```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
from imblearn.over_sampling import SMOTE
from google.colab import drive

# Mount Google Drive
drive.mount('/content/drive')

# Provide the path to your dataset in Google Drive
file_path = '/content/drive/MyDrive/booking.csv'

# Load the dataset
hotel_data = pd.read_csv(file_path)
print("Data Preview:\n", hotel_data.head())

# Data Exploration: Check for missing values and dataset overview
print("Missing Values Summary:\n", hotel_data.isnull().sum())
print("Dataset shape:", hotel_data.shape)

# Feature Engineering
# Convert 'date of reservation' to datetime format and extract useful features
hotel_data['date of reservation'] = pd.to_datetime(hotel_data['date of reservation'], errors='coerce')
hotel_data['reservation_month'] = hotel_data['date of reservation'].dt.month
hotel_data['reservation_year'] = hotel_data['date of reservation'].dt.year

# Convert 'booking status' to a binary target variable
hotel_data['cancellation_status'] = hotel_data['booking status'].apply(lambda x: 1 if x == 'Canceled' else 0)

# Encode categorical variables using one-hot encoding
hotel_data_encoded = pd.get_dummies(hotel_data, columns=['type of meal', 'room type', 'market segment type'], drop_first=True)

# Drop unnecessary columns and handle missing values
hotel_data_encoded.drop(columns=['date of reservation', 'booking status', 'Booking_ID'], inplace=True)
hotel_data_encoded = hotel_data_encoded.dropna()

# Separate features (X) and target (Y)
X = hotel_data_encoded.drop(['cancellation_status'], axis=1)
Y = hotel_data_encoded['cancellation_status']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Handle class imbalance using SMOTE
smote = SMOTE(random_state=42)
X_train_balanced, y_train_balanced = smote.fit_resample(X_train_scaled, y_train)

# Hyperparameter tuning using GridSearchCV
param_grid = {
    'n_neighbors': range(3, 11),
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan']
}

grid_search = GridSearchCV(KNeighborsClassifier(), param_grid, cv=3, scoring='f1', verbose=1)
grid_search.fit(X_train_balanced, y_train_balanced)

# Get the best K-NN model
best_knn_model = grid_search.best_estimator_
print("Best parameters:", grid_search.best_params_)

# Fit the best model
best_knn_model.fit(X_train_balanced, y_train_balanced)

# Predict on the test set
knn_y_pred = best_knn_model.predict(X_test_scaled)

# Evaluate the model
knn_metrics = {
    "Accuracy": accuracy_score(y_test, knn_y_pred),
    "Precision": precision_score(y_test, knn_y_pred),
    "Recall": recall_score(y_test, knn_y_pred),
    "F1 Score": f1_score(y_test, knn_y_pred),
```

```
}
print("Optimized K-Nearest Neighbors Metrics:", knn_metrics)

# Confusion Matrix
print("Confusion Matrix:\n", confusion_matrix(y_test, knn_y_pred))

# Cross-validation scores
cv_scores = cross_val_score(best_knn_model, X_train_balanced, y_train_balanced, cv=5, scoring='f1')
print("Cross-validated F1 scores:", cv_scores)
print("Average F1 score:", cv_scores.mean())
```

Show hidden output