```
import pandas as pd
import numpy as np
from sklearn.model_selection import StratifiedKFold
from sklearn.preprocessing import StandardScaler
from lightgbm import LGBMClassifier
from xgboost import XGBClassifier
from catboost import CatBoostClassifier
from sklearn.metrics import roc_auc_score
import warnings
warnings.filterwarnings('ignore')

# Importing Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, cross_val_score, RandomizedSearchCV, KFold
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score, roc_curve, confusion_matrix, classification_report
from sklearn.feature_selection import RFE
from catboost import CatBoostClassifier
from sklearn.impute import SimpleImputer
from scipy import stats

# Load the data
data = pd.read_csv('dota2_data.csv')  # Replace with your dataset

# Data Preprocessing
# Handling Missing Values
imputer = SimpleImputer(strategy='most_frequent')  # Use 'mean' for numeric features, 'most_frequent' for categorical
data_imputed = data.copy()
data_imputed[data_imputed.columns] = imputer.fit_transform(data_imputed)

# Remove Duplicates
data_imputed = data_imputed.drop_duplicates()

# Converting Categorical to Numeric
categorical_cols = data_imputed.select_dtypes(include=['object']).columns
data_imputed[categorical_cols] = data_imputed[categorical_cols].apply(pd.to_numeric, errors='coerce')

# Outlier Handling
def handle_outliers(df):
    for col in df.select_dtypes(include=[np.number]).columns:
        Q1 = df[col].quantile(0.25)
        Q3 = df[col].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR
        df[col] = np.where(df[col] < lower_bound, df[col].mean(), df[col])
        df[col] = np.where(df[col] > upper_bound, df[col].mean(), df[col])
    return df

data_imputed = handle_outliers(data_imputed)

# Train-Test Split (80-20)
X = data_imputed.drop('target', axis=1)  # Assuming 'target' is your column to predict
y = data_imputed['target']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# Scaling the Data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Exploratory Data Analysis (EDA)
# Plot feature distributions
for col in X.columns:
    plt.figure(figsize=(6,4))
    sns.histplot(X[col], kde=True, bins=20)
    plt.title(f'Distribution of {col}')
    plt.show()

# Apply transformations (Log for right skew, Box-Cox for left skew)
for col in X.columns:
    skewness = X[col].skew()
    if skewness > 0:
        X[col] = np.log1p(X[col])  # Log transformation for right-skewed features
    elif skewness < 0:
        X[col] = stats.boxcox(X[col] + 1)[0]  # Box-Cox for left-skewed features

# Feature Engineering
# Remove near-zero variance features
from sklearn.feature_selection import VarianceThreshold
selector = VarianceThreshold(threshold=0.01)
X_train_filtered = selector.fit_transform(X_train_scaled)
X_test_filtered = selector.transform(X_test_scaled)

# Recursive Feature Elimination (RFE)
rfe = RFE(LogisticRegression(), n_features_to_select=10)
X_train_rfe = rfe.fit_transform(X_train_filtered, y_train)
X_test_rfe = rfe.transform(X_test_filtered)

# Apply PCA after Feature Selection
pca = PCA(n_components=0.95)  # Retain 95% of variance
X_train_pca = pca.fit_transform(X_train_rfe)
X_test_pca = pca.transform(X_test_rfe)

# Model Training
# Logistic Regression
log_model = LogisticRegression(max_iter=1000)
log_model.fit(X_train_pca, y_train)

# CatBoost Classifier
catboost_model = CatBoostClassifier(learning_rate=0.1, iterations=1000, depth=6, cat_features=[i for i in range(X_train_pca.shape[1])])
catboost_model.fit(X_train_pca, y_train)

# Model Evaluation - ROC AUC
y_pred_log = log_model.predict(X_test_pca)
y_pred_catboost = catboost_model.predict(X_test_pca)

roc_auc_log = roc_auc_score(y_test, y_pred_log)
```

```
roc_auc_catboost = roc_auc_score(y_test, y_pred_catboost)

print(f'Logistic Regression ROC AUC: {roc_auc_log:.2f}')
print(f'CatBoost ROC AUC: {roc_auc_catboost:.2f}')

# Plot ROC Curve
fpr_log, tpr_log, _ = roc_curve(y_test, log_model.predict_proba(X_test_pca)[:, 1])
fpr_catboost, tpr_catboost, _ = roc_curve(y_test, catboost_model.predict_proba(X_test_pca)[:, 1])

plt.figure(figsize=(8,6))
plt.plot(fpr_log, tpr_log, color='blue', label=f'Logistic Regression (AUC = {roc_auc_log:.2f})')
plt.plot(fpr_catboost, tpr_catboost, color='green', label=f'CatBoost (AUC = {roc_auc_catboost:.2f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve Comparison')
plt.legend()
plt.show()

# Confusion Matrix and Classification Report for the Best Model (CatBoost)
cm = confusion_matrix(y_test, y_pred_catboost)
print("Confusion Matrix:\n", cm)
print("Classification Report:\n", classification_report(y_test, y_pred_catboost))

# Hyperparameter Tuning with RandomizedSearchCV for CatBoost
param_grid = {
    'iterations': [500, 1000, 1500],
    'depth': [4, 6, 8],
    'learning_rate': [0.05, 0.1, 0.2],
}

random_search = RandomizedSearchCV(catboost_model, param_grid, cv=5, n_iter=5, verbose=2, n_jobs=-1)
random_search.fit(X_train_pca, y_train)
print(f'Best parameters found: {random_search.best_params_}')

# 5-Fold Cross Validation
cv = KFold(n_splits=5, shuffle=True, random_state=42)

# Logistic Regression Cross Validation
log_cv_scores = cross_val_score(log_model, X_train_pca, y_train, cv=cv, scoring='roc_auc')
print(f'Logistic Regression Cross-Validation ROC AUC: {log_cv_scores.mean():.2f} ± {log_cv_scores.std():.2f}')

# CatBoost Cross Validation
catboost_cv_scores = cross_val_score(catboost_model, X_train_pca, y_train, cv=cv, scoring='roc_auc')
print(f'CatBoost Cross-Validation ROC AUC: {catboost_cv_scores.mean():.2f} ± {catboost_cv_scores.std():.2f}')

# Add file upload widgets
df = pd.read_csv('Data/train_data.csv')
test_features = pd.read_csv('Data/test_data.csv')

# Prepare data
print("Creating features...")
train_df = create_advanced_features(df)
test_df = create_advanced_features(test_features)

# Model preparation
drop_columns = ['ID', 'radiant_win']
feature_cols = [col for col in train_df.columns if col not in drop_columns]

X = train_df[feature_cols]
y = train_df['radiant_win']

# Cross validation setup
n_folds = 5
skf = StratifiedKFold(n_splits=n_folds, shuffle=True, random_state=42)

# Initialize arrays for predictions
oof_lgb = np.zeros(len(X))
oof_xgb = np.zeros(len(X))
oof_cat = np.zeros(len(X))
predictions_lgb = np.zeros(len(test_df))
predictions_xgb = np.zeros(len(test_df))
predictions_cat = np.zeros(len(test_df))

# Model parameters
lgb_params = {
    'n_estimators': 2000,
    'learning_rate': 0.01,
    'num_leaves': 63,
    'max_depth': 7,
    'min_child_samples': 20,
    'subsample': 0.8,
    'colsample_bytree': 0.8,
    'random_state': 42
}

xgb_params = {
    'n_estimators': 2000,
    'learning_rate': 0.01,
    'max_depth': 7,
    'min_child_weight': 3,
    'subsample': 0.8,
    'colsample_bytree': 0.8,
    'random_state': 42
}

cat_params = {
    'iterations': 2000,
    'learning_rate': 0.01,
    'depth': 7,
    'l2_leaf_reg': 3,
    'random_seed': 42,
    'verbose': False
}

print("Training models with cross-validation...")
# Cross-validation training
for fold, (train_idx, val_idx) in enumerate(skf.split(X, y)):
    print(f'Fold {fold + 1}')
    X_train, X_val = X.iloc[train_idx], X.iloc[val_idx]
    y_train, y_val = y.iloc[train_idx], y.iloc[val_idx]

    # Scale features
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
```

```python
    X_val_scaled = scaler.transform(X_val)
    X_test_scaled = scaler.transform(test_df[feature_cols])

    # LightGBM
    lgb_model = LGBMClassifier(**lgb_params)
    lgb_model.fit(X_train_scaled, y_train)
    oof_lgb[val_idx] = lgb_model.predict_proba(X_val_scaled)[:, 1]
    predictions_lgb += lgb_model.predict_proba(X_test_scaled)[:, 1] / n_folds

    # XGBoost
    xgb_model = XGBClassifier(**xgb_params)
    xgb_model.fit(X_train_scaled, y_train)
    oof_xgb[val_idx] = xgb_model.predict_proba(X_val_scaled)[:, 1]
    predictions_xgb += xgb_model.predict_proba(X_test_scaled)[:, 1] / n_folds

    # CatBoost
    cat_model = CatBoostClassifier(**cat_params)
    cat_model.fit(X_train_scaled, y_train)
    oof_cat[val_idx] = cat_model.predict_proba(X_val_scaled)[:, 1]
    predictions_cat += cat_model.predict_proba(X_test_scaled)[:, 1] / n_folds

# Evaluate individual models
print("\nCross-validation scores:")
print(f"LightGBM CV score: {roc_auc_score(y, oof_lgb):.4f}")
print(f"XGBoost CV score: {roc_auc_score(y, oof_xgb):.4f}")
print(f"CatBoost CV score: {roc_auc_score(y, oof_cat):.4f}")
```

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

Cross-validation scores:
LightGBM CV score: 0.8207
XGBoost CV score: 0.8229
CatBoost CV score: 0.8230

Optimal weights: [0.33333333 0.33333333 0.33333333]

Final ensemble CV score: 0.8237

Submission file created!
```

```python
# Find optimal weights
from scipy.optimize import minimize

def objective(weights):
    oof_weighted = weights[0] * oof_lgb + weights[1] * oof_xgb + weights[2] * oof_cat
    return -roc_auc_score(y, oof_weighted)

constraints = (
    {'type': 'eq', 'fun': lambda w: np.sum(w) - 1},  # weights sum to 1
    {'type': 'ineq', 'fun': lambda w: w},  # weights are non-negative
)

initial_weights = [1/3, 1/3, 1/3]
bounds = [(0, 1), (0, 1), (0, 1)]
result = minimize(objective, initial_weights, method='SLSQP', bounds=bounds, constraints=constraints)
optimal_weights = result.x

print("\nOptimal weights:", optimal_weights)

# Final ensemble predictions
oof_weighted = (
    optimal_weights[0] * oof_lgb +
    optimal_weights[1] * oof_xgb +
    optimal_weights[2] * oof_cat
)

print(f"\nFinal ensemble CV score: {roc_auc_score(y, oof_weighted):.4f}")

# Create submission
test_pred = (
    optimal_weights[0] * predictions_lgb +
    optimal_weights[1] * predictions_xgb +
    optimal_weights[2] * predictions_cat
```

```
)

submission = pd.DataFrame({
    'ID': test_df['ID'],
    'radiant_win': test_pred
})
submission.to_csv('optimized_ensemble_submission.csv', index=False)
print("\nSubmission file created!")
```