Sornthun Intharasompong 6604051811054 MASDA KMUTNB

Base Model - SMOTE

```
import pandas as pd
import numpy as np
from sklearn.model_selection import StratifiedKFold
from sklearn.feature_selection import SequentialFeatureSelector
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from imblearn.over_sampling import SMOTE
import matplotlib.pyplot as plt

file_path = '/content/Data_Analysis_Jib.xlsx'
data = pd.read_excel(file_path)

features_to_drop = ['Net-income_LY']
data = data.drop(columns=features_to_drop, errors='ignore')

X = data.drop('Target', axis=1)
y = data['Target']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y)

smote = SMOTE(random_state=42)
X_train_balanced, y_train_balanced = smote.fit_resample(X_train, y_train)

base_models = {
    'Decision Tree': DecisionTreeClassifier(criterion='gini', max_depth=5, random_state=42),
    'Support Vector Machine': SVC(probability=True, random_state=42),
    'Gradient Boosting': GradientBoostingClassifier(random_state=42),
    'K-Nearest Neighbor': KNeighborsClassifier(),
    'Naïve Bayes': GaussianNB(),
}
results = {}

kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

for name, model in base_models.items():
    print(f"\nTraining model: {name}")
    test_metrics = {'Accuracy': [], 'Precision': [], 'Recall': [], 'F1': [], 'AUC': []}

    for fold, (train_idx, val_idx) in enumerate(kfold.split(X_train_balanced, y_train_balanced)):
        X_fold_train, X_fold_val = X_train_balanced.iloc[train_idx], X_train_balanced.iloc[val_idx]
        y_fold_train, y_fold_val = y_train_balanced.iloc[train_idx], y_train_balanced.iloc[val_idx]

        selector = SequentialFeatureSelector(LogisticRegression(max_iter=1000, solver='lbfgs'))
        selector.fit(X_fold_train, y_fold_train)
        selected_features = X_fold_train.columns[selector.get_support()]

        model.fit(X_fold_train[selected_features], y_fold_train)
        y_val_pred = model.predict(X_fold_val[selected_features])
        y_val_prob = model.predict_proba(X_fold_val[selected_features])[:, 1] if hasattr(model, "predict_proba") else None

        test_metrics['Accuracy'].append(accuracy_score(y_fold_val, y_val_pred))
        test_metrics['Precision'].append(precision_score(y_fold_val, y_val_pred, zero_division=0))
        test_metrics['Recall'].append(recall_score(y_fold_val, y_val_pred, zero_division=0))
        test_metrics['F1'].append(f1_score(y_fold_val, y_val_pred, zero_division=0))
        if y_val_prob is not None:
            test_metrics['AUC'].append(roc_auc_score(y_fold_val, y_val_prob))

    selector_full = SequentialFeatureSelector(LogisticRegression(max_iter=1000, solver='lbfgs'))
    selector_full.fit(X_train_balanced, y_train_balanced)
```

```
        final_features = X_train_balanced.columns[selector_full.get_support()]
        model.fit(X_train_balanced[final_features], y_train_balanced)
        y_test_pred = model.predict(X_test[final_features])
        y_test_prob = model.predict_proba(X_test[final_features])[:, 1] if hasattr(model, "predict_proba") else None

        results[name] = {
            'Final Accuracy': accuracy_score(y_test, y_test_pred),
            'Final Precision': precision_score(y_test, y_test_pred, zero_division=0),
            'Final Recall': recall_score(y_test, y_test_pred, zero_division=0),
            'Final F1 Score': f1_score(y_test, y_test_pred, zero_division=0),
            'Final AUC': roc_auc_score(y_test, y_test_prob) if y_test_prob is not None else None,
            'Avg CV Accuracy': np.mean(test_metrics['Accuracy']),
            'Avg CV F1 Score': np.mean(test_metrics['F1']),
            'Avg CV AUC': np.mean(test_metrics['AUC']) if test_metrics['AUC'] else None,
        }

    for model_name, metrics in results.items():
        print(f"\nModel: {model_name}")
        for metric, value in metrics.items():
            if value is not None:
                print(f"  {metric}: {value:.3f}")
```

Base Model - ADASYN

```
import pandas as pd
import numpy as np
from sklearn.model_selection import StratifiedKFold
from sklearn.feature_selection import SequentialFeatureSelector
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from imblearn.over_sampling import ADASYN
import matplotlib.pyplot as plt

file_path = '/content/Data_Analysis_Jib.xlsx'
data = pd.read_excel(file_path)

features_to_drop = ['Net-income_LY']
data = data.drop(columns=features_to_drop, errors='ignore')

X = data.drop('Target', axis=1)
y = data['Target']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y)

adasyn = ADASYN(random_state=42)
X_train_balanced, y_train_balanced = adasyn.fit_resample(X_train, y_train)

base_models = {
    'Decision Tree': DecisionTreeClassifier(criterion='gini', max_depth=5, random_state=42),
    'Support Vector Machine': SVC(probability=True, random_state=42),
    'Gradient Boosting': GradientBoostingClassifier(random_state=42),
    'K-Nearest Neighbor': KNeighborsClassifier(),
    'Naïve Bayes': GaussianNB(),
}

results = {}

kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

for name, model in base_models.items():
    print(f"\nTraining model: {name}")
    test_metrics = {'Accuracy': [], 'Precision': [], 'Recall': [], 'F1': [], 'AUC': []}

    for fold, (train_idx, val_idx) in enumerate(kfold.split(X_train_balanced, y_train_balanced)):
        X_fold_train, X_fold_val = X_train_balanced.iloc[train_idx], X_train_balanced.iloc[val_idx]
        y_fold_train, y_fold_val = y_train_balanced.iloc[train_idx], y_train_balanced.iloc[val_idx]

        selector = SequentialFeatureSelector(LogisticRegression(max_iter=1000, solver='lbfgs'))
        selector.fit(X_fold_train, y_fold_train)
        selected_features = X_fold_train.columns[selector.get_support()]

        model.fit(X_fold_train[selected_features], y_fold_train)
        y_val_pred = model.predict(X_fold_val[selected_features])
```

```
            y_val_pred = model.predict(X_fold_val[selected_features])
            y_val_prob = model.predict_proba(X_fold_val[selected_features])[:, 1] if hasattr(model, "predict_proba") else None

            test_metrics['Accuracy'].append(accuracy_score(y_fold_val, y_val_pred))
            test_metrics['Precision'].append(precision_score(y_fold_val, y_val_pred, zero_division=0))
            test_metrics['Recall'].append(recall_score(y_fold_val, y_val_pred, zero_division=0))
            test_metrics['F1'].append(f1_score(y_fold_val, y_val_pred, zero_division=0))
            if y_val_prob is not None:
                test_metrics['AUC'].append(roc_auc_score(y_fold_val, y_val_prob))

        selector_full = SequentialFeatureSelector(LogisticRegression(max_iter=1000, solver='lbfgs'))
        selector_full.fit(X_train_balanced, y_train_balanced)
        final_features = X_train_balanced.columns[selector_full.get_support()]
        model.fit(X_train_balanced[final_features], y_train_balanced)
        y_test_pred = model.predict(X_test[final_features])
        y_test_prob = model.predict_proba(X_test[final_features])[:, 1] if hasattr(model, "predict_proba") else None

        results[name] = {
            'Final Accuracy': accuracy_score(y_test, y_test_pred),
            'Final Precision': precision_score(y_test, y_test_pred, zero_division=0),
            'Final Recall': recall_score(y_test, y_test_pred, zero_division=0),
            'Final F1 Score': f1_score(y_test, y_test_pred, zero_division=0),
            'Final AUC': roc_auc_score(y_test, y_test_prob) if y_test_prob is not None else None,
            'Avg CV Accuracy': np.mean(test_metrics['Accuracy']),
            'Avg CV F1 Score': np.mean(test_metrics['F1']),
            'Avg CV AUC': np.mean(test_metrics['AUC']) if test_metrics['AUC'] else None,
        }

    for model_name, metrics in results.items():
        print(f"\nModel: {model_name}")
        for metric, value in metrics.items():
            if value is not None:
                print(f"  {metric}: {value:.3f}")
```

Base Model - SMOTEEN

```
import pandas as pd
import numpy as np
from sklearn.model_selection import StratifiedKFold
from sklearn.feature_selection import SequentialFeatureSelector
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from imblearn.combine import SMOTEENN
import matplotlib.pyplot as plt


file_path = '/content/Data_Analysis_Jib.xlsx'
data = pd.read_excel(file_path)


features_to_drop = ['Net-income_LY']
data = data.drop(columns=features_to_drop, errors='ignore')

X = data.drop('Target', axis=1)
y = data['Target']


from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y)

smoteenn = SMOTEENN(random_state=42)
X_train_balanced, y_train_balanced = smoteenn.fit_resample(X_train, y_train)

base_models = {
    'Decision Tree': DecisionTreeClassifier(criterion='gini', max_depth=5, random_state=42),
    'Support Vector Machine': SVC(probability=True, random_state=42),
    'Gradient Boosting': GradientBoostingClassifier(random_state=42),
    'K-Nearest Neighbor': KNeighborsClassifier(),
    'Naïve Bayes': GaussianNB(),
}

results = {}

kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
```

```
for name, model in base_models.items():
    print(f"\nTraining model: {name}")
    test_metrics = {'Accuracy': [], 'Precision': [], 'Recall': [], 'F1': [], 'AUC': []}

    for fold, (train_idx, val_idx) in enumerate(kfold.split(X_train_balanced, y_train_balanced)):
        X_fold_train, X_fold_val = X_train_balanced.iloc[train_idx], X_train_balanced.iloc[val_idx]
        y_fold_train, y_fold_val = y_train_balanced.iloc[train_idx], y_train_balanced.iloc[val_idx]

        selector = SequentialFeatureSelector(LogisticRegression(max_iter=1000, solver='lbfgs'))
        selector.fit(X_fold_train, y_fold_train)
        selected_features = X_fold_train.columns[selector.get_support()]

        model.fit(X_fold_train[selected_features], y_fold_train)
        y_val_pred = model.predict(X_fold_val[selected_features])
        y_val_prob = model.predict_proba(X_fold_val[selected_features])[:, 1] if hasattr(model, "predict_proba") else None

        test_metrics['Accuracy'].append(accuracy_score(y_fold_val, y_val_pred))
        test_metrics['Precision'].append(precision_score(y_fold_val, y_val_pred, zero_division=0))
        test_metrics['Recall'].append(recall_score(y_fold_val, y_val_pred, zero_division=0))
        test_metrics['F1'].append(f1_score(y_fold_val, y_val_pred, zero_division=0))
        if y_val_prob is not None:
            test_metrics['AUC'].append(roc_auc_score(y_fold_val, y_val_prob))

    selector_full = SequentialFeatureSelector(LogisticRegression(max_iter=1000, solver='lbfgs'))
    selector_full.fit(X_train_balanced, y_train_balanced)
    final_features = X_train_balanced.columns[selector_full.get_support()]
    model.fit(X_train_balanced[final_features], y_train_balanced)
    y_test_pred = model.predict(X_test[final_features])
    y_test_prob = model.predict_proba(X_test[final_features])[:, 1] if hasattr(model, "predict_proba") else None

    results[name] = {
        'Final Accuracy': accuracy_score(y_test, y_test_pred),
        'Final Precision': precision_score(y_test, y_test_pred, zero_division=0),
        'Final Recall': recall_score(y_test, y_test_pred, zero_division=0),
        'Final F1 Score': f1_score(y_test, y_test_pred, zero_division=0),
        'Final AUC': roc_auc_score(y_test, y_test_prob) if y_test_prob is not None else None,
        'Avg CV Accuracy': np.mean(test_metrics['Accuracy']),
        'Avg CV F1 Score': np.mean(test_metrics['F1']),
        'Avg CV AUC': np.mean(test_metrics['AUC']) if test_metrics['AUC'] else None,
    }

for model_name, metrics in results.items():
    print(f"\nModel: {model_name}")
    for metric, value in metrics.items():
        if value is not None:
            print(f"  {metric}: {value:.3f}")
```

Base Model - SMOTETomek

```
import pandas as pd
import numpy as np
from sklearn.model_selection import StratifiedKFold
from sklearn.feature_selection import SequentialFeatureSelector
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from imblearn.combine import SMOTETomek
import matplotlib.pyplot as plt

file_path = '/content/Data_Analysis_Jib.xlsx'
data = pd.read_excel(file_path)

features_to_drop = ['Net-income_LY']
data = data.drop(columns=features_to_drop, errors='ignore')

X = data.drop('Target', axis=1)
y = data['Target']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y)

smotetomek = SMOTETomek(random_state=42)
X_train_balanced, y_train_balanced = smotetomek.fit_resample(X_train, y_train)
```

```
base_models = {
    'Decision Tree': DecisionTreeClassifier(criterion='gini', max_depth=5, random_state=42),
    'Support Vector Machine': SVC(probability=True, random_state=42),
    'Gradient Boosting': GradientBoostingClassifier(random_state=42),
    'K-Nearest Neighbor': KNeighborsClassifier(),
    'Naïve Bayes': GaussianNB(),
}

results = {}

kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

for name, model in base_models.items():
    print(f"\nTraining model: {name}")
    test_metrics = {'Accuracy': [], 'Precision': [], 'Recall': [], 'F1': [], 'AUC': []}

    for fold, (train_idx, val_idx) in enumerate(kfold.split(X_train_balanced, y_train_balanced)):
        X_fold_train, X_fold_val = X_train_balanced.iloc[train_idx], X_train_balanced.iloc[val_idx]
        y_fold_train, y_fold_val = y_train_balanced.iloc[train_idx], y_train_balanced.iloc[val_idx]

        selector = SequentialFeatureSelector(LogisticRegression(max_iter=1000, solver='lbfgs'))
        selector.fit(X_fold_train, y_fold_train)
        selected_features = X_fold_train.columns[selector.get_support()]

        model.fit(X_fold_train[selected_features], y_fold_train)
        y_val_pred = model.predict(X_fold_val[selected_features])
        y_val_prob = model.predict_proba(X_fold_val[selected_features])[:, 1] if hasattr(model, "predict_proba") else None

        test_metrics['Accuracy'].append(accuracy_score(y_fold_val, y_val_pred))
        test_metrics['Precision'].append(precision_score(y_fold_val, y_val_pred, zero_division=0))
        test_metrics['Recall'].append(recall_score(y_fold_val, y_val_pred, zero_division=0))
        test_metrics['F1'].append(f1_score(y_fold_val, y_val_pred, zero_division=0))
        if y_val_prob is not None:
            test_metrics['AUC'].append(roc_auc_score(y_fold_val, y_val_prob))

    selector_full = SequentialFeatureSelector(LogisticRegression(max_iter=1000, solver='lbfgs'))
    selector_full.fit(X_train_balanced, y_train_balanced)
    final_features = X_train_balanced.columns[selector_full.get_support()]
    model.fit(X_train_balanced[final_features], y_train_balanced)
    y_test_pred = model.predict(X_test[final_features])
    y_test_prob = model.predict_proba(X_test[final_features])[:, 1] if hasattr(model, "predict_proba") else None

    results[name] = {
        'Final Accuracy': accuracy_score(y_test, y_test_pred),
        'Final Precision': precision_score(y_test, y_test_pred, zero_division=0),
        'Final Recall': recall_score(y_test, y_test_pred, zero_division=0),
        'Final F1 Score': f1_score(y_test, y_test_pred, zero_division=0),
        'Final AUC': roc_auc_score(y_test, y_test_prob) if y_test_prob is not None else None,
        'Avg CV Accuracy': np.mean(test_metrics['Accuracy']),
        'Avg CV F1 Score': np.mean(test_metrics['F1']),
        'Avg CV AUC': np.mean(test_metrics['AUC']) if test_metrics['AUC'] else None,
    }

for model_name, metrics in results.items():
    print(f"\nModel: {model_name}")
    for metric, value in metrics.items():
        if value is not None:
            print(f"  {metric}: {value:.3f}")
```

META Model - LR by SMOTE

```
import pandas as pd
import numpy as np
from sklearn.model_selection import StratifiedKFold, train_test_split
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
from sklearn.feature_selection import SequentialFeatureSelector
from imblearn.over_sampling import SMOTE
import matplotlib.pyplot as plt


file_path = '/content/Data_Analysis_Jib.xlsx'
data = pd.read_excel(file_path)
```

```
X = data.drop('Target', axis=1)
y = data['Target']


smote = SMOTE(random_state=42)
X_balanced, y_balanced = smote.fit_resample(X, y)


X_train, X_test, y_train, y_test = train_test_split(
    X_balanced, y_balanced, test_size=0.2, random_state=42, stratify=y_balanced
)

base_models = {
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "SVM": SVC(probability=True, random_state=42),
    "Gradient Boosting": GradientBoostingClassifier(random_state=42),
    "K-Nearest Neighbors": KNeighborsClassifier(),
    "Naive Bayes": GaussianNB(),
}

meta_model = LogisticRegression(max_iter=1000, random_state=42)

kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
meta_cv_scores = []
base_model_results = {name: {'accuracy': [], 'f1': [], 'auc': []} for name in base_models}

print("\n=== Cross-Validation (Base Models + Stacking Meta-Model) ===")

for fold, (train_idx, val_idx) in enumerate(kfold.split(X_train, y_train), 1):
    X_fold_train, X_fold_val = X_train.iloc[train_idx], X_train.iloc[val_idx]
    y_fold_train, y_fold_val = y_train.iloc[train_idx], y_train.iloc[val_idx]

    meta_features = np.zeros((X_fold_val.shape[0], len(base_models)))

    for i, (model_name, model) in enumerate(base_models.items()):

        selector = SequentialFeatureSelector(LogisticRegression(max_iter=1000), direction='forward')
        selector.fit(X_fold_train, y_fold_train)
        selected_features = X_fold_train.columns[selector.get_support()]

        model.fit(X_fold_train[selected_features], y_fold_train)

        if hasattr(model, "predict_proba"):
            prob = model.predict_proba(X_fold_val[selected_features])[:, 1]
        else:
            prob = model.decision_function(X_fold_val[selected_features])
            prob = (prob - prob.min()) / (prob.max() - prob.min())

        pred = model.predict(X_fold_val[selected_features])

        base_model_results[model_name]['accuracy'].append(accuracy_score(y_fold_val, pred))
        base_model_results[model_name]['f1'].append(f1_score(y_fold_val, pred))
        base_model_results[model_name]['auc'].append(roc_auc_score(y_fold_val, prob))

        meta_features[:, i] = prob

    meta_model.fit(meta_features, y_fold_val)
    y_pred_meta = meta_model.predict(meta_features)
    meta_cv_scores.append(accuracy_score(y_fold_val, y_pred_meta))
    print(f"  Fold {fold}: Meta Accuracy = {meta_cv_scores[-1]:.4f}")

print("\n=== Average CV Performance of Base Models ===")
for name, scores in base_model_results.items():
    print(f"  {name}")
    print(f"    Avg Accuracy: {np.mean(scores['accuracy']):.4f}")
    print(f"    Avg F1 Score: {np.mean(scores['f1']):.4f}")
    print(f"    Avg AUC: {np.mean(scores['auc']):.4f}")

print(f"\nMeta-Model Mean CV Accuracy: {np.mean(meta_cv_scores):.4f}")

meta_features_test = np.zeros((X_test.shape[0], len(base_models)))

for i, (model_name, model) in enumerate(base_models.items()):
    selector = SequentialFeatureSelector(LogisticRegression(max_iter=1000), direction='forward')
    selector.fit(X_train, y_train)
    selected_features = X_train.columns[selector.get_support()]

    model.fit(X_train[selected_features], y_train)

    if hasattr(model, "predict_proba"):
        meta_features_test[:, i] = model.predict_proba(X_test[selected_features])[:, 1]
    else:
```

```
        prob = model.decision_function(X_test[selected_features])
        meta_features_test[:, i] = (prob - prob.min()) / (prob.max() - prob.min())

meta_model.fit(meta_features, y_fold_val)  # Fit with latest fold for demo (optionally retrain on all data)
y_pred_test = meta_model.predict(meta_features_test)
y_pred_prob_test = meta_model.predict_proba(meta_features_test)[:, 1]

print("\n=== Meta-Model Test Performance ===")
print(f"  Accuracy: {accuracy_score(y_test, y_pred_test):.4f}")
print(f"  Precision: {precision_score(y_test, y_pred_test):.4f}")
print(f"  Recall: {recall_score(y_test, y_pred_test):.4f}")
print(f"  F1 Score: {f1_score(y_test, y_pred_test):.4f}")
print(f"  AUC: {roc_auc_score(y_test, y_pred_prob_test):.4f}")

from sklearn.metrics import roc_curve, auc
fpr, tpr, _ = roc_curve(y_test, y_pred_prob_test)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate (1 - Specificity)')
plt.ylabel('True Positive Rate (Sensitivity)')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```

META Model - LR by ADASYN

```
import pandas as pd
import numpy as np
from sklearn.model_selection import StratifiedKFold, train_test_split
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
from sklearn.feature_selection import SequentialFeatureSelector
from imblearn.over_sampling import ADASYN
import matplotlib.pyplot as plt

file_path = '/content/Data_Analysis_Jib.xlsx'
data = pd.read_excel(file_path)

X = data.drop('Target', axis=1)
y = data['Target']

adasyn = ADASYN(random_state=42)
X_balanced, y_balanced = adasyn.fit_resample(X, y)

X_train, X_test, y_train, y_test = train_test_split(
    X_balanced, y_balanced, test_size=0.2, random_state=42, stratify=y_balanced
)

base_models = {
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "SVM": SVC(probability=True, random_state=42),
    "Gradient Boosting": GradientBoostingClassifier(random_state=42),
    "K-Nearest Neighbors": KNeighborsClassifier(),
    "Naive Bayes": GaussianNB(),
}

meta_model = LogisticRegression(max_iter=1000, random_state=42)

kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
meta_cv_scores = []
base_model_results = {name: {'accuracy': [], 'f1': [], 'auc': []} for name in base_models}

print("\n=== Cross-Validation (Base Models + Stacking Meta-Model) ===")

for fold, (train_idx, val_idx) in enumerate(kfold.split(X_train, y_train), 1):
    X_fold_train, X_fold_val = X_train.iloc[train_idx], X_train.iloc[val_idx]
    y_fold_train, y_fold_val = y_train.iloc[train_idx], y_train.iloc[val_idx]

    meta_features = np.zeros((X_fold_val.shape[0], len(base_models)))

    for i, (model_name, model) in enumerate(base_models.items()):

        selector = SequentialFeatureSelector(LogisticRegression(max_iter=1000), direction='forward')
        selector.fit(X_fold_train, y_fold_train)
        selected_features = X_fold_train.columns[selector.get_support()]

        model.fit(X_fold_train[selected_features], y_fold_train)

        if hasattr(model, "predict_proba"):
            prob = model.predict_proba(X_fold_val[selected_features])[:, 1]
        else:
            prob = model.decision_function(X_fold_val[selected_features])
            prob = (prob - prob.min()) / (prob.max() - prob.min())

        pred = model.predict(X_fold_val[selected_features])

        base_model_results[model_name]['accuracy'].append(accuracy_score(y_fold_val, pred))
        base_model_results[model_name]['f1'].append(f1_score(y_fold_val, pred))
        base_model_results[model_name]['auc'].append(roc_auc_score(y_fold_val, prob))

        meta_features[:, i] = prob

    meta_model.fit(meta_features, y_fold_val)
    y_pred_meta = meta_model.predict(meta_features)
    meta_cv_scores.append(accuracy_score(y_fold_val, y_pred_meta))
    print(f"  Fold {fold}: Meta Accuracy = {meta_cv_scores[-1]:.4f}")

print("\n=== Average CV Performance of Base Models ===")
for name, scores in base_model_results.items():
    print(f"  {name}")
    print(f"    Avg Accuracy: {np.mean(scores['accuracy']):.4f}")
    print(f"    Avg F1 Score: {np.mean(scores['f1']):.4f}")
    print(f"    Avg AUC: {np.mean(scores['auc']):.4f}")
```

```
print(f"\nMeta-Model Mean CV Accuracy: {np.mean(meta_cv_scores):.4f}")

meta_features_test = np.zeros((X_test.shape[0], len(base_models)))

for i, (model_name, model) in enumerate(base_models.items()):
    selector = SequentialFeatureSelector(LogisticRegression(max_iter=1000), direction='forward')
    selector.fit(X_train, y_train)
    selected_features = X_train.columns[selector.get_support()]

    model.fit(X_train[selected_features], y_train)

    if hasattr(model, "predict_proba"):
        meta_features_test[:, i] = model.predict_proba(X_test[selected_features])[:, 1]
    else:
        prob = model.decision_function(X_test[selected_features])
        meta_features_test[:, i] = (prob - prob.min()) / (prob.max() - prob.min())

meta_model.fit(meta_features, y_fold_val)
y_pred_test = meta_model.predict(meta_features_test)
y_pred_prob_test = meta_model.predict_proba(meta_features_test)[:, 1]

print("\n=== Meta-Model Test Performance ===")
print(f"  Accuracy: {accuracy_score(y_test, y_pred_test):.4f}")
print(f"  Precision: {precision_score(y_test, y_pred_test):.4f}")
print(f"  Recall: {recall_score(y_test, y_pred_test):.4f}")
print(f"  F1 Score: {f1_score(y_test, y_pred_test):.4f}")
print(f"  AUC: {roc_auc_score(y_test, y_pred_prob_test):.4f}")

from sklearn.metrics import roc_curve, auc
fpr, tpr, _ = roc_curve(y_test, y_pred_prob_test)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate (1 - Specificity)')
plt.ylabel('True Positive Rate (Sensitivity)')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```

META Model - LR by SMOTEEN

```
import pandas as pd
import numpy as np
from sklearn.model_selection import StratifiedKFold, train_test_split
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
from sklearn.feature_selection import SequentialFeatureSelector
from imblearn.combine import SMOTEENN
from imblearn.combine import SMOTETomek
import matplotlib.pyplot as plt

file_path = '/content/Data_Analysis_Jib.xlsx'
data = pd.read_excel(file_path)

X = data.drop('Target', axis=1)
y = data['Target']

smoteenn = SMOTEENN(random_state=42)
X_balanced, y_balanced = smoteenn.fit_resample(X, y)

X_train, X_test, y_train, y_test = train_test_split(
    X_balanced, y_balanced, test_size=0.2, random_state=42, stratify=y_balanced
)

base_models = {
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "SVM": SVC(probability=True, random_state=42),
    "Gradient Boosting": GradientBoostingClassifier(random_state=42),
    "K-Nearest Neighbors": KNeighborsClassifier(),
    "Naive Bayes": GaussianNB(),
}

meta_model = LogisticRegression(max_iter=1000, random_state=42)

kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
meta_cv_scores = []
base_model_results = {name: {'accuracy': [], 'f1': [], 'auc': []} for name in base_models}

print("\n=== Cross-Validation (Base Models + Stacking Meta-Model) ===")

for fold, (train_idx, val_idx) in enumerate(kfold.split(X_train, y_train), 1):
    X_fold_train, X_fold_val = X_train.iloc[train_idx], X_train.iloc[val_idx]
    y_fold_train, y_fold_val = y_train.iloc[train_idx], y_train.iloc[val_idx]

    meta_features = np.zeros((X_fold_val.shape[0], len(base_models)))

    for i, (model_name, model) in enumerate(base_models.items()):

        selector = SequentialFeatureSelector(LogisticRegression(max_iter=1000), direction='forward')
        selector.fit(X_fold_train, y_fold_train)
        selected_features = X_fold_train.columns[selector.get_support()]

        model.fit(X_fold_train[selected_features], y_fold_train)

        if hasattr(model, "predict_proba"):
            prob = model.predict_proba(X_fold_val[selected_features])[:, 1]
        else:
            prob = model.decision_function(X_fold_val[selected_features])
            prob = (prob - prob.min()) / (prob.max() - prob.min())

        pred = model.predict(X_fold_val[selected_features])

        base_model_results[model_name]['accuracy'].append(accuracy_score(y_fold_val, pred))
        base_model_results[model_name]['f1'].append(f1_score(y_fold_val, pred))
        base_model_results[model_name]['auc'].append(roc_auc_score(y_fold_val, prob))

        meta_features[:, i] = prob

    meta_model.fit(meta_features, y_fold_val)
    y_pred_meta = meta_model.predict(meta_features)
    meta_cv_scores.append(accuracy_score(y_fold_val, y_pred_meta))
    print(f"  Fold {fold}: Meta Accuracy = {meta_cv_scores[-1]:.4f}")

print("\n=== Average CV Performance of Base Models ===")
for name, scores in base_model_results.items():
    print(f"  {name}")
    print(f"    Avg Accuracy: {np.mean(scores['accuracy']):.4f}")
    print(f"    Avg F1 Score: {np.mean(scores['f1']):.4f}")
```

```
        print(f"    Avg AUC: {np.mean(scores['auc']):.4f}")

    print(f"\nMeta-Model Mean CV Accuracy: {np.mean(meta_cv_scores):.4f}")

    meta_features_test = np.zeros((X_test.shape[0], len(base_models)))

    for i, (model_name, model) in enumerate(base_models.items()):
        selector = SequentialFeatureSelector(LogisticRegression(max_iter=1000), direction='forward')
        selector.fit(X_train, y_train)
        selected_features = X_train.columns[selector.get_support()]

        model.fit(X_train[selected_features], y_train)

        if hasattr(model, "predict_proba"):
            meta_features_test[:, i] = model.predict_proba(X_test[selected_features])[:, 1]
        else:
            prob = model.decision_function(X_test[selected_features])
            meta_features_test[:, i] = (prob - prob.min()) / (prob.max() - prob.min())

    meta_model.fit(meta_features, y_fold_val)
    y_pred_test = meta_model.predict(meta_features_test)
    y_pred_prob_test = meta_model.predict_proba(meta_features_test)[:, 1]

    print("\n=== Meta-Model Test Performance ===")
    print(f"  Accuracy: {accuracy_score(y_test, y_pred_test):.4f}")
    print(f"  Precision: {precision_score(y_test, y_pred_test):.4f}")
    print(f"  Recall: {recall_score(y_test, y_pred_test):.4f}")
    print(f"  F1 Score: {f1_score(y_test, y_pred_test):.4f}")
    print(f"  AUC: {roc_auc_score(y_test, y_pred_prob_test):.4f}")

    from sklearn.metrics import roc_curve, auc
    fpr, tpr, _ = roc_curve(y_test, y_pred_prob_test)
    roc_auc = auc(fpr, tpr)

    plt.figure(figsize=(8, 6))
    plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
    plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate (1 - Specificity)')
    plt.ylabel('True Positive Rate (Sensitivity)')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend(loc='lower right')
    plt.show()
```

META Model - LR by SMOTTomek

```
import pandas as pd
import numpy as np
from sklearn.model_selection import StratifiedKFold, train_test_split
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
from sklearn.feature_selection import SequentialFeatureSelector
from imblearn.combine import SMOTEENN
from imblearn.combine import SMOTETomek
import matplotlib.pyplot as plt

file_path = '/content/Data_Analysis_Jib.xlsx'
data = pd.read_excel(file_path)

X = data.drop('Target', axis=1)
y = data['Target']

smotetomek = SMOTETomek(random_state=42)
X_balanced, y_balanced = smotetomek.fit_resample(X, y)

X_train, X_test, y_train, y_test = train_test_split(
    X_balanced, y_balanced, test_size=0.2, random_state=42, stratify=y_balanced
)

base_models = {
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "SVM": SVC(probability=True, random_state=42),
    "Gradient Boosting": GradientBoostingClassifier(random_state=42),
    "K-Nearest Neighbors": KNeighborsClassifier(),
    "Naive Bayes": GaussianNB(),
}

meta_model = LogisticRegression(max_iter=1000, random_state=42)

kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
meta_cv_scores = []
base_model_results = {name: {'accuracy': [], 'f1': [], 'auc': []} for name in base_models}

print("\n=== Cross-Validation (Base Models + Stacking Meta-Model) ===")

for fold, (train_idx, val_idx) in enumerate(kfold.split(X_train, y_train), 1):
    X_fold_train, X_fold_val = X_train.iloc[train_idx], X_train.iloc[val_idx]
    y_fold_train, y_fold_val = y_train.iloc[train_idx], y_train.iloc[val_idx]

    meta_features = np.zeros((X_fold_val.shape[0], len(base_models)))

    for i, (model_name, model) in enumerate(base_models.items()):

        selector = SequentialFeatureSelector(LogisticRegression(max_iter=1000), direction='forward')
        selector.fit(X_fold_train, y_fold_train)
        selected_features = X_fold_train.columns[selector.get_support()]

        model.fit(X_fold_train[selected_features], y_fold_train)

        if hasattr(model, "predict_proba"):
            prob = model.predict_proba(X_fold_val[selected_features])[:, 1]
        else:
            prob = model.decision_function(X_fold_val[selected_features])
            prob = (prob - prob.min()) / (prob.max() - prob.min())

        pred = model.predict(X_fold_val[selected_features])

        base_model_results[model_name]['accuracy'].append(accuracy_score(y_fold_val, pred))
        base_model_results[model_name]['f1'].append(f1_score(y_fold_val, pred))
        base_model_results[model_name]['auc'].append(roc_auc_score(y_fold_val, prob))

        meta_features[:, i] = prob

    meta_model.fit(meta_features, y_fold_val)
    y_pred_meta = meta_model.predict(meta_features)
    meta_cv_scores.append(accuracy_score(y_fold_val, y_pred_meta))
    print(f"  Fold {fold}: Meta Accuracy = {meta_cv_scores[-1]:.4f}")

print("\n=== Average CV Performance of Base Models ===")
for name, scores in base_model_results.items():
    print(f"  {name}")
    print(f"    Avg Accuracy: {np.mean(scores['accuracy']):.4f}")
    print(f"    Avg F1 Score: {np.mean(scores['f1']):.4f}")
```

```
        print(f"    Avg AUC: {np.mean(scores['auc']):.4f}")

    print(f"\nMeta-Model Mean CV Accuracy: {np.mean(meta_cv_scores):.4f}")

    meta_features_test = np.zeros((X_test.shape[0], len(base_models)))

    for i, (model_name, model) in enumerate(base_models.items()):
        selector = SequentialFeatureSelector(LogisticRegression(max_iter=1000), direction='forward')
        selector.fit(X_train, y_train)
        selected_features = X_train.columns[selector.get_support()]

        model.fit(X_train[selected_features], y_train)

        if hasattr(model, "predict_proba"):
            meta_features_test[:, i] = model.predict_proba(X_test[selected_features])[:, 1]
        else:
            prob = model.decision_function(X_test[selected_features])
            meta_features_test[:, i] = (prob - prob.min()) / (prob.max() - prob.min())

    meta_model.fit(meta_features, y_fold_val)
    y_pred_test = meta_model.predict(meta_features_test)
    y_pred_prob_test = meta_model.predict_proba(meta_features_test)[:, 1]

    print("\n=== Meta-Model Test Performance ===")
    print(f"  Accuracy: {accuracy_score(y_test, y_pred_test):.4f}")
    print(f"  Precision: {precision_score(y_test, y_pred_test):.4f}")
    print(f"  Recall: {recall_score(y_test, y_pred_test):.4f}")
    print(f"  F1 Score: {f1_score(y_test, y_pred_test):.4f}")
    print(f"  AUC: {roc_auc_score(y_test, y_pred_prob_test):.4f}")

    from sklearn.metrics import roc_curve, auc
    fpr, tpr, _ = roc_curve(y_test, y_pred_prob_test)
    roc_auc = auc(fpr, tpr)

    plt.figure(figsize=(8, 6))
    plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
    plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate (1 - Specificity)')
    plt.ylabel('True Positive Rate (Sensitivity)')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend(loc='lower right')
    plt.show()
```

META Model - GB by SMOTE

```
import pandas as pd
import numpy as np
from sklearn.model_selection import StratifiedKFold, train_test_split
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
from sklearn.feature_selection import SequentialFeatureSelector
from imblearn.over_sampling import SMOTE
import matplotlib.pyplot as plt

file_path = '/content/Data_Analysis_Jib.xlsx'
data = pd.read_excel(file_path)

X = data.drop('Target', axis=1)
y = data['Target']

smote = SMOTE(random_state=42)
X_balanced, y_balanced = smote.fit_resample(X, y)

X_train, X_test, y_train, y_test = train_test_split(
    X_balanced, y_balanced, test_size=0.2, random_state=42, stratify=y_balanced
)

base_models = {
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "SVM": SVC(probability=True, random_state=42),
    "Gradient Boosting": GradientBoostingClassifier(random_state=42),
    "K-Nearest Neighbors": KNeighborsClassifier(),
```

```
        "Naive Bayes": GaussianNB(),
    }

    meta_model = GradientBoostingClassifier(random_state=42)

    kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
    meta_cv_scores = []
    base_model_results = {name: {'accuracy': [], 'f1': [], 'auc': []} for name in base_models}

    print("\n=== Cross-Validation (Base Models + Stacking Meta-Model) ===")

    for fold, (train_idx, val_idx) in enumerate(kfold.split(X_train, y_train), 1):
        X_fold_train, X_fold_val = X_train.iloc[train_idx], X_train.iloc[val_idx]
        y_fold_train, y_fold_val = y_train.iloc[train_idx], y_train.iloc[val_idx]

        meta_features = np.zeros((X_fold_val.shape[0], len(base_models)))

        for i, (model_name, model) in enumerate(base_models.items()):
            # Feature selection
            selector = SequentialFeatureSelector(LogisticRegression(max_iter=1000), direction='forward')
            selector.fit(X_fold_train, y_fold_train)
            selected_features = X_fold_train.columns[selector.get_support()]

            model.fit(X_fold_train[selected_features], y_fold_train)

            if hasattr(model, "predict_proba"):
                prob = model.predict_proba(X_fold_val[selected_features])[:, 1]
            else:
                prob = model.decision_function(X_fold_val[selected_features])
                prob = (prob - prob.min()) / (prob.max() - prob.min())  # scale to [0,1]

            pred = model.predict(X_fold_val[selected_features])

            base_model_results[model_name]['accuracy'].append(accuracy_score(y_fold_val, pred))
            base_model_results[model_name]['f1'].append(f1_score(y_fold_val, pred))
            base_model_results[model_name]['auc'].append(roc_auc_score(y_fold_val, prob))

            meta_features[:, i] = prob

        meta_model.fit(meta_features, y_fold_val)
        y_pred_meta = meta_model.predict(meta_features)
        meta_cv_scores.append(accuracy_score(y_fold_val, y_pred_meta))
        print(f"  Fold {fold}: Meta Accuracy = {meta_cv_scores[-1]:.4f}")

    print("\n=== Average CV Performance of Base Models ===")
    for name, scores in base_model_results.items():
        print(f"  {name}")
        print(f"    Avg Accuracy: {np.mean(scores['accuracy']):.4f}")
        print(f"    Avg F1 Score: {np.mean(scores['f1']):.4f}")
        print(f"    Avg AUC: {np.mean(scores['auc']):.4f}")

    print(f"\nMeta-Model Mean CV Accuracy: {np.mean(meta_cv_scores):.4f}")

    meta_features_test = np.zeros((X_test.shape[0], len(base_models)))

    for i, (model_name, model) in enumerate(base_models.items()):
        selector = SequentialFeatureSelector(LogisticRegression(max_iter=1000), direction='forward')
        selector.fit(X_train, y_train)
        selected_features = X_train.columns[selector.get_support()]

        model.fit(X_train[selected_features], y_train)

        if hasattr(model, "predict_proba"):
            meta_features_test[:, i] = model.predict_proba(X_test[selected_features])[:, 1]
        else:
            prob = model.decision_function(X_test[selected_features])
            meta_features_test[:, i] = (prob - prob.min()) / (prob.max() - prob.min())

    meta_model.fit(meta_features, y_fold_val)  # Fit with latest fold for demo (optionally retrain on all data)
    y_pred_test = meta_model.predict(meta_features_test)
    y_pred_prob_test = meta_model.predict_proba(meta_features_test)[:, 1]

    print("\n=== Meta-Model Test Performance ===")
    print(f"  Accuracy: {accuracy_score(y_test, y_pred_test):.4f}")
    print(f"  Precision: {precision_score(y_test, y_pred_test):.4f}")
    print(f"  Recall: {recall_score(y_test, y_pred_test):.4f}")
    print(f"  F1 Score: {f1_score(y_test, y_pred_test):.4f}")
    print(f"  AUC: {roc_auc_score(y_test, y_pred_prob_test):.4f}")

    from sklearn.metrics import roc_curve, auc
    fpr, tpr, _ = roc_curve(y_test, y_pred_prob_test)
    roc_auc = auc(fpr, tpr)
```

```
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate (1 - Specificity)')
plt.ylabel('True Positive Rate (Sensitivity)')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```

META Model - GB by ADASYN

```
import pandas as pd
import numpy as np
from sklearn.model_selection import StratifiedKFold, train_test_split
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
from sklearn.feature_selection import SequentialFeatureSelector
from imblearn.over_sampling import ADASYN
import matplotlib.pyplot as plt

file_path = '/content/Data_Analysis_Jib.xlsx'
data = pd.read_excel(file_path)

X = data.drop('Target', axis=1)
y = data['Target']

adasyn = ADASYN(random_state=42)
X_balanced, y_balanced = adasyn.fit_resample(X, y)

X_train, X_test, y_train, y_test = train_test_split(
    X_balanced, y_balanced, test_size=0.2, random_state=42, stratify=y_balanced
)

base_models = {
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "SVM": SVC(probability=True, random_state=42),
    "Gradient Boosting": GradientBoostingClassifier(random_state=42),
    "K-Nearest Neighbors": KNeighborsClassifier(),
    "Naive Bayes": GaussianNB(),
}

meta_model = GradientBoostingClassifier(random_state=42)

kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
meta_cv_scores = []
base_model_results = {name: {'accuracy': [], 'f1': [], 'auc': []} for name in base_models}

print("\n=== Cross-Validation (Base Models + Stacking Meta-Model) ===")

for fold, (train_idx, val_idx) in enumerate(kfold.split(X_train, y_train), 1):
    X_fold_train, X_fold_val = X_train.iloc[train_idx], X_train.iloc[val_idx]
    y_fold_train, y_fold_val = y_train.iloc[train_idx], y_train.iloc[val_idx]

    meta_features = np.zeros((X_fold_val.shape[0], len(base_models)))

    for i, (model_name, model) in enumerate(base_models.items()):

        selector = SequentialFeatureSelector(LogisticRegression(max_iter=1000), direction='forward')
        selector.fit(X_fold_train, y_fold_train)
        selected_features = X_fold_train.columns[selector.get_support()]

        model.fit(X_fold_train[selected_features], y_fold_train)

        if hasattr(model, "predict_proba"):
            prob = model.predict_proba(X_fold_val[selected_features])[:, 1]
        else:
            prob = model.decision_function(X_fold_val[selected_features])
            prob = (prob - prob.min()) / (prob.max() - prob.min())

        pred = model.predict(X_fold_val[selected_features])

        base_model_results[model_name]['accuracy'].append(accuracy_score(y_fold_val, pred))
        base_model_results[model_name]['f1'].append(f1_score(y_fold_val, pred))
        base_model_results[model_name]['auc'].append(roc_auc_score(y_fold_val, prob))

        meta_features[:, i] = prob

    meta_model.fit(meta_features, y_fold_val)
    y_pred_meta = meta_model.predict(meta_features)
    meta_cv_scores.append(accuracy_score(y_fold_val, y_pred_meta))
    print(f"  Fold {fold}: Meta Accuracy = {meta_cv_scores[-1]:.4f}")

print("\n=== Average CV Performance of Base Models ===")
for name, scores in base_model_results.items():
    print(f"  {name}")
    print(f"    Avg Accuracy: {np.mean(scores['accuracy']):.4f}")
    print(f"    Avg F1 Score: {np.mean(scores['f1']):.4f}")
    print(f"    Avg AUC: {np.mean(scores['auc']):.4f}")
```

```
print(f"\nMeta-Model Mean CV Accuracy: {np.mean(meta_cv_scores):.4f}")

meta_features_test = np.zeros((X_test.shape[0], len(base_models)))

for i, (model_name, model) in enumerate(base_models.items()):
    selector = SequentialFeatureSelector(LogisticRegression(max_iter=1000), direction='forward')
    selector.fit(X_train, y_train)
    selected_features = X_train.columns[selector.get_support()]

    model.fit(X_train[selected_features], y_train)

    if hasattr(model, "predict_proba"):
        meta_features_test[:, i] = model.predict_proba(X_test[selected_features])[:, 1]
    else:
        prob = model.decision_function(X_test[selected_features])
        meta_features_test[:, i] = (prob - prob.min()) / (prob.max() - prob.min())

meta_model.fit(meta_features, y_fold_val)
y_pred_test = meta_model.predict(meta_features_test)
y_pred_prob_test = meta_model.predict_proba(meta_features_test)[:, 1]

print("\n=== Meta-Model Test Performance ===")
print(f"  Accuracy: {accuracy_score(y_test, y_pred_test):.4f}")
print(f"  Precision: {precision_score(y_test, y_pred_test):.4f}")
print(f"  Recall: {recall_score(y_test, y_pred_test):.4f}")
print(f"  F1 Score: {f1_score(y_test, y_pred_test):.4f}")
print(f"  AUC: {roc_auc_score(y_test, y_pred_prob_test):.4f}")

from sklearn.metrics import roc_curve, auc
fpr, tpr, _ = roc_curve(y_test, y_pred_prob_test)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate (1 - Specificity)')
plt.ylabel('True Positive Rate (Sensitivity)')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```

META Model - GB by SMOTEEN

```
import pandas as pd
import numpy as np
from sklearn.model_selection import StratifiedKFold, train_test_split
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
from sklearn.feature_selection import SequentialFeatureSelector
from imblearn.combine import SMOTEENN
from imblearn.combine import SMOTETomek
import matplotlib.pyplot as plt

file_path = '/content/Data_Analysis_Jib.xlsx'
data = pd.read_excel(file_path)

X = data.drop('Target', axis=1)
y = data['Target']

smoteenn = SMOTEENN(random_state=42)
X_balanced, y_balanced = smoteenn.fit_resample(X, y)

X_train, X_test, y_train, y_test = train_test_split(
    X_balanced, y_balanced, test_size=0.2, random_state=42, stratify=y_balanced
)

base_models = {
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "SVM": SVC(probability=True, random_state=42),
    "Gradient Boosting": GradientBoostingClassifier(random_state=42),
    "K-Nearest Neighbors": KNeighborsClassifier(),
```

```
        "Naive Bayes": GaussianNB(),
    }

    meta_model = GradientBoostingClassifier(random_state=42)

    kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
    meta_cv_scores = []
    base_model_results = {name: {'accuracy': [], 'f1': [], 'auc': []} for name in base_models}

    print("\n=== Cross-Validation (Base Models + Stacking Meta-Model) ===")

    for fold, (train_idx, val_idx) in enumerate(kfold.split(X_train, y_train), 1):
        X_fold_train, X_fold_val = X_train.iloc[train_idx], X_train.iloc[val_idx]
        y_fold_train, y_fold_val = y_train.iloc[train_idx], y_train.iloc[val_idx]

        meta_features = np.zeros((X_fold_val.shape[0], len(base_models)))

        for i, (model_name, model) in enumerate(base_models.items()):

            selector = SequentialFeatureSelector(LogisticRegression(max_iter=1000), direction='forward')
            selector.fit(X_fold_train, y_fold_train)
            selected_features = X_fold_train.columns[selector.get_support()]

            model.fit(X_fold_train[selected_features], y_fold_train)

            if hasattr(model, "predict_proba"):
                prob = model.predict_proba(X_fold_val[selected_features])[:, 1]
            else:
                prob = model.decision_function(X_fold_val[selected_features])
                prob = (prob - prob.min()) / (prob.max() - prob.min())

            pred = model.predict(X_fold_val[selected_features])

            base_model_results[model_name]['accuracy'].append(accuracy_score(y_fold_val, pred))
            base_model_results[model_name]['f1'].append(f1_score(y_fold_val, pred))
            base_model_results[model_name]['auc'].append(roc_auc_score(y_fold_val, prob))

            meta_features[:, i] = prob

        meta_model.fit(meta_features, y_fold_val)
        y_pred_meta = meta_model.predict(meta_features)
        meta_cv_scores.append(accuracy_score(y_fold_val, y_pred_meta))
        print(f"  Fold {fold}: Meta Accuracy = {meta_cv_scores[-1]:.4f}")

    print("\n=== Average CV Performance of Base Models ===")
    for name, scores in base_model_results.items():
        print(f"  {name}")
        print(f"    Avg Accuracy: {np.mean(scores['accuracy']):.4f}")
        print(f"    Avg F1 Score: {np.mean(scores['f1']):.4f}")
        print(f"    Avg AUC: {np.mean(scores['auc']):.4f}")

    print(f"\nMeta-Model Mean CV Accuracy: {np.mean(meta_cv_scores):.4f}")

    meta_features_test = np.zeros((X_test.shape[0], len(base_models)))

    for i, (model_name, model) in enumerate(base_models.items()):
        selector = SequentialFeatureSelector(LogisticRegression(max_iter=1000), direction='forward')
        selector.fit(X_train, y_train)
        selected_features = X_train.columns[selector.get_support()]

        model.fit(X_train[selected_features], y_train)

        if hasattr(model, "predict_proba"):
            meta_features_test[:, i] = model.predict_proba(X_test[selected_features])[:, 1]
        else:
            prob = model.decision_function(X_test[selected_features])
            meta_features_test[:, i] = (prob - prob.min()) / (prob.max() - prob.min())

    meta_model.fit(meta_features, y_fold_val)
    y_pred_test = meta_model.predict(meta_features_test)
    y_pred_prob_test = meta_model.predict_proba(meta_features_test)[:, 1]

    print("\n=== Meta-Model Test Performance ===")
    print(f"  Accuracy: {accuracy_score(y_test, y_pred_test):.4f}")
    print(f"  Precision: {precision_score(y_test, y_pred_test):.4f}")
    print(f"  Recall: {recall_score(y_test, y_pred_test):.4f}")
    print(f"  F1 Score: {f1_score(y_test, y_pred_test):.4f}")
    print(f"  AUC: {roc_auc_score(y_test, y_pred_prob_test):.4f}")

    from sklearn.metrics import roc_curve, auc
    fpr, tpr, _ = roc_curve(y_test, y_pred_prob_test)
    roc_auc = auc(fpr, tpr)
```

```
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate (1 - Specificity)')
plt.ylabel('True Positive Rate (Sensitivity)')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```

META Model - GB by SMOTTomek

```
import pandas as pd
import numpy as np
from sklearn.model_selection import StratifiedKFold, train_test_split
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
from sklearn.feature_selection import SequentialFeatureSelector
from imblearn.combine import SMOTETomek


file_path = '/content/Data_Analysis_Jib.xlsx'
data = pd.read_excel(file_path)

X = data.drop('Target', axis=1)
y = data['Target']

smotetomek = SMOTETomek(random_state=42)
X_balanced, y_balanced = smotetomek.fit_resample(X, y)

X_train, X_test, y_train, y_test = train_test_split(
    X_balanced, y_balanced, test_size=0.2, random_state=42, stratify=y_balanced
)

base_models = {
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "SVM": SVC(probability=True, random_state=42),
    "Gradient Boosting": GradientBoostingClassifier(random_state=42),
    "K-Nearest Neighbors": KNeighborsClassifier(),
    "Naive Bayes": GaussianNB(),
}

meta_model = GradientBoostingClassifier(random_state=42)

kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
meta_cv_scores = []
base_model_results = {name: {'accuracy': [], 'f1': [], 'auc': []} for name in base_models}

print("\n=== Cross-Validation (Base Models + Stacking Meta-Model) ===")

for fold, (train_idx, val_idx) in enumerate(kfold.split(X_train, y_train), 1):
    X_fold_train, X_fold_val = X_train.iloc[train_idx], X_train.iloc[val_idx]
    y_fold_train, y_fold_val = y_train.iloc[train_idx], y_train.iloc[val_idx]

    meta_features = np.zeros((X_fold_val.shape[0], len(base_models)))

    for i, (model_name, model) in enumerate(base_models.items()):

        selector = SequentialFeatureSelector(LogisticRegression(max_iter=1000), direction='forward')
        selector.fit(X_fold_train, y_fold_train)
        selected_features = X_fold_train.columns[selector.get_support()]

        model.fit(X_fold_train[selected_features], y_fold_train)

        if hasattr(model, "predict_proba"):
            prob = model.predict_proba(X_fold_val[selected_features])[:, 1]
        else:
            prob = model.decision_function(X_fold_val[selected_features])
            prob = (prob - prob.min()) / (prob.max() - prob.min())

        pred = model.predict(X_fold_val[selected_features])

        base_model_results[model_name]['accuracy'].append(accuracy_score(y_fold_val, pred))
        base_model_results[model_name]['f1'].append(f1_score(y_fold_val, pred))
        base_model_results[model_name]['auc'].append(roc_auc_score(y_fold_val, prob))

        meta_features[:, i] = prob

    meta_model.fit(meta_features, y_fold_val)
    y_pred_meta = meta_model.predict(meta_features)
    meta_cv_scores.append(accuracy_score(y_fold_val, y_pred_meta))
    print(f"  Fold {fold}: Meta Accuracy = {meta_cv_scores[-1]:.4f}")

print("\n=== Average CV Performance of Base Models ===")
for name, scores in base_model_results.items():
    print(f"  {name}")
    print(f"    Avg Accuracy: {np.mean(scores['accuracy']):.4f}")
    print(f"    Avg F1 Score: {np.mean(scores['f1']):.4f}")
    print(f"    Avg AUC: {np.mean(scores['auc']):.4f}")
```

```
print(f"\nMeta-Model Mean CV Accuracy: {np.mean(meta_cv_scores):.4f}")

meta_features_test = np.zeros((X_test.shape[0], len(base_models)))

for i, (model_name, model) in enumerate(base_models.items()):
    selector = SequentialFeatureSelector(LogisticRegression(max_iter=1000), direction='forward')
    selector.fit(X_train, y_train)
    selected_features = X_train.columns[selector.get_support()]

    model.fit(X_train[selected_features], y_train)

    if hasattr(model, "predict_proba"):
        meta_features_test[:, i] = model.predict_proba(X_test[selected_features])[:, 1]
    else:
        prob = model.decision_function(X_test[selected_features])
        meta_features_test[:, i] = (prob - prob.min()) / (prob.max() - prob.min())

meta_model.fit(meta_features, y_fold_val)
y_pred_test = meta_model.predict(meta_features_test)
y_pred_prob_test = meta_model.predict_proba(meta_features_test)[:, 1]

print("\n=== Meta-Model Test Performance ===")
print(f"  Accuracy: {accuracy_score(y_test, y_pred_test):.4f}")
print(f"  Precision: {precision_score(y_test, y_pred_test):.4f}")
print(f"  Recall: {recall_score(y_test, y_pred_test):.4f}")
print(f"  F1 Score: {f1_score(y_test, y_pred_test):.4f}")
print(f"  AUC: {roc_auc_score(y_test, y_pred_prob_test):.4f}")

from sklearn.metrics import roc_curve, auc
fpr, tpr, _ = roc_curve(y_test, y_pred_prob_test)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate (1 - Specificity)')
plt.ylabel('True Positive Rate (Sensitivity)')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```

META Model - XGB by SMOTE

```
import pandas as pd
import numpy as np
from sklearn.model_selection import StratifiedKFold, train_test_split
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
from sklearn.feature_selection import SequentialFeatureSelector
from imblearn.over_sampling import SMOTE
import matplotlib.pyplot as plt
import xgboost as xgb

file_path = '/content/Data_Analysis_Jib.xlsx'
data = pd.read_excel(file_path)

X = data.drop('Target', axis=1)
y = data['Target']

smote = SMOTE(random_state=42)
X_balanced, y_balanced = smote.fit_resample(X, y)

X_train, X_test, y_train, y_test = train_test_split(
    X_balanced, y_balanced, test_size=0.2, random_state=42, stratify=y_balanced
)

base_models = {
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "SVM": SVC(probability=True, random_state=42),
    "Gradient Boosting": GradientBoostingClassifier(random_state=42),
    "K-Nearest Neighbors": KNeighborsClassifier(),
```

```
        "Naive Bayes": GaussianNB(),
    }

    meta_model = xgb.XGBClassifier()

    kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
    meta_cv_scores = []
    base_model_results = {name: {'accuracy': [], 'f1': [], 'auc': []} for name in base_models}

    print("\n=== Cross-Validation (Base Models + Stacking Meta-Model) ===")

    for fold, (train_idx, val_idx) in enumerate(kfold.split(X_train, y_train), 1):
        X_fold_train, X_fold_val = X_train.iloc[train_idx], X_train.iloc[val_idx]
        y_fold_train, y_fold_val = y_train.iloc[train_idx], y_train.iloc[val_idx]

        meta_features = np.zeros((X_fold_val.shape[0], len(base_models)))

        for i, (model_name, model) in enumerate(base_models.items()):

            selector = SequentialFeatureSelector(LogisticRegression(max_iter=1000), direction='forward')
            selector.fit(X_fold_train, y_fold_train)
            selected_features = X_fold_train.columns[selector.get_support()]

            model.fit(X_fold_train[selected_features], y_fold_train)

            if hasattr(model, "predict_proba"):
                prob = model.predict_proba(X_fold_val[selected_features])[:, 1]
            else:
                prob = model.decision_function(X_fold_val[selected_features])
                prob = (prob - prob.min()) / (prob.max() - prob.min())  # scale to [0,1]

            pred = model.predict(X_fold_val[selected_features])

            base_model_results[model_name]['accuracy'].append(accuracy_score(y_fold_val, pred))
            base_model_results[model_name]['f1'].append(f1_score(y_fold_val, pred))
            base_model_results[model_name]['auc'].append(roc_auc_score(y_fold_val, prob))

            meta_features[:, i] = prob

        meta_model.fit(meta_features, y_fold_val)
        y_pred_meta = meta_model.predict(meta_features)
        meta_cv_scores.append(accuracy_score(y_fold_val, y_pred_meta))
        print(f"  Fold {fold}: Meta Accuracy = {meta_cv_scores[-1]:.4f}")

    print("\n=== Average CV Performance of Base Models ===")
    for name, scores in base_model_results.items():
        print(f"  {name}")
        print(f"    Avg Accuracy: {np.mean(scores['accuracy']):.4f}")
        print(f"    Avg F1 Score: {np.mean(scores['f1']):.4f}")
        print(f"    Avg AUC: {np.mean(scores['auc']):.4f}")

    print(f"\nMeta-Model Mean CV Accuracy: {np.mean(meta_cv_scores):.4f}")

    meta_features_test = np.zeros((X_test.shape[0], len(base_models)))

    for i, (model_name, model) in enumerate(base_models.items()):
        selector = SequentialFeatureSelector(LogisticRegression(max_iter=1000), direction='forward')
        selector.fit(X_train, y_train)
        selected_features = X_train.columns[selector.get_support()]

        model.fit(X_train[selected_features], y_train)

        if hasattr(model, "predict_proba"):
            meta_features_test[:, i] = model.predict_proba(X_test[selected_features])[:, 1]
        else:
            prob = model.decision_function(X_test[selected_features])
            meta_features_test[:, i] = (prob - prob.min()) / (prob.max() - prob.min())

    meta_model.fit(meta_features, y_fold_val)  # Fit with latest fold for demo (optionally retrain on all data)
    y_pred_test = meta_model.predict(meta_features_test)
    y_pred_prob_test = meta_model.predict_proba(meta_features_test)[:, 1]

    print("\n=== Meta-Model Test Performance ===")
    print(f"  Accuracy: {accuracy_score(y_test, y_pred_test):.4f}")
    print(f"  Precision: {precision_score(y_test, y_pred_test):.4f}")
    print(f"  Recall: {recall_score(y_test, y_pred_test):.4f}")
    print(f"  F1 Score: {f1_score(y_test, y_pred_test):.4f}")
    print(f"  AUC: {roc_auc_score(y_test, y_pred_prob_test):.4f}")

    from sklearn.metrics import roc_curve, auc
    fpr, tpr, _ = roc_curve(y_test, y_pred_prob_test)
    roc_auc = auc(fpr, tpr)
```

```
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate (1 - Specificity)')
plt.ylabel('True Positive Rate (Sensitivity)')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```

META Model - XGB by ADASYN

```
import pandas as pd
import numpy as np
from sklearn.model_selection import StratifiedKFold, train_test_split
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
from sklearn.feature_selection import SequentialFeatureSelector
from imblearn.over_sampling import ADASYN
import matplotlib.pyplot as plt
import xgboost as xgb

file_path = '/content/Data_Analysis_Jib.xlsx'
data = pd.read_excel(file_path)

X = data.drop('Target', axis=1)
y = data['Target']

adasyn = ADASYN(random_state=42)
X_balanced, y_balanced = adasyn.fit_resample(X, y)

X_train, X_test, y_train, y_test = train_test_split(
    X_balanced, y_balanced, test_size=0.2, random_state=42, stratify=y_balanced
)

base_models = {
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "SVM": SVC(probability=True, random_state=42),
    "Gradient Boosting": GradientBoostingClassifier(random_state=42),
    "K-Nearest Neighbors": KNeighborsClassifier(),
    "Naive Bayes": GaussianNB(),
}

meta_model = xgb.XGBClassifier()

kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
meta_cv_scores = []
base_model_results = {name: {'accuracy': [], 'f1': [], 'auc': []} for name in base_models}

print("\n=== Cross-Validation (Base Models + Stacking Meta-Model) ===")

for fold, (train_idx, val_idx) in enumerate(kfold.split(X_train, y_train), 1):
    X_fold_train, X_fold_val = X_train.iloc[train_idx], X_train.iloc[val_idx]
    y_fold_train, y_fold_val = y_train.iloc[train_idx], y_train.iloc[val_idx]

    meta_features = np.zeros((X_fold_val.shape[0], len(base_models)))

    for i, (model_name, model) in enumerate(base_models.items()):

        selector = SequentialFeatureSelector(LogisticRegression(max_iter=1000), direction='forward')
        selector.fit(X_fold_train, y_fold_train)
        selected_features = X_fold_train.columns[selector.get_support()]

        model.fit(X_fold_train[selected_features], y_fold_train)

        if hasattr(model, "predict_proba"):
            prob = model.predict_proba(X_fold_val[selected_features])[:, 1]
        else:
            prob = model.decision_function(X_fold_val[selected_features])
            prob = (prob - prob.min()) / (prob.max() - prob.min())

        pred = model.predict(X_fold_val[selected_features])

        base_model_results[model_name]['accuracy'].append(accuracy_score(y_fold_val, pred))
        base_model_results[model_name]['f1'].append(f1_score(y_fold_val, pred))
        base_model_results[model_name]['auc'].append(roc_auc_score(y_fold_val, prob))

        meta_features[:, i] = prob

    meta_model.fit(meta_features, y_fold_val)
    y_pred_meta = meta_model.predict(meta_features)
    meta_cv_scores.append(accuracy_score(y_fold_val, y_pred_meta))
    print(f"  Fold {fold}: Meta Accuracy = {meta_cv_scores[-1]:.4f}")

print("\n=== Average CV Performance of Base Models ===")
for name, scores in base_model_results.items():
    print(f"  {name}")
    print(f"    Avg Accuracy: {np.mean(scores['accuracy']):.4f}")
    print(f"    Avg F1 Score: {np.mean(scores['f1']):.4f}")
```

```
        print(f"    Avg AUC: {np.mean(scores['auc']):.4f}")

    print(f"\nMeta-Model Mean CV Accuracy: {np.mean(meta_cv_scores):.4f}")

    meta_features_test = np.zeros((X_test.shape[0], len(base_models)))

    for i, (model_name, model) in enumerate(base_models.items()):
        selector = SequentialFeatureSelector(LogisticRegression(max_iter=1000), direction='forward')
        selector.fit(X_train, y_train)
        selected_features = X_train.columns[selector.get_support()]

        model.fit(X_train[selected_features], y_train)

        if hasattr(model, "predict_proba"):
            meta_features_test[:, i] = model.predict_proba(X_test[selected_features])[:, 1]
        else:
            prob = model.decision_function(X_test[selected_features])
            meta_features_test[:, i] = (prob - prob.min()) / (prob.max() - prob.min())

    meta_model.fit(meta_features, y_fold_val)
    y_pred_test = meta_model.predict(meta_features_test)
    y_pred_prob_test = meta_model.predict_proba(meta_features_test)[:, 1]

    print("\n=== Meta-Model Test Performance ===")
    print(f"  Accuracy: {accuracy_score(y_test, y_pred_test):.4f}")
    print(f"  Precision: {precision_score(y_test, y_pred_test):.4f}")
    print(f"  Recall: {recall_score(y_test, y_pred_test):.4f}")
    print(f"  F1 Score: {f1_score(y_test, y_pred_test):.4f}")
    print(f"  AUC: {roc_auc_score(y_test, y_pred_prob_test):.4f}")

    from sklearn.metrics import roc_curve, auc
    fpr, tpr, _ = roc_curve(y_test, y_pred_prob_test)
    roc_auc = auc(fpr, tpr)

    plt.figure(figsize=(8, 6))
    plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
    plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate (1 - Specificity)')
    plt.ylabel('True Positive Rate (Sensitivity)')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend(loc='lower right')
    plt.show()
```

META Model - XGB by SMOTEEN

```
import pandas as pd
import numpy as np
from sklearn.model_selection import StratifiedKFold, train_test_split
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
from sklearn.feature_selection import SequentialFeatureSelector
from imblearn.combine import SMOTEENN
import xgboost as xgb
import matplotlib.pyplot as plt

file_path = '/content/Data_Analysis_Jib.xlsx'
data = pd.read_excel(file_path)

X = data.drop('Target', axis=1)
y = data['Target']

smoteenn = SMOTEENN(random_state=42)
X_balanced, y_balanced = smoteenn.fit_resample(X, y)

X_train, X_test, y_train, y_test = train_test_split(
    X_balanced, y_balanced, test_size=0.2, random_state=42, stratify=y_balanced
)

base_models = {
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "SVM": SVC(probability=True, random_state=42),
    "Gradient Boosting": GradientBoostingClassifier(random_state=42),
    "K-Nearest Neighbors": KNeighborsClassifier(),
    "Naive Bayes": GaussianNB(),
}

meta_model = xgb.XGBClassifier()

kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
meta_cv_scores = []
base_model_results = {name: {'accuracy': [], 'f1': [], 'auc': []} for name in base_models}

print("\n=== Cross-Validation (Base Models + Stacking Meta-Model) ===")

for fold, (train_idx, val_idx) in enumerate(kfold.split(X_train, y_train), 1):
    X_fold_train, X_fold_val = X_train.iloc[train_idx], X_train.iloc[val_idx]
    y_fold_train, y_fold_val = y_train.iloc[train_idx], y_train.iloc[val_idx]

    meta_features = np.zeros((X_fold_val.shape[0], len(base_models)))

    for i, (model_name, model) in enumerate(base_models.items()):

        selector = SequentialFeatureSelector(LogisticRegression(max_iter=1000), direction='forward')
        selector.fit(X_fold_train, y_fold_train)
        selected_features = X_fold_train.columns[selector.get_support()]

        model.fit(X_fold_train[selected_features], y_fold_train)

        if hasattr(model, "predict_proba"):
            prob = model.predict_proba(X_fold_val[selected_features])[:, 1]
        else:
            prob = model.decision_function(X_fold_val[selected_features])
            prob = (prob - prob.min()) / (prob.max() - prob.min())

        pred = model.predict(X_fold_val[selected_features])

        base_model_results[model_name]['accuracy'].append(accuracy_score(y_fold_val, pred))
        base_model_results[model_name]['f1'].append(f1_score(y_fold_val, pred))
        base_model_results[model_name]['auc'].append(roc_auc_score(y_fold_val, prob))

        meta_features[:, i] = prob

    meta_model.fit(meta_features, y_fold_val)
    y_pred_meta = meta_model.predict(meta_features)
    meta_cv_scores.append(accuracy_score(y_fold_val, y_pred_meta))
    print(f"  Fold {fold}: Meta Accuracy = {meta_cv_scores[-1]:.4f}")

print("\n=== Average CV Performance of Base Models ===")
for name, scores in base_model_results.items():
    print(f"  {name}")
    print(f"    Avg Accuracy: {np.mean(scores['accuracy']):.4f}")
    print(f"    Avg F1 Score: {np.mean(scores['f1']):.4f}")
```

```
        print(f"    Avg AUC: {np.mean(scores['auc']):.4f}")

    print(f"\nMeta-Model Mean CV Accuracy: {np.mean(meta_cv_scores):.4f}")

    meta_features_test = np.zeros((X_test.shape[0], len(base_models)))

    for i, (model_name, model) in enumerate(base_models.items()):
        selector = SequentialFeatureSelector(LogisticRegression(max_iter=1000), direction='forward')
        selector.fit(X_train, y_train)
        selected_features = X_train.columns[selector.get_support()]

        model.fit(X_train[selected_features], y_train)

        if hasattr(model, "predict_proba"):
            meta_features_test[:, i] = model.predict_proba(X_test[selected_features])[:, 1]
        else:
            prob = model.decision_function(X_test[selected_features])
            meta_features_test[:, i] = (prob - prob.min()) / (prob.max() - prob.min())

    meta_model.fit(meta_features, y_fold_val)
    y_pred_test = meta_model.predict(meta_features_test)
    y_pred_prob_test = meta_model.predict_proba(meta_features_test)[:, 1]

    print("\n=== Meta-Model Test Performance ===")
    print(f"  Accuracy: {accuracy_score(y_test, y_pred_test):.4f}")
    print(f"  Precision: {precision_score(y_test, y_pred_test):.4f}")
    print(f"  Recall: {recall_score(y_test, y_pred_test):.4f}")
    print(f"  F1 Score: {f1_score(y_test, y_pred_test):.4f}")
    print(f"  AUC: {roc_auc_score(y_test, y_pred_prob_test):.4f}")

    from sklearn.metrics import roc_curve, auc
    fpr, tpr, _ = roc_curve(y_test, y_pred_prob_test)
    roc_auc = auc(fpr, tpr)

    plt.figure(figsize=(8, 6))
    plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
    plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate (1 - Specificity)')
    plt.ylabel('True Positive Rate (Sensitivity)')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend(loc='lower right')
    plt.show()
```

META Model - XGB by SMOTTomek

```
import pandas as pd
import numpy as np
from sklearn.model_selection import StratifiedKFold, train_test_split
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
from sklearn.feature_selection import SequentialFeatureSelector
from imblearn.combine import SMOTETomek
import xgboost as xgb

file_path = '/content/Data_Analysis_Jib.xlsx'
data = pd.read_excel(file_path)

X = data.drop('Target', axis=1)
y = data['Target']

smotetomek = SMOTETomek(random_state=42)
X_balanced, y_balanced = smotetomek.fit_resample(X, y)

X_train, X_test, y_train, y_test = train_test_split(
    X_balanced, y_balanced, test_size=0.2, random_state=42, stratify=y_balanced
)

base_models = {
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "SVM": SVC(probability=True, random_state=42),
    "Gradient Boosting": GradientBoostingClassifier(random_state=42),
    "K-Nearest Neighbors": KNeighborsClassifier(),
    "Naive Bayes": GaussianNB(),
}

meta_model = xgb.XGBClassifier()

kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
meta_cv_scores = []
base_model_results = {name: {'accuracy': [], 'f1': [], 'auc': []} for name in base_models}

print("\n=== Cross-Validation (Base Models + Stacking Meta-Model) ===")

for fold, (train_idx, val_idx) in enumerate(kfold.split(X_train, y_train), 1):
    X_fold_train, X_fold_val = X_train.iloc[train_idx], X_train.iloc[val_idx]
    y_fold_train, y_fold_val = y_train.iloc[train_idx], y_train.iloc[val_idx]

    meta_features = np.zeros((X_fold_val.shape[0], len(base_models)))

    for i, (model_name, model) in enumerate(base_models.items()):

        selector = SequentialFeatureSelector(LogisticRegression(max_iter=1000), direction='forward')
        selector.fit(X_fold_train, y_fold_train)
        selected_features = X_fold_train.columns[selector.get_support()]

        model.fit(X_fold_train[selected_features], y_fold_train)

        if hasattr(model, "predict_proba"):
            prob = model.predict_proba(X_fold_val[selected_features])[:, 1]
        else:
            prob = model.decision_function(X_fold_val[selected_features])
            prob = (prob - prob.min()) / (prob.max() - prob.min())

        pred = model.predict(X_fold_val[selected_features])

        base_model_results[model_name]['accuracy'].append(accuracy_score(y_fold_val, pred))
        base_model_results[model_name]['f1'].append(f1_score(y_fold_val, pred))
        base_model_results[model_name]['auc'].append(roc_auc_score(y_fold_val, prob))

        meta_features[:, i] = prob

    meta_model.fit(meta_features, y_fold_val)
    y_pred_meta = meta_model.predict(meta_features)
    meta_cv_scores.append(accuracy_score(y_fold_val, y_pred_meta))
    print(f"  Fold {fold}: Meta Accuracy = {meta_cv_scores[-1]:.4f}")

print("\n=== Average CV Performance of Base Models ===")
for name, scores in base_model_results.items():
    print(f"  {name}")
    print(f"    Avg Accuracy: {np.mean(scores['accuracy']):.4f}")
    print(f"    Avg F1 Score: {np.mean(scores['f1']):.4f}")
    print(f"    Avg AUC: {np.mean(scores['auc']):.4f}")
```

```
print(f"\nMeta-Model Mean CV Accuracy: {np.mean(meta_cv_scores):.4f}")

meta_features_test = np.zeros((X_test.shape[0], len(base_models)))

for i, (model_name, model) in enumerate(base_models.items()):
    selector = SequentialFeatureSelector(LogisticRegression(max_iter=1000), direction='forward')
    selector.fit(X_train, y_train)
    selected_features = X_train.columns[selector.get_support()]

    model.fit(X_train[selected_features], y_train)

    if hasattr(model, "predict_proba"):
        meta_features_test[:, i] = model.predict_proba(X_test[selected_features])[:, 1]
    else:
        prob = model.decision_function(X_test[selected_features])
        meta_features_test[:, i] = (prob - prob.min()) / (prob.max() - prob.min())

meta_model.fit(meta_features, y_fold_val)
y_pred_test = meta_model.predict(meta_features_test)
y_pred_prob_test = meta_model.predict_proba(meta_features_test)[:, 1]

print("\n=== Meta-Model Test Performance ===")
print(f"  Accuracy: {accuracy_score(y_test, y_pred_test):.4f}")
print(f"  Precision: {precision_score(y_test, y_pred_test):.4f}")
print(f"  Recall: {recall_score(y_test, y_pred_test):.4f}")
print(f"  F1 Score: {f1_score(y_test, y_pred_test):.4f}")
print(f"  AUC: {roc_auc_score(y_test, y_pred_prob_test):.4f}")

from sklearn.metrics import roc_curve, auc
fpr, tpr, _ = roc_curve(y_test, y_pred_prob_test)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate (1 - Specificity)')
plt.ylabel('True Positive Rate (Sensitivity)')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```

META Model - MLP by SMOTE

```
import pandas as pd
import numpy as np
from sklearn.model_selection import StratifiedKFold, train_test_split
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
from sklearn.feature_selection import SequentialFeatureSelector
from imblearn.over_sampling import SMOTE
import matplotlib.pyplot as plt
from sklearn.neural_network import MLPClassifier

file_path = '/content/Data_Analysis_Jib.xlsx'
data = pd.read_excel(file_path)

X = data.drop('Target', axis=1)
y = data['Target']

smote = SMOTE(random_state=42)
X_balanced, y_balanced = smote.fit_resample(X, y)

X_train, X_test, y_train, y_test = train_test_split(
    X_balanced, y_balanced, test_size=0.2, random_state=42, stratify=y_balanced
)

base_models = {
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "SVM": SVC(probability=True, random_state=42),
    "Gradient Boosting": GradientBoostingClassifier(random_state=42),
    "K-Nearest Neighbors": KNeighborsClassifier(),
```

```
        "Naive Bayes": GaussianNB(),
    }

    meta_model = MLPClassifier(
        hidden_layer_sizes=(64, 32),
        activation='relu',
        solver='adam',
        random_state=42,
        max_iter=500
    )

    kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
    meta_cv_scores = []
    base_model_results = {name: {'accuracy': [], 'f1': [], 'auc': []} for name in base_models}

    print("\n=== Cross-Validation (Base Models + Stacking Meta-Model) ===")

    for fold, (train_idx, val_idx) in enumerate(kfold.split(X_train, y_train), 1):
        X_fold_train, X_fold_val = X_train.iloc[train_idx], X_train.iloc[val_idx]
        y_fold_train, y_fold_val = y_train.iloc[train_idx], y_train.iloc[val_idx]

        meta_features = np.zeros((X_fold_val.shape[0], len(base_models)))

        for i, (model_name, model) in enumerate(base_models.items()):

            selector = SequentialFeatureSelector(LogisticRegression(max_iter=1000), direction='forward')
            selector.fit(X_fold_train, y_fold_train)
            selected_features = X_fold_train.columns[selector.get_support()]

            model.fit(X_fold_train[selected_features], y_fold_train)

            if hasattr(model, "predict_proba"):
                prob = model.predict_proba(X_fold_val[selected_features])[:, 1]
            else:
                prob = model.decision_function(X_fold_val[selected_features])
                prob = (prob - prob.min()) / (prob.max() - prob.min())  # scale to [0,1]

            pred = model.predict(X_fold_val[selected_features])

            base_model_results[model_name]['accuracy'].append(accuracy_score(y_fold_val, pred))
            base_model_results[model_name]['f1'].append(f1_score(y_fold_val, pred))
            base_model_results[model_name]['auc'].append(roc_auc_score(y_fold_val, prob))

            meta_features[:, i] = prob

        meta_model.fit(meta_features, y_fold_val)
        y_pred_meta = meta_model.predict(meta_features)
        meta_cv_scores.append(accuracy_score(y_fold_val, y_pred_meta))
        print(f"  Fold {fold}: Meta Accuracy = {meta_cv_scores[-1]:.4f}")

    print("\n=== Average CV Performance of Base Models ===")
    for name, scores in base_model_results.items():
        print(f"  {name}")
        print(f"    Avg Accuracy: {np.mean(scores['accuracy']):.4f}")
        print(f"    Avg F1 Score: {np.mean(scores['f1']):.4f}")
        print(f"    Avg AUC: {np.mean(scores['auc']):.4f}")

    print(f"\nMeta-Model Mean CV Accuracy: {np.mean(meta_cv_scores):.4f}")

    meta_features_test = np.zeros((X_test.shape[0], len(base_models)))

    for i, (model_name, model) in enumerate(base_models.items()):
        selector = SequentialFeatureSelector(LogisticRegression(max_iter=1000), direction='forward')
        selector.fit(X_train, y_train)
        selected_features = X_train.columns[selector.get_support()]

        model.fit(X_train[selected_features], y_train)

        if hasattr(model, "predict_proba"):
            meta_features_test[:, i] = model.predict_proba(X_test[selected_features])[:, 1]
        else:
            prob = model.decision_function(X_test[selected_features])
            meta_features_test[:, i] = (prob - prob.min()) / (prob.max() - prob.min())

    meta_model.fit(meta_features, y_fold_val)  # Fit with latest fold for demo (optionally retrain on all data)
    y_pred_test = meta_model.predict(meta_features_test)
    y_pred_prob_test = meta_model.predict_proba(meta_features_test)[:, 1]

    print("\n=== Meta-Model Test Performance ===")
    print(f"  Accuracy: {accuracy_score(y_test, y_pred_test):.4f}")
    print(f"  Precision: {precision_score(y_test, y_pred_test):.4f}")
    print(f"  Recall: {recall_score(y_test, y_pred_test):.4f}")
```

```
print(f"  F1 Score: {f1_score(y_test, y_pred_test):.4f}")
print(f"  AUC: {roc_auc_score(y_test, y_pred_prob_test):.4f}")

from sklearn.metrics import roc_curve, auc
fpr, tpr, _ = roc_curve(y_test, y_pred_prob_test)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate (1 - Specificity)')
plt.ylabel('True Positive Rate (Sensitivity)')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```

META Model - MLP by ADASYN

```
import pandas as pd
import numpy as np
from sklearn.model_selection import StratifiedKFold, train_test_split
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
from sklearn.feature_selection import SequentialFeatureSelector
from imblearn.over_sampling import ADASYN
import matplotlib.pyplot as plt
from sklearn.neural_network import MLPClassifier

file_path = '/content/Data_Analysis_Jib.xlsx'
data = pd.read_excel(file_path)

X = data.drop('Target', axis=1)
y = data['Target']

adasyn = ADASYN(random_state=42)
X_balanced, y_balanced = adasyn.fit_resample(X, y)

X_train, X_test, y_train, y_test = train_test_split(
    X_balanced, y_balanced, test_size=0.2, random_state=42, stratify=y_balanced
)

base_models = {
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "SVM": SVC(probability=True, random_state=42),
    "Gradient Boosting": GradientBoostingClassifier(random_state=42),
    "K-Nearest Neighbors": KNeighborsClassifier(),
    "Naive Bayes": GaussianNB(),
}

meta_model = MLPClassifier(
    hidden_layer_sizes=(64, 32),
    activation='relu',
    solver='adam',
    random_state=42,
    max_iter=500
)

kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
meta_cv_scores = []
base_model_results = {name: {'accuracy': [], 'f1': [], 'auc': []} for name in base_models}

print("\n=== Cross-Validation (Base Models + Stacking Meta-Model) ===")
```