yes'],
    mode='classification'
)

# pick one instance from test set
i = 0
exp = explainer.explain_instance(
    X_test.iloc[i].values,
    trained_ml['SVM'].predict_proba,
    num_features=10
)
print('\nLIME explanation for one instance:')
print(exp.as_list())
# Optional (if using Jupyter): exp.show_in_notebook()

# %%
# 9. XAI: SHAP (example with XGBoost and one deep model)

shap.initjs()

# --- SHAP for XGBoost (tree-based model) ---
def xgb_predict_raw(X_input):
    return trained_ensembles['XGBoost'].predict_proba(X_input)[:, 1]

explainer_xgb = shap.Explainer(xgb_predict_raw, X_test)
shap_values_xgb = explainer_xgb(X_test)

# summary plot (global importance)
shap.summary_plot(shap_values_xgb, X_test, show=True)

# --- SHAP for Deep Learning (Keras model) ---
background = X_train.sample(n=50, random_state=42)
model_for_shap = models_dl['DL_model_1']

def keras_predict(X_input):
    # Return positive class probability only
    return model_for_shap.predict(X_input).flatten()

# Use KernelExplainer (slower, but model-agnostic)
explainer_kernel = shap.KernelExplainer(keras_predict, background)

# Take small subset for speed
X_sample = X_test.iloc[:20]

# Compute SHAP values
shap_values_keras = explainer_kernel.shap_values(X_sample)

# Handle version differences: SHAP may return list or np.array
```

```
if isinstance(shap_values_keras, list):
    shap_values_keras = shap_values_keras[0]

# SHAP summary plot for the deep learning model
shap.summary_plot(shap_values_keras, X_sample, show=True)

# %%
# 10. XAI: PDP and ICE (Partial Dependence & Individual Conditional
Expectation)

# Choose a numerical feature to visualize
feature_to_plot = 0  # you can change index here
feat_name = X_train.columns[feature_to_plot]

# PDP (average relationship)
PartialDependenceDisplay.from_estimator(
    trained_ml['DecisionTree'],
    X_test,
    [feat_name],
    kind='average'
)
plt.title(f'Partial Dependence Plot (Feature: {feat_name})')
plt.show()

# ICE (individual-level relationships)
PartialDependenceDisplay.from_estimator(
    trained_ml['DecisionTree'],
    X_test,
    [feat_name],
    kind='individual'
)
plt.title(f'Individual Conditional Expectation (Feature:
{feat_name})')
plt.show()

# %%
# 11. Inference (2 lines)
print('\nInference:')
print('1) After pre-processing and balancing, models learned from a
more representative dataset; scaling and encoding stabilized
training.')
print('2) SHAP and LIME explain model predictions, highlighting the
most influential features for both global and individual
predictions.')
```

```
Requirement already satisfied: pandas in
/usr/local/lib/python3.12/dist-packages (2.2.2)
Requirement already satisfied: numpy in
/usr/local/lib/python3.12/dist-packages (2.0.2)
Requirement already satisfied: scikit-learn in
```

```
/usr/local/lib/python3.12/dist-packages (1.6.1)
Requirement already satisfied: matplotlib in
/usr/local/lib/python3.12/dist-packages (3.10.0)
Requirement already satisfied: seaborn in
/usr/local/lib/python3.12/dist-packages (0.13.2)
Requirement already satisfied: imbalanced-learn in
/usr/local/lib/python3.12/dist-packages (0.14.0)
Requirement already satisfied: lime in /usr/local/lib/python3.12/dist-
packages (0.2.0.1)
Requirement already satisfied: shap in /usr/local/lib/python3.12/dist-
packages (0.49.1)
Requirement already satisfied: xgboost in
/usr/local/lib/python3.12/dist-packages (3.1.1)
Requirement already satisfied: lightgbm in
/usr/local/lib/python3.12/dist-packages (4.6.0)
Requirement already satisfied: catboost in
/usr/local/lib/python3.12/dist-packages (1.2.8)
Requirement already satisfied: pdpbox in
/usr/local/lib/python3.12/dist-packages (0.3.0)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.12/dist-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.12/dist-packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in
/usr/local/lib/python3.12/dist-packages (from pandas) (2025.2)
Requirement already satisfied: scipy>=1.6.0 in
/usr/local/lib/python3.12/dist-packages (from scikit-learn) (1.16.3)
Requirement already satisfied: joblib>=1.2.0 in
/usr/local/lib/python3.12/dist-packages (from scikit-learn) (1.5.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in
/usr/local/lib/python3.12/dist-packages (from scikit-learn) (3.6.0)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.12/dist-packages (from matplotlib) (1.3.3)
Requirement already satisfied: cycler>=0.10 in
/usr/local/lib/python3.12/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.12/dist-packages (from matplotlib) (4.60.1)
Requirement already satisfied: kiwisolver>=1.3.1 in
/usr/local/lib/python3.12/dist-packages (from matplotlib) (1.4.9)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.12/dist-packages (from matplotlib) (25.0)
Requirement already satisfied: pillow>=8 in
/usr/local/lib/python3.12/dist-packages (from matplotlib) (11.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.12/dist-packages (from matplotlib) (3.2.5)
Requirement already satisfied: tqdm in /usr/local/lib/python3.12/dist-
packages (from lime) (4.67.1)
Requirement already satisfied: scikit-image>=0.12 in
/usr/local/lib/python3.12/dist-packages (from lime) (0.25.2)
```

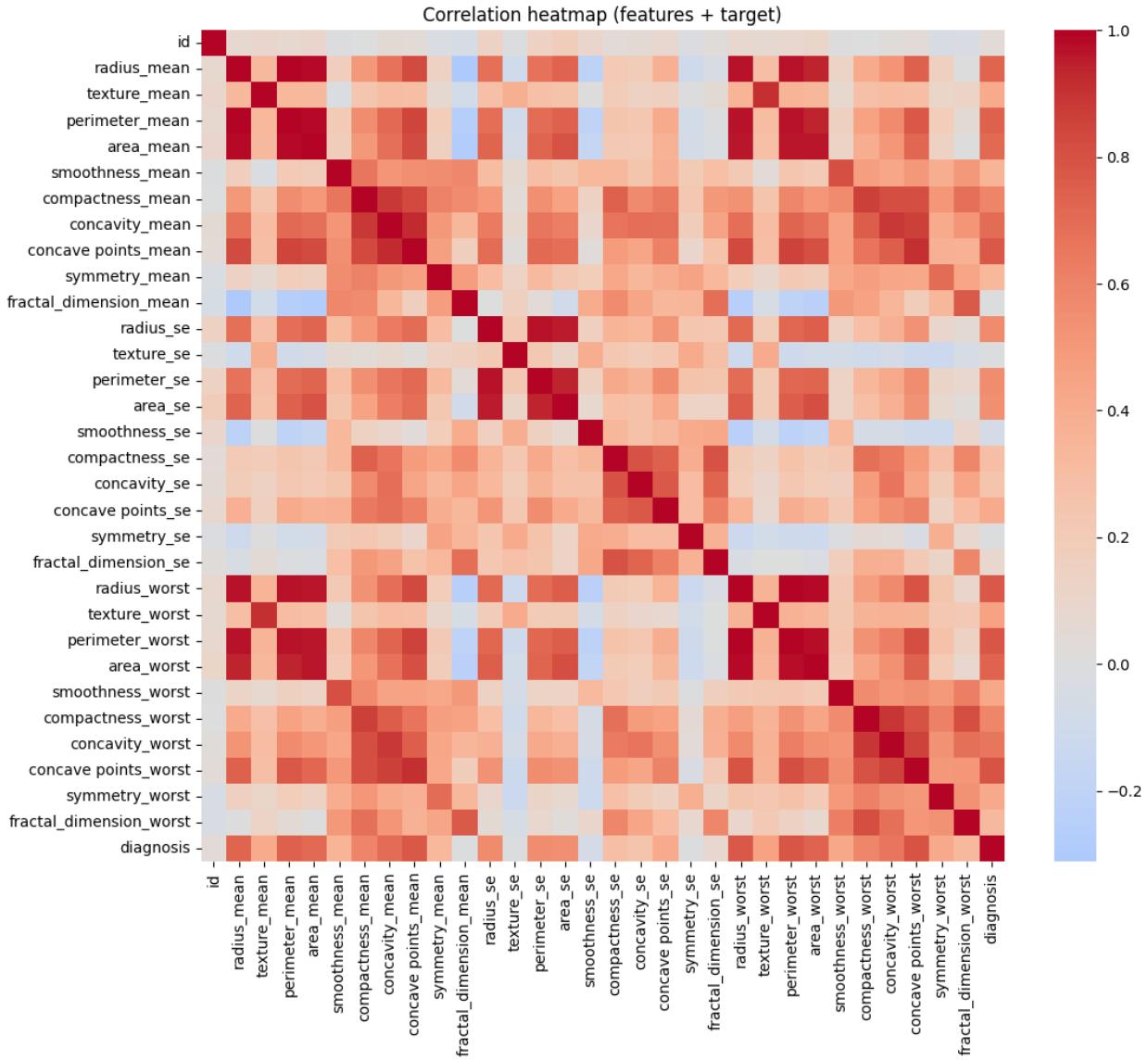
```
Requirement already satisfied: slicer==0.0.8 in
/usr/local/lib/python3.12/dist-packages (from shap) (0.0.8)
Requirement already satisfied: numba>=0.54 in
/usr/local/lib/python3.12/dist-packages (from shap) (0.60.0)
Requirement already satisfied: cloudpickle in
/usr/local/lib/python3.12/dist-packages (from shap) (3.1.1)
Requirement already satisfied: typing-extensions in
/usr/local/lib/python3.12/dist-packages (from shap) (4.15.0)
Requirement already satisfied: nvidia-nccl-cu12 in
/usr/local/lib/python3.12/dist-packages (from xgboost) (2.27.3)
Requirement already satisfied: graphviz in
/usr/local/lib/python3.12/dist-packages (from catboost) (0.21)
Requirement already satisfied: plotly in
/usr/local/lib/python3.12/dist-packages (from catboost) (5.24.1)
Requirement already satisfied: six in /usr/local/lib/python3.12/dist-
packages (from catboost) (1.17.0)
Requirement already satisfied: pqdm>=0.2.0 in
/usr/local/lib/python3.12/dist-packages (from pdpbox) (0.2.0)
Requirement already satisfied: psutil>=5.9.0 in
/usr/local/lib/python3.12/dist-packages (from pdpbox) (5.9.5)
Requirement already satisfied: pytest in
/usr/local/lib/python3.12/dist-packages (from pdpbox) (8.4.2)
Requirement already satisfied: setuptools in
/usr/local/lib/python3.12/dist-packages (from pdpbox) (75.2.0)
Requirement already satisfied: sphinx>=5.0.2 in
/usr/local/lib/python3.12/dist-packages (from pdpbox) (8.2.3)
Requirement already satisfied: sphinx-rtd-theme>=1.1.1 in
/usr/local/lib/python3.12/dist-packages (from pdpbox) (3.0.2)
Requirement already satisfied: numpydoc>=1.4.0 in
/usr/local/lib/python3.12/dist-packages (from pdpbox) (1.9.0)
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in
/usr/local/lib/python3.12/dist-packages (from numba>=0.54->shap)
(0.43.0)
Requirement already satisfied: tenacity>=6.2.0 in
/usr/local/lib/python3.12/dist-packages (from plotly->catboost)
(8.5.0)
Requirement already satisfied: bounded-pool-executor in
/usr/local/lib/python3.12/dist-packages (from pqdm>=0.2.0->pdpbox)
(0.0.3)
Requirement already satisfied: networkx>=3.0 in
/usr/local/lib/python3.12/dist-packages (from scikit-image>=0.12-
>lime) (3.5)
Requirement already satisfied: imageio!=2.35.0,>=2.33 in
/usr/local/lib/python3.12/dist-packages (from scikit-image>=0.12-
>lime) (2.37.0)
Requirement already satisfied: tifffile>=2022.8.12 in
/usr/local/lib/python3.12/dist-packages (from scikit-image>=0.12-
>lime) (2025.10.16)
Requirement already satisfied: lazy-loader>=0.4 in
```

```
/usr/local/lib/python3.12/dist-packages (from scikit-image>=0.12->lime) (0.4)
Requirement already satisfied: sphinxcontrib-applehelp>=1.0.7 in /usr/local/lib/python3.12/dist-packages (from sphinx>=5.0.2->pdpbox) (2.0.0)
Requirement already satisfied: sphinxcontrib-devhelp>=1.0.6 in /usr/local/lib/python3.12/dist-packages (from sphinx>=5.0.2->pdpbox) (2.0.0)
Requirement already satisfied: sphinxcontrib-htmlhelp>=2.0.6 in /usr/local/lib/python3.12/dist-packages (from sphinx>=5.0.2->pdpbox) (2.1.0)
Requirement already satisfied: sphinxcontrib-jsmath>=1.0.1 in /usr/local/lib/python3.12/dist-packages (from sphinx>=5.0.2->pdpbox) (1.0.1)
Requirement already satisfied: sphinxcontrib-qthelp>=1.0.6 in /usr/local/lib/python3.12/dist-packages (from sphinx>=5.0.2->pdpbox) (2.0.0)
Requirement already satisfied: sphinxcontrib-serializinghtml>=1.1.9 in /usr/local/lib/python3.12/dist-packages (from sphinx>=5.0.2->pdpbox) (2.0.0)
Requirement already satisfied: Jinja2>=3.1 in /usr/local/lib/python3.12/dist-packages (from sphinx>=5.0.2->pdpbox) (3.1.6)
Requirement already satisfied: Pygments>=2.17 in /usr/local/lib/python3.12/dist-packages (from sphinx>=5.0.2->pdpbox) (2.19.2)
Requirement already satisfied: docutils<0.22,>=0.20 in /usr/local/lib/python3.12/dist-packages (from sphinx>=5.0.2->pdpbox) (0.21.2)
Requirement already satisfied: snowballstemmer>=2.2 in /usr/local/lib/python3.12/dist-packages (from sphinx>=5.0.2->pdpbox) (3.0.1)
Requirement already satisfied: babel>=2.13 in /usr/local/lib/python3.12/dist-packages (from sphinx>=5.0.2->pdpbox) (2.17.0)
Requirement already satisfied: alabaster>=0.7.14 in /usr/local/lib/python3.12/dist-packages (from sphinx>=5.0.2->pdpbox) (1.0.0)
Requirement already satisfied: imagesize>=1.3 in /usr/local/lib/python3.12/dist-packages (from sphinx>=5.0.2->pdpbox) (1.4.1)
Requirement already satisfied: requests>=2.30.0 in /usr/local/lib/python3.12/dist-packages (from sphinx>=5.0.2->pdpbox) (2.32.4)
Requirement already satisfied: roman-numerals-py>=1.0.0 in /usr/local/lib/python3.12/dist-packages (from sphinx>=5.0.2->pdpbox) (3.1.0)
Requirement already satisfied: sphinxcontrib-jquery<5,>=4 in /usr/local/lib/python3.12/dist-packages (from sphinx-rtd-theme>=1.1.1-
```

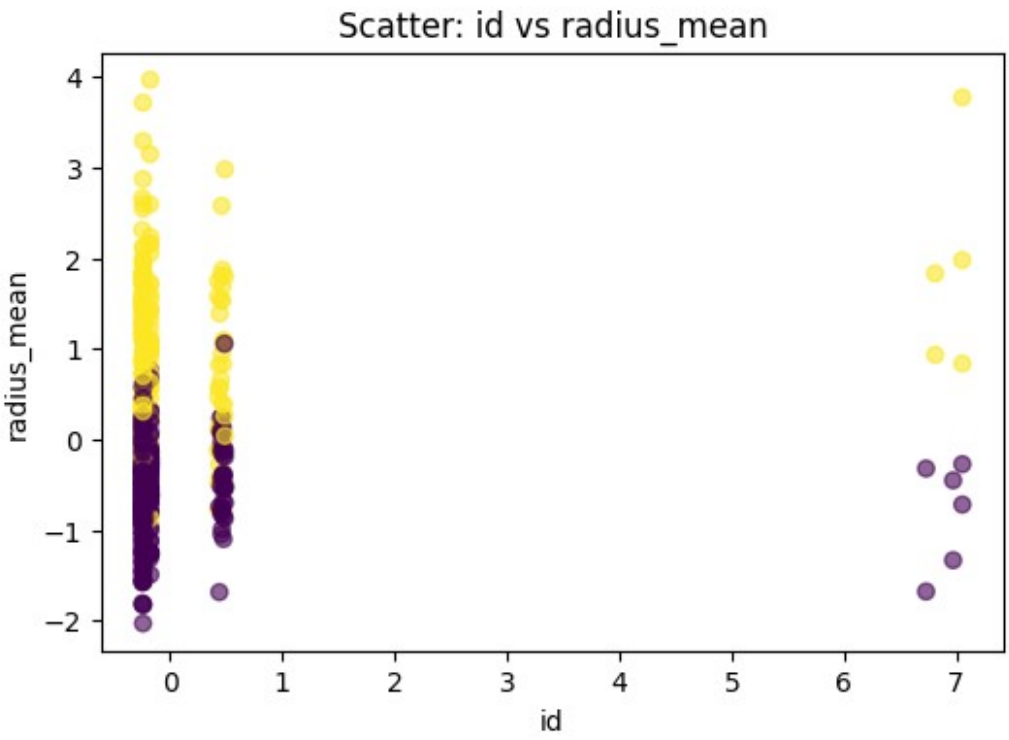
```
>pdpbox) (4.1)
Requirement already satisfied: iniconfig>=1 in
/usr/local/lib/python3.12/dist-packages (from pytest->pdpbox) (2.3.0)
Requirement already satisfied: pluggy<2,>=1.5 in
/usr/local/lib/python3.12/dist-packages (from pytest->pdpbox) (1.6.0)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.12/dist-packages (from Jinja2>=3.1-
>sphinx>=5.0.2->pdpbox) (3.0.3)
Requirement already satisfied: charset_normalizer<4,>=2 in
/usr/local/lib/python3.12/dist-packages (from requests>=2.30.0-
>sphinx>=5.0.2->pdpbox) (3.4.4)
Requirement already satisfied: idna<4,>=2.5 in
/usr/local/lib/python3.12/dist-packages (from requests>=2.30.0-
>sphinx>=5.0.2->pdpbox) (3.11)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.12/dist-packages (from requests>=2.30.0-
>sphinx>=5.0.2->pdpbox) (2.5.0)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.12/dist-packages (from requests>=2.30.0-
>sphinx>=5.0.2->pdpbox) (2025.10.5)
Shape: (569, 32)
         id diagnosis  radius_mean  texture_mean  perimeter_mean
area_mean  \
0    842302         M        17.99         10.38          122.80
1001.0
1    842517         M        20.57         17.77          132.90
1326.0
2  84300903         M        19.69         21.25          130.00
1203.0
3  84348301         M        11.42         20.38           77.58
386.1
4  84358402         M        20.29         14.34          135.10
1297.0

   smoothness_mean  compactness_mean  concavity_mean  concave
points_mean  \
0          0.11840           0.27760          0.3001
0.14710
1          0.08474           0.07864          0.0869
0.07017
2          0.10960           0.15990          0.1974
0.12790
3          0.14250           0.28390          0.2414
0.10520
4          0.10030           0.13280          0.1980
0.10430

   ...  radius_worst  texture_worst  perimeter_worst  area_worst  \
0  ...         25.38          17.33           184.60      2019.0
```

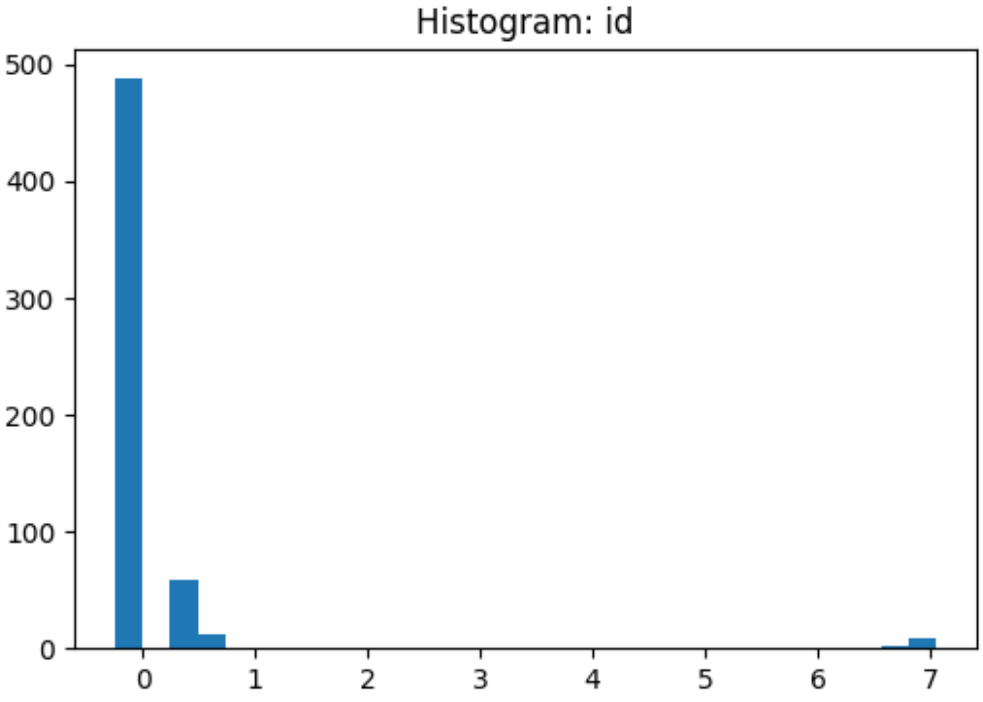
```
1  ...         24.99          23.41           158.80      1956.0
2  ...         23.57          25.53           152.50      1709.0
3  ...         14.91          26.50            98.87       567.7
4  ...         22.54          16.67           152.20      1575.0

   smoothness_worst  compactness_worst  concavity_worst  concave
points_worst  \
0            0.1622             0.6656           0.7119
0.2654
1            0.1238             0.1866           0.2416
0.1860
2            0.1444             0.4245           0.4504
0.2430
3            0.2098             0.8663           0.6869
0.2575
4            0.1374             0.2050           0.4000
0.1625

   symmetry_worst  fractal_dimension_worst
0          0.4601                  0.11890
1          0.2750                  0.08902
2          0.3613                  0.08758
3          0.6638                  0.17300
4          0.2364                  0.07678

[5 rows x 32 columns]

Missing values per column:
id                         0
diagnosis                  0
radius_mean                0
texture_mean               0
perimeter_mean             0
area_mean                  0
smoothness_mean            0
compactness_mean           0
concavity_mean             0
concave points_mean        0
symmetry_mean              0
fractal_dimension_mean     0
radius_se                  0
texture_se                 0
perimeter_se               0
area_se                    0
smoothness_se              0
compactness_se             0
concavity_se               0
concave points_se          0
symmetry_se                0
fractal_dimension_se       0
```

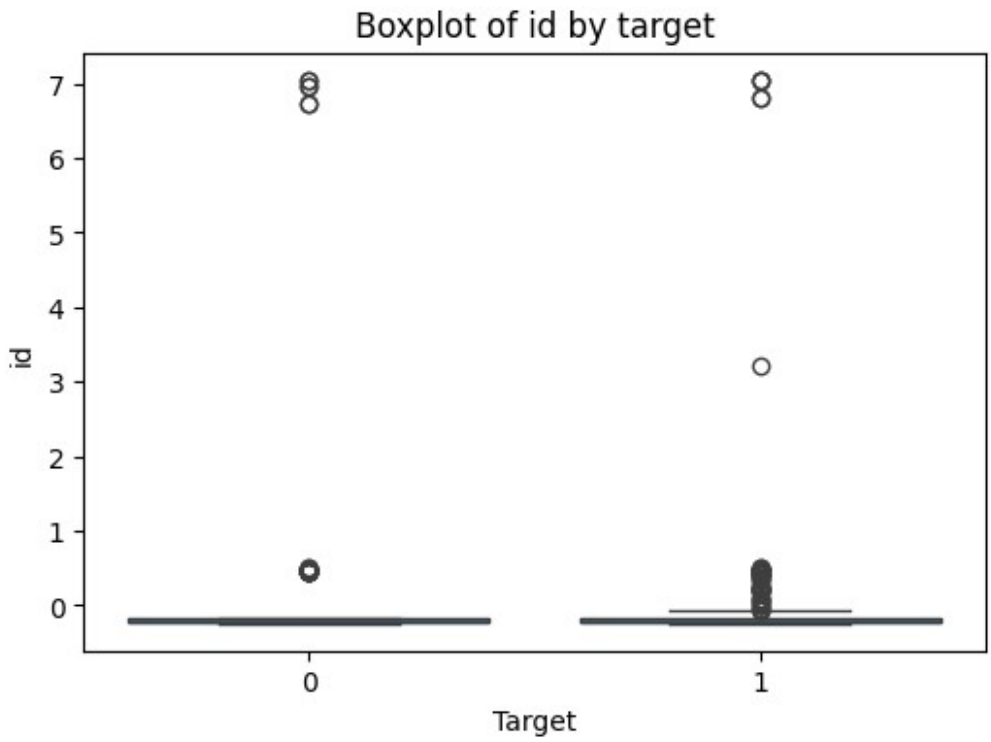
```
radius_worst               0
texture_worst              0
perimeter_worst            0
area_worst                 0
smoothness_worst           0
compactness_worst          0
concavity_worst            0
concave points_worst       0
symmetry_worst             0
fractal_dimension_worst    0
dtype: int64

Missing values after imputation:
id                         0
diagnosis                  0
radius_mean                0
texture_mean               0
perimeter_mean             0
area_mean                  0
smoothness_mean            0
compactness_mean           0
concavity_mean             0
concave points_mean        0
symmetry_mean              0
fractal_dimension_mean     0
radius_se                  0
texture_se                 0
perimeter_se               0
area_se                    0
smoothness_se              0
compactness_se             0
concavity_se               0
concave points_se          0
symmetry_se                0
fractal_dimension_se       0
radius_worst               0
texture_worst              0
perimeter_worst            0
area_worst                 0
smoothness_worst           0
compactness_worst          0
concavity_worst            0
concave points_worst       0
symmetry_worst             0
fractal_dimension_worst    0
dtype: int64

Target column chosen: diagnosis
Categorical features: []
```




```
Class distribution before SMOTE:
diagnosis
0    357
1    212
Name: count, dtype: int64

Shape after SMOTE: (714, 31)
Class distribution after SMOTE:
diagnosis
1    357
0    357
Name: count, dtype: int64
```


Scatter: id vs radius_mean
radius_mean
id
4
3
2
1
0
−1
−2
0
1
2
3
4
5
6
7


Histogram: id
500
400
300
200
100
0
0
1
2
3
4
5
6
7



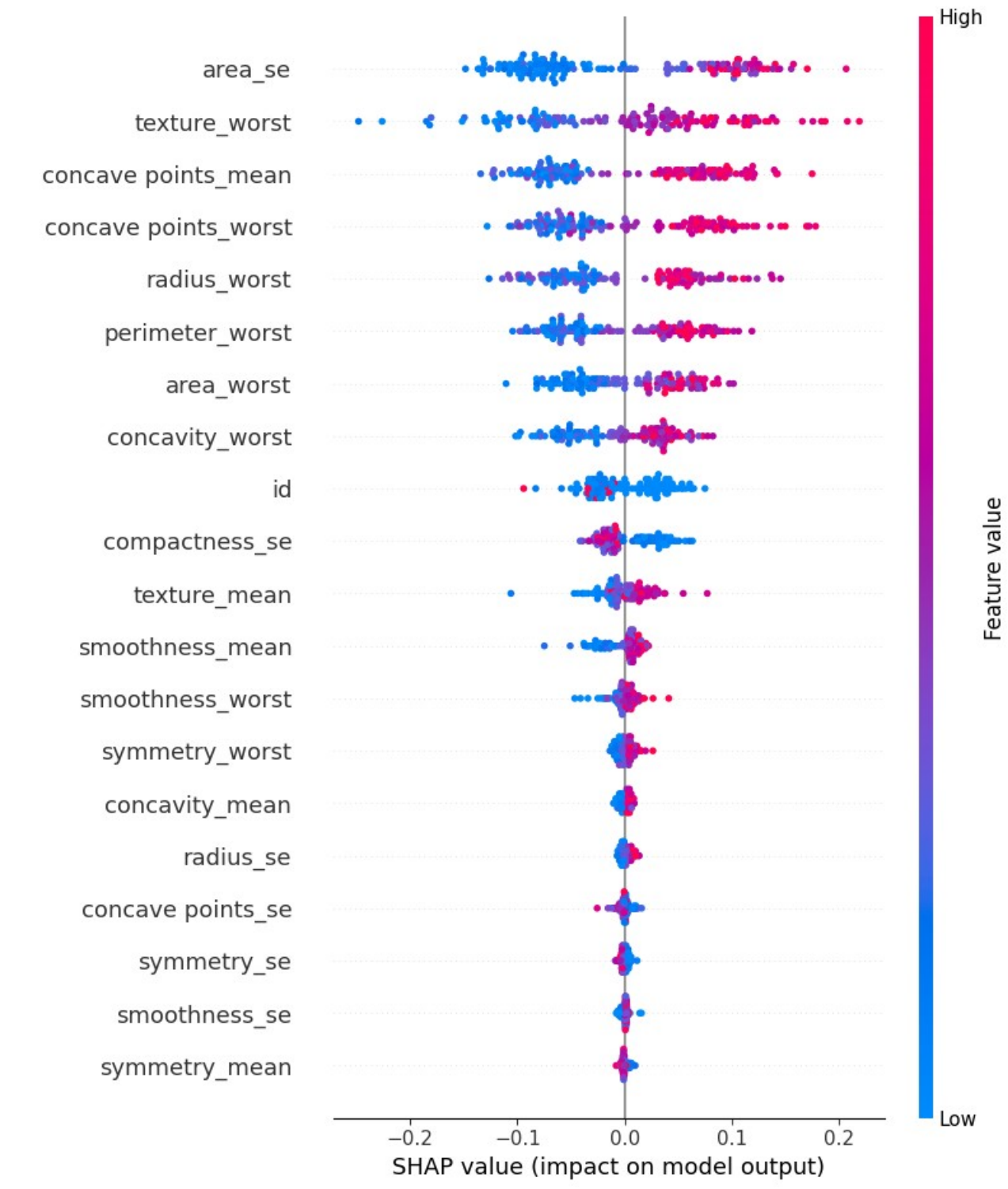

```
LogisticRegression trained
DecisionTree trained
KNN trained
SVM trained
NeuralNet_MLP trained
Bagging trained
AdaBoost trained
GradientBoosting trained
XGBoost trained
[LightGBM] [Warning] Found whitespace in feature_names, replace with
underlines
[LightGBM] [Info] Number of positive: 286, number of negative: 285
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead
of testing was 0.000357 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 5906
[LightGBM] [Info] Number of data points in the train set: 571, number
of used features: 31
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500876 ->
initscore=0.003503
[LightGBM] [Info] Start training from score 0.003503
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
```

```
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
```

```
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
LightGBM trained
CatBoost trained
Voting_hard trained
Stacking trained
ExtraTrees trained
HistGradientBoosting trained
Training DL_model_1
DL_model_1 trained
Training DL_model_2
DL_model_2 trained
Training DL_model_3
DL_model_3 trained
Training DL_model_4
DL_model_4 trained
Training DL_model_5
DL_model_5 trained

LIME explanation for one instance:
[('radius_worst > 0.94', 0.13056809453579038), ('concave points_mean >
0.93', 0.12499396404449922), ('perimeter_worst > 0.94',
0.12211887157752657), ('area_worst > 0.76', 0.12096079585749306),
('symmetry_worst > 0.58', 0.10771269689944485), ('concavity_mean >
0.76', 0.10441307358250997), ('radius_se > 0.58',
0.10168692926547362), ('compactness_se > 0.47', -0.09980286141563913),
('concavity_worst > 0.74', 0.09910934153638686),
('fractal_dimension_se > 0.29', -0.09462177513108162)]

<IPython.core.display.HTML object>
```

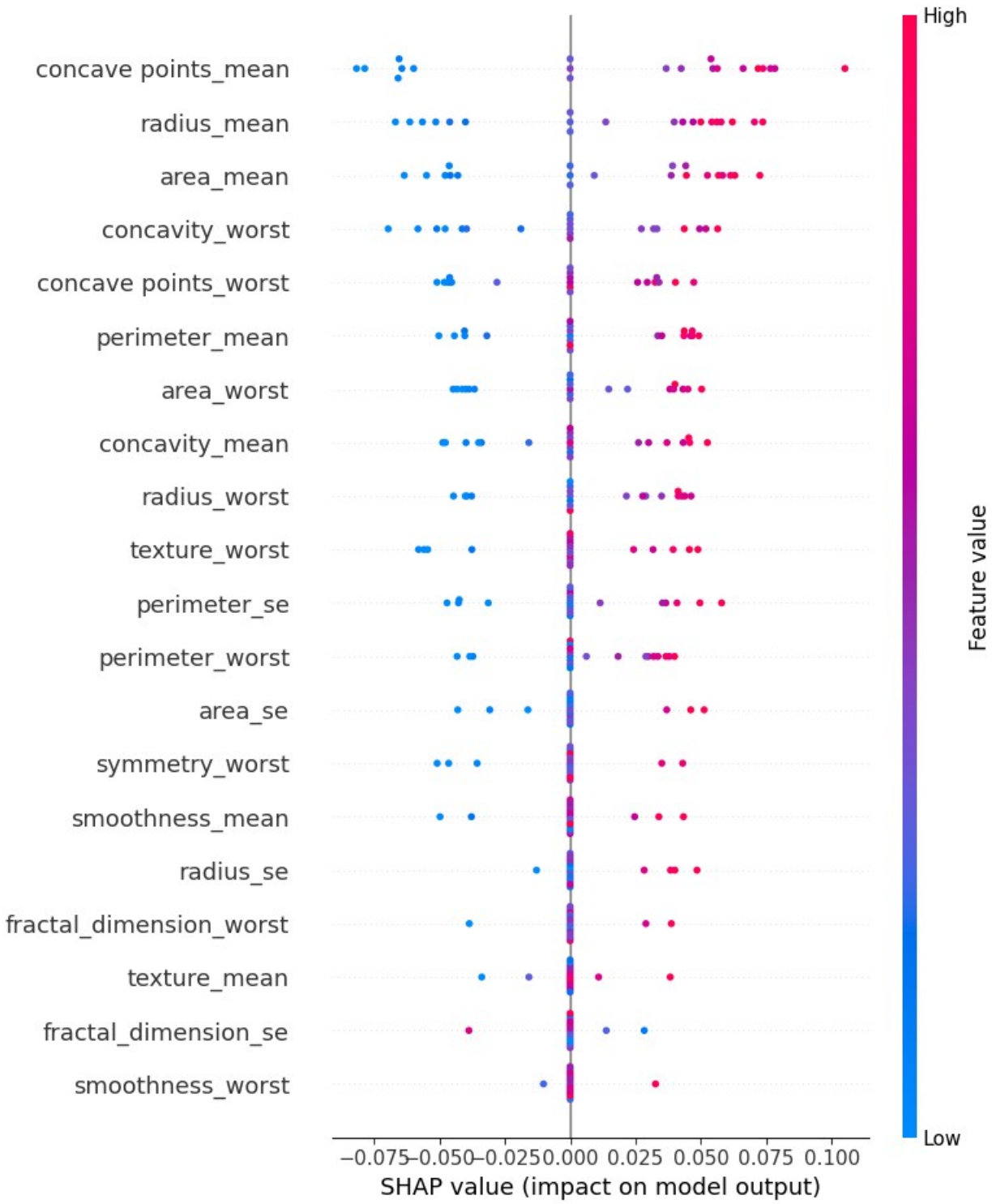
```
PermutationExplainer explainer: 144it [00:15,  3.46it/s]
```

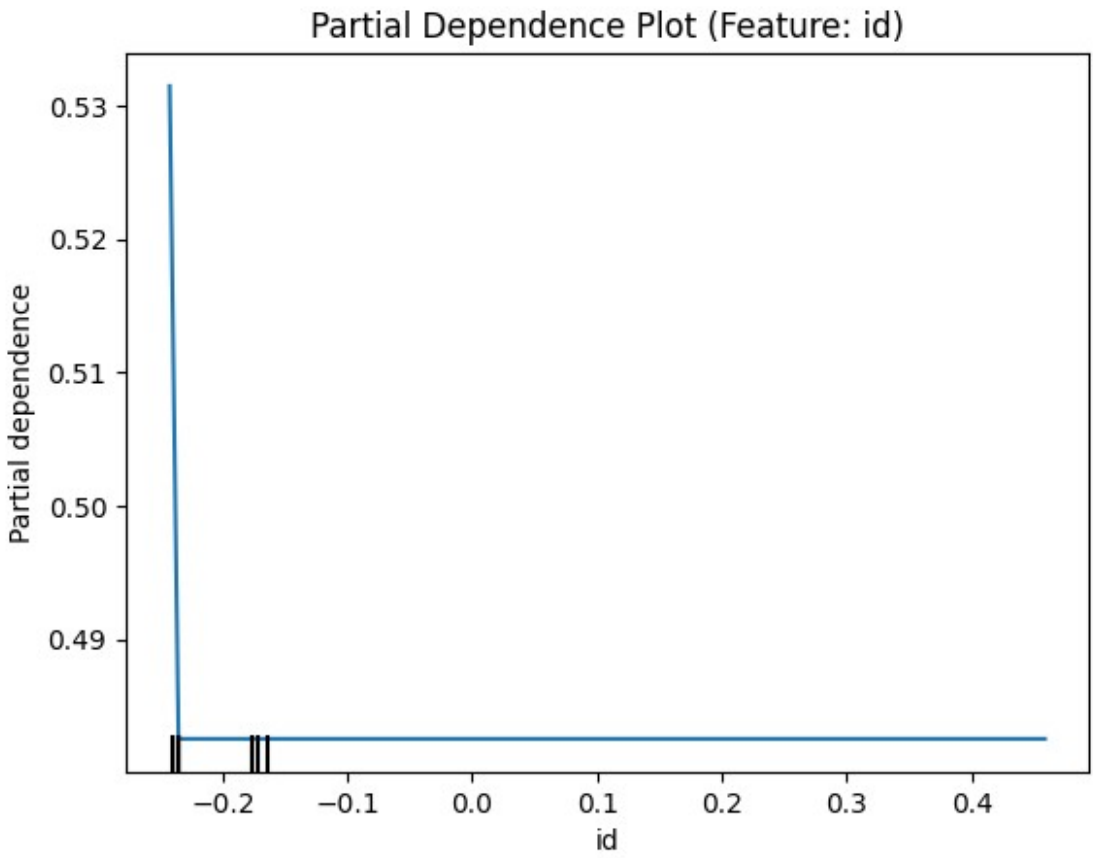


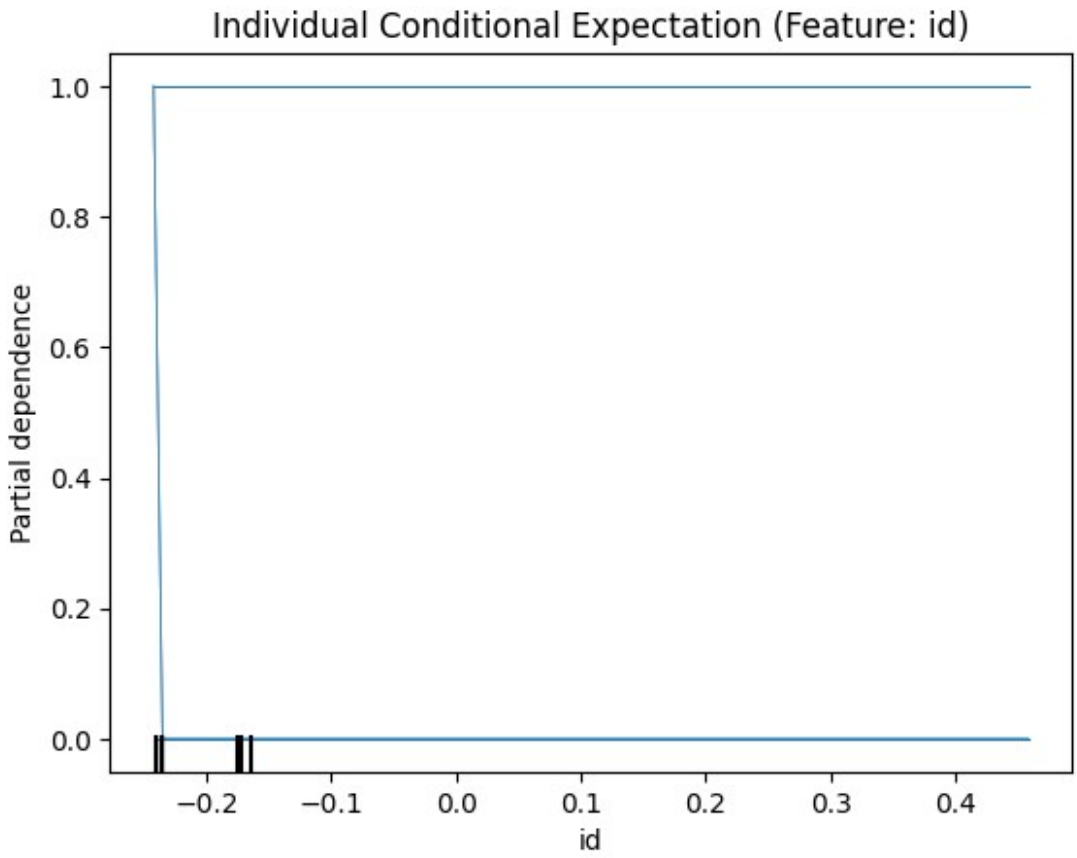

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
```

{"model_id":"1b0ccfc0f92d4ed88749d96ef8557153","version_major":2,"version_minor":0}

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
3297/3297 ━━━━━━━━━━━━━━━━━━━━ 5s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
3297/3297 ━━━━━━━━━━━━━━━━━━━━ 4s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 64ms/step
3297/3297 ━━━━━━━━━━━━━━━━━━━━ 5s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step
3297/3297 ━━━━━━━━━━━━━━━━━━━━ 4s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
3297/3297 ━━━━━━━━━━━━━━━━━━━━ 5s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
3297/3297 ━━━━━━━━━━━━━━━━━━━━ 4s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
3297/3297 ━━━━━━━━━━━━━━━━━━━━ 4s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 67ms/step
3297/3297 ━━━━━━━━━━━━━━━━━━━━ 4s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step
3297/3297 ━━━━━━━━━━━━━━━━━━━━ 5s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 62ms/step
3297/3297 ━━━━━━━━━━━━━━━━━━━━ 4s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
3297/3297 ━━━━━━━━━━━━━━━━━━━━ 4s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 65ms/step
3297/3297 ━━━━━━━━━━━━━━━━━━━━ 5s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
3297/3297 ━━━━━━━━━━━━━━━━━━━━ 4s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step
3297/3297 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
3297/3297 ━━━━━━━━━━━━━━━━━━━━ 4s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
3297/3297 ━━━━━━━━━━━━━━━━━━━━ 5s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
3297/3297 ━━━━━━━━━━━━━━━━━━━━ 4s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
3297/3297 ━━━━━━━━━━━━━━━━━━━━ 4s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
3297/3297 ━━━━━━━━━━━━━━━━━━━━ 4s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
3297/3297 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step
```


concave points_mean
radius_mean
area_mean
concavity_worst
concave points_worst
perimeter_mean
area_worst
concavity_mean
radius_worst
texture_worst
perimeter_se
perimeter_worst
area_se
symmetry_worst
smoothness_mean
radius_se
fractal_dimension_worst
texture_mean
fractal_dimension_se
smoothness_worst
−0.075 −0.050 −0.025 0.000 0.025 0.050 0.075 0.100
SHAP value (impact on model output)
High
Feature value
Low


Partial Dependence Plot (Feature: id)
Partial dependence
0.53
0.52
0.51
0.50
0.49
−0.2
−0.1
0.0
0.1
0.2
0.3
0.4
id



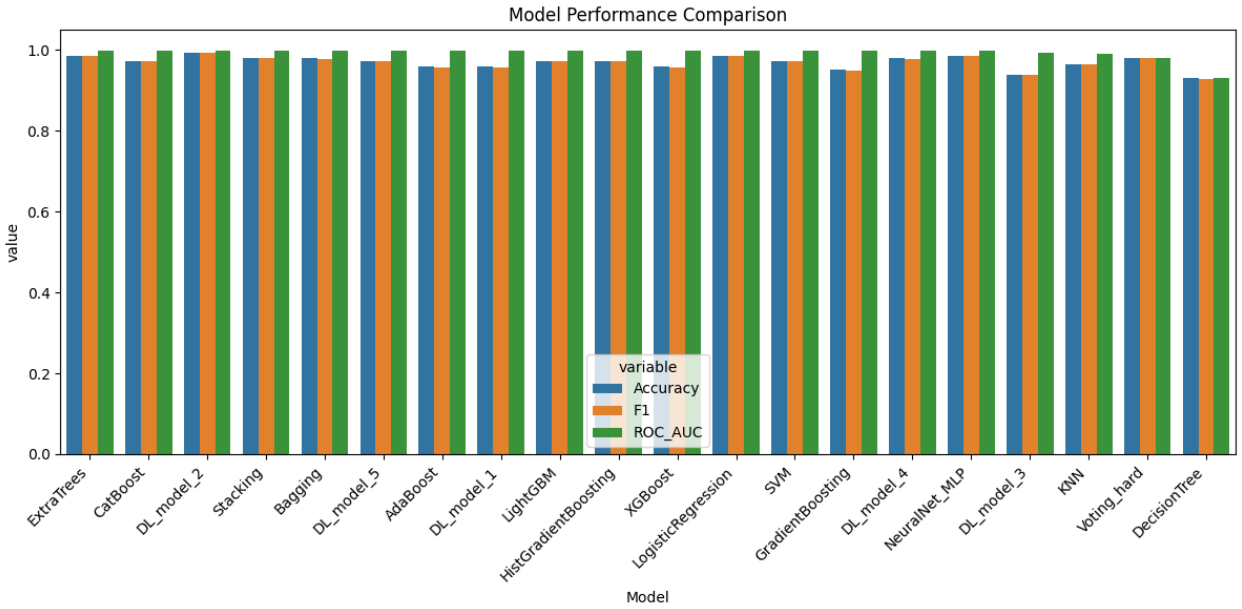

```
Inference:
1) After pre-processing and balancing, models learned from a more
representative dataset; scaling and encoding stabilized training.
2) SHAP and LIME explain model predictions, highlighting the most
influential features for both global and individual predictions.

# %%
# 12. Evaluation & Model Comparison

from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, roc_auc_score, confusion_matrix,
RocCurveDisplay

# --- Helper function for metrics ---
def evaluate_model(name, model, X_test, y_test):
    y_pred = model.predict(X_test)
    if hasattr(model, "predict_proba"):
        y_proba = model.predict_proba(X_test)[:, 1]
    else:
        # fallback for models without predict_proba (e.g., SVM without
prob=True)
        try:
```

```
            y_proba = model.decision_function(X_test)
        except:
            y_proba = y_pred

    return {
        "Model": name,
        "Accuracy": accuracy_score(y_test, y_pred),
        "Precision": precision_score(y_test, y_pred),
        "Recall": recall_score(y_test, y_pred),
        "F1": f1_score(y_test, y_pred),
        "ROC_AUC": roc_auc_score(y_test, y_proba)
    }

# --- Evaluate ML models ---
results_ml = [evaluate_model(name, model, X_test, y_test) for name,
model in trained_ml.items()]

# --- Evaluate Ensemble models ---
results_ens = [evaluate_model(name, model, X_test, y_test) for name,
model in trained_ensembles.items()]

# --- Evaluate Deep Learning models ---
results_dl = []
for name, model in models_dl.items():
    y_proba = model.predict(X_test_np).flatten()
    y_pred = (y_proba > 0.5).astype(int)
    results_dl.append({
        "Model": name,
        "Accuracy": accuracy_score(y_test, y_pred),
        "Precision": precision_score(y_test, y_pred),
        "Recall": recall_score(y_test, y_pred),
        "F1": f1_score(y_test, y_pred),
        "ROC_AUC": roc_auc_score(y_test, y_proba)
    })

# --- Combine all results ---
df_results = pd.DataFrame(results_ml + results_ens + results_dl)
df_results = df_results.sort_values(by="ROC_AUC",
ascending=False).reset_index(drop=True)

print("\n Model Performance Summary:")
print(df_results.round(4))

# --- Plot performance comparison ---
plt.figure(figsize=(12,6))
sns.barplot(data=df_results.melt(id_vars="Model",
value_vars=["Accuracy","F1","ROC_AUC"]),
            x="Model", y="value", hue="variable")
plt.title("Model Performance Comparison")
plt.xticks(rotation=45, ha='right')
```

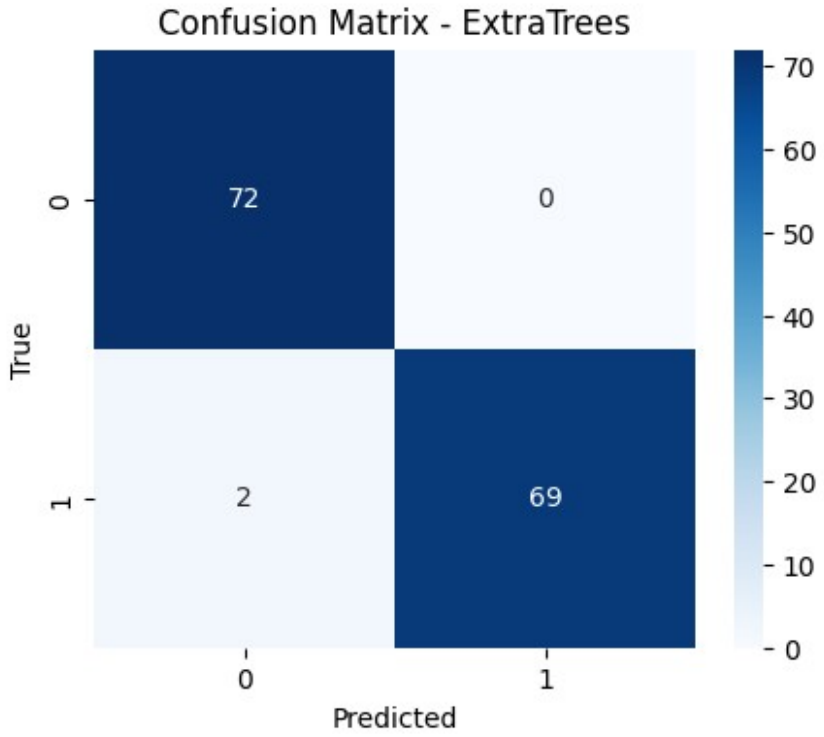
```
plt.tight_layout()
plt.show()

# --- Confusion Matrix for Best Model ---
best_model_name = df_results.iloc[0]['Model']
print(f"\nBest model based on ROC-AUC: {best_model_name}")

# Get model object
if best_model_name in trained_ml:
    best_model = trained_ml[best_model_name]
elif best_model_name in trained_ensembles:
    best_model = trained_ensembles[best_model_name]
else:
    best_model = models_dl[best_model_name]

# Predict and plot confusion matrix
if "DL_model" in best_model_name:
    y_proba = best_model.predict(X_test_np).flatten()
    y_pred = (y_proba > 0.5).astype(int)
else:
    y_pred = best_model.predict(X_test)
cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(5,4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
plt.title(f"Confusion Matrix - {best_model_name}")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.show()

# --- ROC Curve for Top 3 Models ---
top3 = df_results['Model'].head(3).tolist()
plt.figure(figsize=(8,6))
for mname in top3:
    if mname in trained_ml:
        model = trained_ml[mname]
    elif mname in trained_ensembles:
        model = trained_ensembles[mname]
    else:
        model = models_dl[mname]

    if "DL_model" in mname:
        y_proba = model.predict(X_test_np).flatten()
    else:
        y_proba = model.predict_proba(X_test)[:,1]
    RocCurveDisplay.from_predictions(y_test, y_proba, name=mname,
ax=plt.gca())

plt.title("ROC Curves - Top 3 Models")
plt.show()
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 170ms/step

 Model Performance Summary:
```

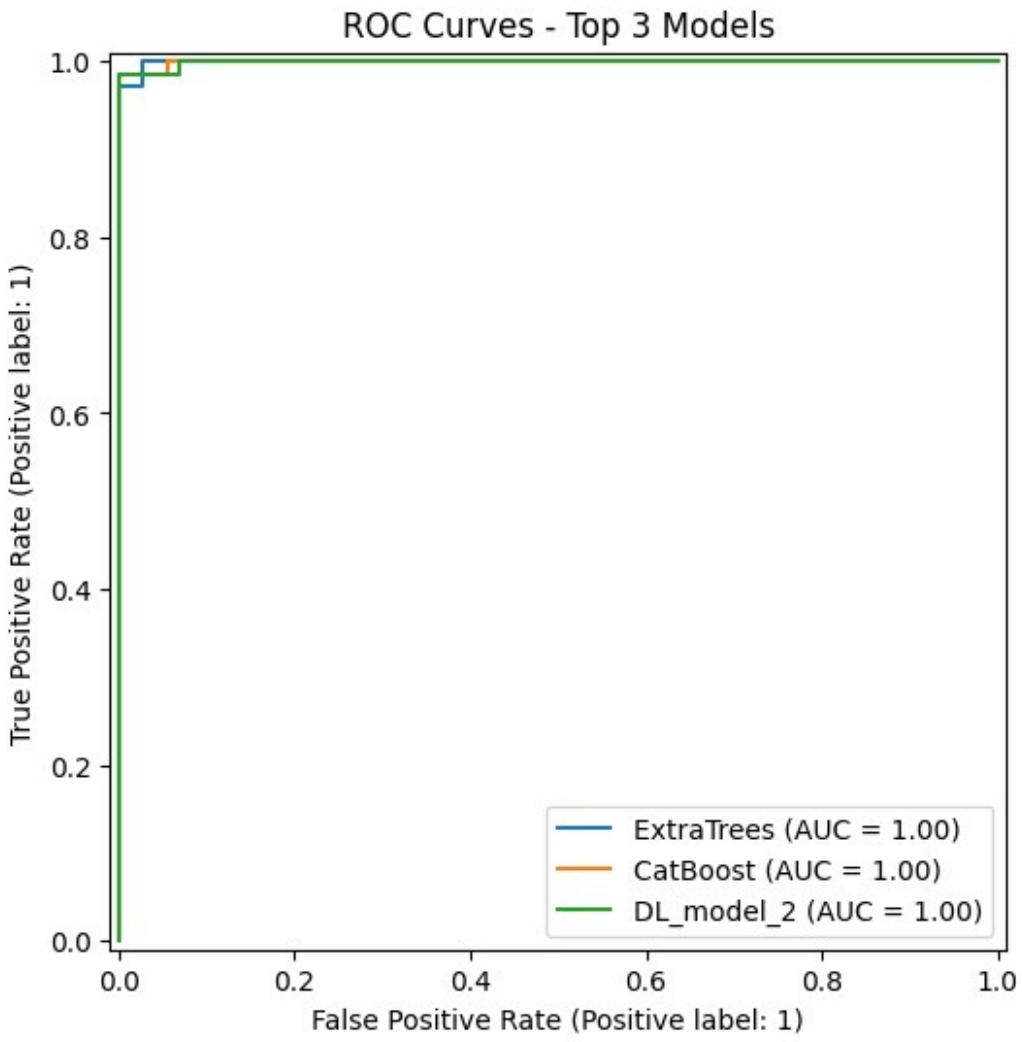
	Model	Accuracy	Precision	Recall	F1	ROC_AUC
0	ExtraTrees	0.9860	1.0000	0.9718	0.9857	0.9992
1	CatBoost	0.9720	1.0000	0.9437	0.9710	0.9992
2	DL_model_2	0.9930	1.0000	0.9859	0.9929	0.9990
3	Stacking	0.9790	0.9857	0.9718	0.9787	0.9990
4	Bagging	0.9790	1.0000	0.9577	0.9784	0.9989
5	DL_model_5	0.9720	0.9718	0.9718	0.9718	0.9980
6	AdaBoost	0.9580	1.0000	0.9155	0.9559	0.9980
7	DL_model_1	0.9580	0.9710	0.9437	0.9571	0.9978
8	LightGBM	0.9720	1.0000	0.9437	0.9710	0.9977
9	HistGradientBoosting	0.9720	1.0000	0.9437	0.9710	0.9977
10	XGBoost	0.9580	1.0000	0.9155	0.9559	0.9975
11	LogisticRegression	0.9860	0.9859	0.9859	0.9859	0.9973
12	SVM	0.9720	1.0000	0.9437	0.9710	0.9973
13	GradientBoosting	0.9510	0.9848	0.9155	0.9489	0.9973
14	DL_model_4	0.9790	1.0000	0.9577	0.9784	0.9971
15	NeuralNet_MLP	0.9860	1.0000	0.9718	0.9857	0.9969
16	DL_model_3	0.9371	0.9306	0.9437	0.9371	0.9920
17	KNN	0.9650	0.9714	0.9577	0.9645	0.9896
18	Voting_hard	0.9790	0.9857	0.9718	0.9787	0.9790
19	DecisionTree	0.9301	0.9420	0.9155	0.9286	0.9300



```
Best model based on ROC-AUC: ExtraTrees
```




```
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
```



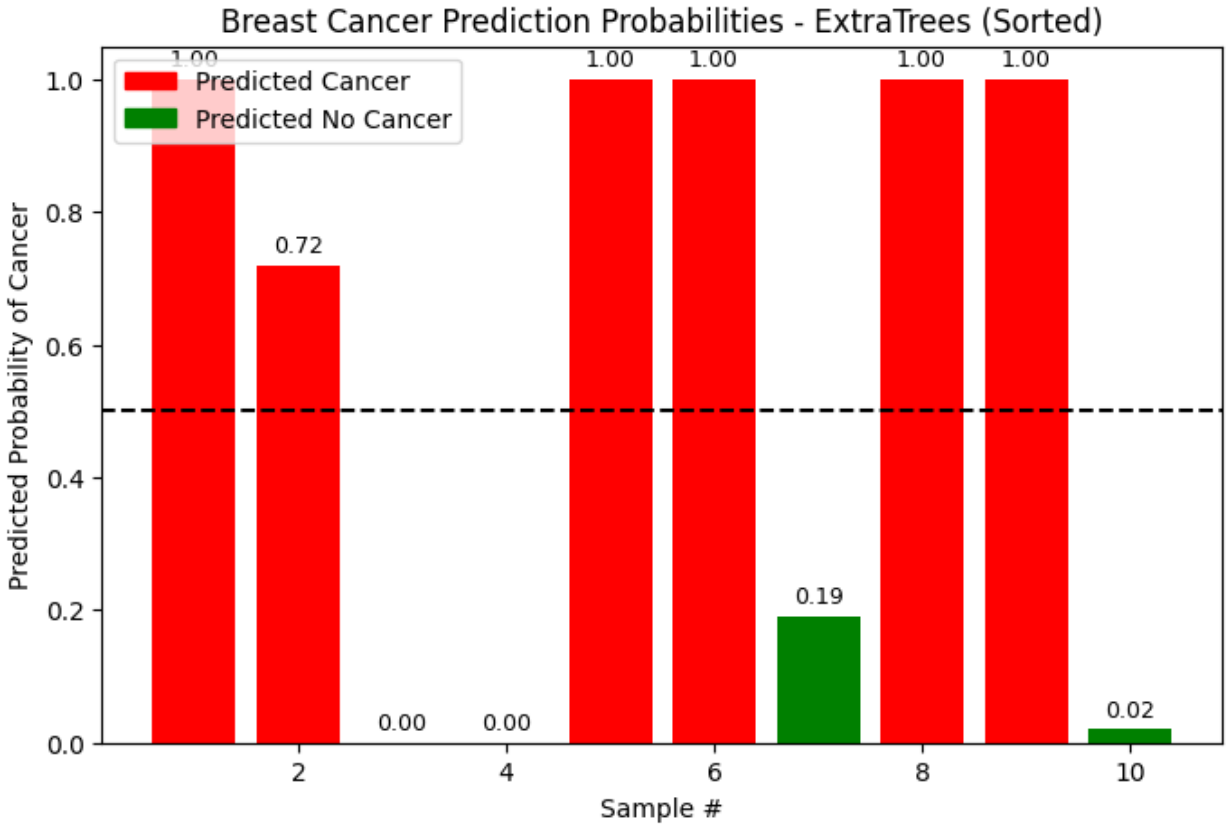

```
# %%
# 13. Prediction Demo - Breast Cancer Outcome

# Identify the best model (from Section 12 results)
best_model_name = df_results.iloc[0]['Model']
print(f"□ Using best model for prediction: {best_model_name}")

# Retrieve the trained model
if best_model_name in trained_ml:
    best_model = trained_ml[best_model_name]
    X_used = X_test
elif best_model_name in trained_ensembles:
    best_model = trained_ensembles[best_model_name]
    X_used = X_test
else:
    best_model = models_dl[best_model_name]
    X_used = X_test_np  # NumPy format for deep learning
```

```
# --- Pick a sample from test data ---
sample_index = 0  # Change to test different patients
sample_features = X_test.iloc[sample_index:sample_index+1] if not
"DL_model" in best_model_name else
X_test_np[sample_index:sample_index+1]

print("\n Features for one sample (scaled):")
print(sample_features)

# --- Make prediction ---
if "DL_model" in best_model_name:
    pred_proba = best_model.predict(sample_features).flatten()[0]
    pred_label = int(pred_proba > 0.5)
else:
    pred_proba = best_model.predict_proba(sample_features)[:,1][0]
    pred_label = best_model.predict(sample_features)[0]

# --- Map back to original class labels if applicable ---
if 'le_target' in locals():
    pred_class = le_target.inverse_transform([pred_label])[0]
else:
    pred_class = "Positive (Cancer)" if pred_label == 1 else "Negative
(No Cancer)"

print(f"\n Predicted probability of cancer: {pred_proba:.4f}")
print(f" Final Prediction: {pred_class}")
```

```
 Using best model for prediction: ExtraTrees

 Features for one sample (scaled):
           id  radius_mean  texture_mean  perimeter_mean  area_mean  \
631 -0.240488     1.917095     -0.614953        2.103541    1.95904

     smoothness_mean  compactness_mean  concavity_mean  concave
points_mean  \
631         2.250183          3.273287        3.722583
3.144977

     symmetry_mean  ...  radius_worst  texture_worst  perimeter_worst
\
631       2.554936  ...      2.308577      -0.186048         2.749677


     area_worst  smoothness_worst  compactness_worst  concavity_worst
\
631    2.405625          1.541303           2.763002         2.915069


     concave points_worst  symmetry_worst  fractal_dimension_worst
631              2.559036        2.154742                 1.151587
```

```
[1 rows x 31 columns]

 Predicted probability of cancer: 1.0000
 Final Prediction: M

# %%
#  Visualization: Breast Cancer Prediction Probabilities

# Predict probabilities and labels
y_proba_all = trained_ensembles['ExtraTrees'].predict_proba(X_test)[:,
1]
y_pred_all = (y_proba_all >= 0.5).astype(int)

# Create dataframe for visualization
df_pred_viz = pd.DataFrame({
    'Sample': range(1, 11),
    'Predicted_Probability': y_proba_all[:10],
    'Predicted_Label': y_pred_all[:10]
})

# Sort samples by probability
df_pred_viz = df_pred_viz.sort_values("Predicted_Probability",
ascending=False)

# Color map
colors = df_pred_viz["Predicted_Label"].map({1: 'red', 0: 'green'})

# Plot
plt.figure(figsize=(8,5))
bars = plt.bar(df_pred_viz["Sample"],
df_pred_viz["Predicted_Probability"], color=colors)

# Add threshold line
plt.axhline(0.5, color='black', linestyle='--', label='Threshold =
0.5')

# Add probability labels on top of bars
for i, val in enumerate(df_pred_viz["Predicted_Probability"]):
    plt.text(df_pred_viz["Sample"].iloc[i], val + 0.02, f"{val:.2f}",
ha='center', fontsize=9)

# Labels and legend
plt.title("Breast Cancer Prediction Probabilities - ExtraTrees
(Sorted)")
plt.ylabel("Predicted Probability of Cancer")
plt.xlabel("Sample #")
plt.legend(handles=[
    plt.Rectangle((0,0),1,1,color='red',label='Predicted Cancer'),
    plt.Rectangle((0,0),1,1,color='green',label='Predicted No Cancer')
```

```
])
plt.show()
```


Breast Cancer Prediction Probabilities - ExtraTrees (Sorted)
Predicted Cancer
Predicted No Cancer
Predicted Probability of Cancer
Sample #
1.00
0.72
0.00
0.00
1.00
1.00
0.19
1.00
1.00
0.02
0.0
0.2
0.4
0.6
0.8
1.0
2
4
6
8
10