```
# Essential Imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import datasets, preprocessing, model_selection, metrics
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import (RandomForestClassifier, GradientBoostingClassifier,
                              BaggingClassifier, AdaBoostClassifier, GradientBoostingRegressor)
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import (classification_report, confusion_matrix, accuracy_score,
                             mean_absolute_error, mean_squared_error, r2_score, silhouette_score)

# 1. DATA EXPLORATION & PREPROCESSING
def preprocess_data(df, target_column=None):
    """Complete data preprocessing pipeline"""
    # Handle missing values
    df_clean = handle_missing_values(df, 'mean')

    # Separate features and target
    if target_column:
        X = df_clean.drop(target_column, axis=1)
        y = df_clean[target_column]
    else:
        X = df_clean
        y = None

    # Scale numerical features
    scaler = StandardScaler()
    numerical_cols = X.select_dtypes(include=[np.number]).columns
    X[numerical_cols] = scaler.fit_transform(X[numerical_cols])

    return X, y, scaler

# 2. SUPERVISED LEARNING - REGRESSION
def linear_regression_example(X, y):
    """Linear Regression Implementation"""
    from sklearn.model_selection import train_test_split
    from sklearn.linear_model import LinearRegression

    # Split data
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

    # Create and train model
    model = LinearRegression()
    model.fit(X_train, y_train)

    # Predictions
    y_pred = model.predict(X_test)

    # Evaluation
    mae = mean_absolute_error(y_test, y_pred)
    mse = mean_squared_error(y_test, y_pred)
    rmse = np.sqrt(mse)
    r2 = r2_score(y_test, y_pred)

    print("📈 Linear Regression Results:")
    print(f"MAE: {mae:.4f}, MSE: {mse:.4f}, RMSE: {rmse:.4f}, R²: {r2:.4f}")

    return model, y_pred

def random_forest_regression(X, y):
    """Random Forest for Regression"""
    from sklearn.ensemble import RandomForestRegressor

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

    model = RandomForestRegressor(n_estimators=100, random_state=42)
    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)

    # Feature importance
```

```
    feature_importance = pd.DataFrame({
        'feature': X.columns,
        'importance': model.feature_importances_
    }).sort_values('importance', ascending=False)

    print("🌲 Random Forest Feature Importance:")
    print(feature_importance.head())

    return model, y_pred, feature_importance

# 3. SUPERVISED LEARNING - CLASSIFICATION

def logistic_regression_example(X, y):
    """Logistic Regression Classification"""
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)

    model = LogisticRegression(random_state=42)
    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)
    y_pred_proba = model.predict_proba(X_test)

    print("📊 Logistic Regression Results:")
    print(classification_report(y_test, y_pred))
    print("Confusion Matrix:")
    print(confusion_matrix(y_test, y_pred))

    return model, y_pred, y_pred_proba

def random_forest_classification(X, y):
    """Random Forest for Classification"""
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)

    model = RandomForestClassifier(n_estimators=100, random_state=42)
    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)

    print("🌲 Random Forest Classification Results:")
    print(classification_report(y_test, y_pred))

    return model, y_pred

def svm_classification(X, y):
    """Support Vector Machine Classification"""
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)

    model = SVC(kernel='rbf', probability=True, random_state=42)
    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)

    print("🎯 SVM Classification Results:")
    print(classification_report(y_test, y_pred))

    return model, y_pred

# 4. UNSUPERVISED LEARNING - CLUSTERING

def kmeans_clustering(X, n_clusters=3):
    """K-Means Clustering Implementation"""
    model = KMeans(n_clusters=n_clusters, random_state=42)
    labels = model.fit_predict(X)

    # Calculate silhouette score
    silhouette_avg = silhouette_score(X, labels)
    print(f"🎯 K-Means Clustering (k={n_clusters})")
    print(f"Silhouette Score: {silhouette_avg:.4f}")

    return model, labels

def find_optimal_k(X, max_k=10):
    """Find optimal number of clusters using elbow method"""
    distortions = []
    K = range(1, max_k + 1)

    for k in K:
        kmeans = KMeans(n_clusters=k, random_state=42)
        kmeans.fit(X)
        distortions.append(kmeans.inertia_)
```

```
    # Plot elbow curve
    plt.figure(figsize=(10, 6))
    plt.plot(K, distortions, 'bx-')
    plt.xlabel('Number of Clusters (k)')
    plt.ylabel('Distortion')
    plt.title('Elbow Method for Optimal k')
    plt.show()

    return distortions

# 5. DIMENSIONALITY REDUCTION
def pca_analysis(X, n_components=None, variance_threshold=0.95):
    """Principal Component Analysis"""
    if n_components is None:
        pca = PCA(n_components=variance_threshold)
    else:
        pca = PCA(n_components=n_components)

    X_pca = pca.fit_transform(X)

    print("🔍 PCA Analysis Results:")
    print(f"Original Shape: {X.shape}")
    print(f"PCA Shape: {X_pca.shape}")
    print(f"Explained Variance Ratio: {pca.explained_variance_ratio_}")
    print(f"Total Variance Explained: {pca.explained_variance_ratio_.sum():.4f}")

    # Plot explained variance
    plt.figure(figsize=(10, 6))
    plt.plot(range(1, len(pca.explained_variance_ratio_) + 1),
             np.cumsum(pca.explained_variance_ratio_))
    plt.xlabel('Number of Components')
    plt.ylabel('Cumulative Explained Variance')
    plt.title('PCA Explained Variance')
    plt.grid(True)
    plt.show()

    return pca, X_pca

# 6. MODEL EVALUATION & HYPERPARAMETER TUNING
def evaluate_classification_model(model, X_test, y_test):
    """Comprehensive classification model evaluation"""
    y_pred = model.predict(X_test)
    y_pred_proba = model.predict_proba(X_test)[:, 1] if hasattr(model, "predict_proba") else None

    print("📊 Model Evaluation Metrics:")
    print("="*50)
    print(classification_report(y_test, y_pred))
    print("\nConfusion Matrix:")
    print(confusion_matrix(y_test, y_pred))
    print(f"\nAccuracy Score: {accuracy_score(y_test, y_pred):.4f}")

    return y_pred, y_pred_proba

def hyperparameter_tuning(model, param_grid, X, y, cv=5):
    """Hyperparameter tuning using GridSearchCV"""
    grid_search = GridSearchCV(
        estimator=model,
        param_grid=param_grid,
        cv=cv,
        scoring='accuracy',
        n_jobs=-1,
        verbose=1
    )

    grid_search.fit(X, y)

    print("🎯 Hyperparameter Tuning Results:")
    print(f"Best Parameters: {grid_search.best_params_}")
    print(f"Best Cross-Validation Score: {grid_search.best_score_:.4f}")

    return grid_search

# 7. COMPREHENSIVE ML PIPELINE
def complete_ml_pipeline(df, target_column, problem_type='classification'):
    """
    Complete ML pipeline from data loading to model evaluation

    Parameters:
    - df: pandas DataFrame
    - target_column: name of target variable
    - problem_type: 'classification' or 'regression'
```

```
    """
    print("🚀 STARTING COMPLETE ML PIPELINE")
    print("="*60)

    # Step 1: Data Exploration
    print("1. 📊 DATA EXPLORATION")
    explore_data(df)

    # Step 2: Preprocessing
    print("\n2. 🔄 DATA PREPROCESSING")
    X, y, scaler = preprocess_data(df, target_column)

    # Step 3: Model Training based on problem type
    print(f"\n3. 🤖 MODEL TRAINING ({problem_type.upper()})")

    if problem_type == 'classification':
        models = {
            'Logistic Regression': LogisticRegression(random_state=42),
            'Random Forest': RandomForestClassifier(random_state=42),
            'SVM': SVC(random_state=42, probability=True)
        }
    else:  # regression
        models = {
            'Linear Regression': LinearRegression(),
            'Random Forest': RandomForestRegressor(random_state=42)
        }

    # Train and evaluate each model
    results = {}
    for name, model in models.items():
        print(f"\n--- {name} ---")
        if problem_type == 'classification':
            X_train, X_test, y_train, y_test = train_test_split(
                X, y, test_size=0.3, random_state=42, stratify=y)
        else:
            X_train, X_test, y_train, y_test = train_test_split(
                X, y, test_size=0.3, random_state=42)

        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)

        if problem_type == 'classification':
            accuracy = accuracy_score(y_test, y_pred)
            results[name] = accuracy
            print(f"Accuracy: {accuracy:.4f}")
        else:
            r2 = r2_score(y_test, y_pred)
            results[name] = r2
            print(f"R² Score: {r2:.4f}")

    # Find best model
    best_model_name = max(results, key=results.get)
    print(f"\n🏆 BEST MODEL: {best_model_name} (Score: {results[best_model_name]:.4f})")

    return results, models[best_model_name]

# 8. QUICK START EXAMPLES
def quick_classification_example():
    """Quick classification example with iris dataset"""
    from sklearn.datasets import load_iris

    # Load data
    iris = load_iris()
    X, y = iris.data, iris.target
    feature_names = iris.feature_names

    print("🌸 Iris Dataset Classification Example")
    print("Features:", feature_names)
    print("Target classes:", np.unique(y))

    # Preprocessing
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    # Train model
    X_train, X_test, y_train, y_test = train_test_split(
        X_scaled, y, test_size=0.3, random_state=42, stratify=y)

    model = RandomForestClassifier(n_estimators=100, random_state=42)
    model.fit(X_train, y_train)
```

```
    # Evaluate
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)

    print(f"Model Accuracy: {accuracy:.4f}")
    print("\nClassification Report:")
    print(classification_report(y_test, y_pred, target_names=iris.target_names))

    return model, accuracy

def quick_regression_example():
    """Quick regression example with diabetes dataset"""
    from sklearn.datasets import load_diabetes

    # Load data
    diabetes = load_diabetes()
    X, y = diabetes.data, diabetes.target

    print("💊 Diabetes Dataset Regression Example")
    print(f"Dataset shape: {X.shape}")

    # Train model
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

    model = RandomForestRegressor(n_estimators=100, random_state=42)
    model.fit(X_train, y_train)

    # Evaluate
    y_pred = model.predict(X_test)
    r2 = r2_score(y_test, y_pred)
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))

    print(f"R² Score: {r2:.4f}")
    print(f"RMSE: {rmse:.4f}")

    return model, r2, rmse


# 9. VISUALIZATION TOOLS
def plot_feature_importance(model, feature_names, top_n=10):
    """Plot feature importance for tree-based models"""
    if hasattr(model, 'feature_importances_'):
        importance = model.feature_importances_
        indices = np.argsort(importance)[::-1]

        plt.figure(figsize=(10, 6))
        plt.title("Feature Importance")
        plt.bar(range(min(top_n, len(importance))),
                importance[indices][:top_n])
        plt.xticks(range(min(top_n, len(importance))),
                   [feature_names[i] for i in indices[:top_n]], rotation=45)
        plt.tight_layout()
        plt.show()
    else:
        print("Model doesn't have feature_importances_ attribute")

def plot_confusion_matrix_heatmap(y_true, y_pred, class_names=None):
    """Plot confusion matrix as heatmap"""
    cm = confusion_matrix(y_true, y_pred)

    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=class_names, yticklabels=class_names)
    plt.title('Confusion Matrix')
    plt.ylabel('True Label')
    plt.xlabel('Predicted Label')
    plt.show()
```