```
from google.colab import files
uploaded = files.upload()
```

```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, roc_auc_score
import numpy as np

# Load datasets with consistent dtypes and low_memory=False to avoid DtypeWarning
app_df = pd.read_csv('app.csv', dtype={'identity_id_hash': 'object'}, low_memory=False)
campa_df = pd.read_csv('campa.csv', dtype={'identity_id_hash': 'object'}, low_memory=False)
ib_df = pd.read_csv('ib.csv', dtype={'identity_id_hash': 'object'}, low_memory=False)
prodeje_df = pd.read_csv('prodeje.csv', dtype={'identity_id_hash': 'object'}, low_memory=False)
web_df = pd.read_csv('web.csv', dtype={'identity_id_hash': 'object'}, low_memory=False)

# Merge datasets on 'identity_id_hash'
merged_df = app_df.merge(campa_df, on='identity_id_hash', how='outer', suffixes=('_app', '_campa')) \
                  .merge(ib_df, on='identity_id_hash', how='outer', suffixes=('', '_ib')) \
                  .merge(prodeje_df, on='identity_id_hash', how='outer', suffixes=('', '_prodeje')) \
                  .merge(web_df, on='identity_id_hash', how='outer', suffixes=('', '_web'))

# Drop columns with all missing values
columns_to_drop = [col for col in merged_df.columns if merged_df[col].isnull().sum() == len(merged_df)]
cleaned_df = merged_df.drop(columns=columns_to_drop)

# Fill missing values for categorical and numerical columns
categorical_cols = cleaned_df.select_dtypes(include=['object']).columns
numerical_cols = cleaned_df.select_dtypes(include=['float64', 'int64']).columns
cleaned_df[categorical_cols] = cleaned_df[categorical_cols].fillna('Unknown')
cleaned_df[numerical_cols] = cleaned_df[numerical_cols].fillna(cleaned_df[numerical_cols].median())

# Feature engineering
extra_features = pd.DataFrame({
    'avg_session_duration': cleaned_df[['session_time', 'session_time_ib']].mean(axis=1),
    'interaction_count': cleaned_df.apply(lambda row: sum([1 for event in ['page_view', 'custom_event', 'screen_view', 'back_intent_event', 'show_content_event'] if event in row.
    'campaign_engagement': cleaned_df['campaign_planning_name'].apply(lambda x: 0 if x == 'Unknown' else 1),
    'unique_product_interest': cleaned_df[['product_l1', 'product_l2', 'product_l2_prodeje']].nunique(axis=1)
})

cleaned_df = pd.concat([cleaned_df, extra_features], axis=1)

# Define target variable
cleaned_df['purchase'] = cleaned_df['agreement_status'].apply(lambda x: 1 if x == 'Y' else 0)
# cleaned_df['purchase'] = cleaned_df['nbi_fictive_flag'].apply(lambda x: 1 if x == '>2100' else 0)

# Select only numeric columns for correlation to avoid ValueError
numeric_df = cleaned_df.select_dtypes(include=[np.number])

```

```
# Data leakage check by examining correlation with the target variable
correlations = numeric_df.corr()
high_corr_features = correlations['purchase'].loc[lambda x: abs(x) > 0.8].drop('purchase').index.tolist()

 if high_corr_features:
    print("Potential data leakage detected with high correlation features:", high_corr_features)
 else:
    print("No high correlation features detected, data leakage is less likely.")

# Select features and target, excluding highly correlated features if necessary
feature_columns = ['avg_session_duration', 'interaction_count', 'campaign_engagement', 'unique_product_interest']
if high_corr_features:
    feature_columns = [col for col in feature_columns if col not in high_corr_features]

X = cleaned_df[feature_columns]
y = cleaned_df['purchase']

# Split data into train and test sets with an 80-20 split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Set up RandomForest with GridSearchCV for hyperparameter tuning
rf_model = RandomForestClassifier(random_state=42)
param_grid = {
    'n_estimators': [50],
    'max_depth': [5],
    'min_samples_split': [2]
}
grid_search = GridSearchCV(rf_model, param_grid, cv=5, scoring='roc_auc', n_jobs=-1)

# Fit model with GridSearch
grid_search.fit(X_train, y_train)

# Best model and cross-validation scores
best_model = grid_search.best_estimator_
cross_val_scores = cross_val_score(best_model, X_train, y_train, cv=5, scoring='roc_auc')

# Make predictions with the best model on the test set
y_pred = best_model.predict(X_test)
y_prob = best_model.predict_proba(X_test)[:, 1]

# Evaluation metrics
classification_rep = classification_report(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_prob)

# Output results
print("Best Parameters:", grid_search.best_params_)
print("Cross-validation AUC scores:", cross_val_scores)
print("Average CV AUC:", cross_val_scores.mean())
print("Test Set Classification Report:\n", classification_rep)
print("Test Set AUC-ROC:", roc_auc)

```

```
# Segment customers based on predicted purchase probabilities
customer_probabilities = best_model.predict_proba(X_test)[:, 1]  # Probability of being in class 1 (purchase)

# Define segments based on probability thresholds
segments = []
for prob in customer_probabilities:
    if prob > 0.8:
        segments.append('High Value Engaged')
    elif 0.5 < prob <= 0.8:
        segments.append('Potential Upsell')
    elif 0.3 < prob <= 0.5:
        segments.append('Nurture and Educate')
    else:
        segments.append('Low Engagement and Awareness')

# Add segment labels and probability to DataFrame
X_test['purchase_probability'] = customer_probabilities
X_test['segment'] = segments

# Product recommendations based on segments
recommendations = []
for segment in segments:
    if segment == 'High Value Engaged':
        recommendations.append('Offer premium services, investment opportunities, and loyalty programs.')
    elif segment == 'Potential Upsell':
        recommendations.append('Suggest additional products like credit cards or loans with personalized offers.')
    elif segment == 'Nurture and Educate':
        recommendations.append('Provide educational content on products, focusing on convenience and value.')
    else:  # 'Low Engagement and Awareness'
        recommendations.append('Target with brand awareness campaigns and simple introductory offers.')

# Add recommendations to the DataFrame
X_test['recommendation'] = recommendations

# Display a sample of the segmented data with recommendations
X_test[['purchase_probability', 'segment', 'recommendation']].head(10)
```

```
Best Parameters: {'max_depth': 5, 'min_samples_split': 2, 'n_estimators': 50}
Cross-validation AUC scores: [0.95188881 0.94734026 0.94265207 0.95369071 0.95975599]
Average CV AUC: 0.951065568125418
Test Set Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.88      0.91       695
           1       0.81      0.92      0.86       391

    accuracy                           0.89      1086
   macro avg       0.88      0.90      0.89      1086
weighted avg       0.90      0.89      0.89      1086

Test Set AUC-ROC: 0.9418498960422454
```

| | purchase_probability | segment | recommendation |
|---|---|---|---|
| **79** | 0.777550 | Potential Upsell | Suggest additional products like credit cards ... |
| **4139** | 0.909583 | High Value Engaged | Offer premium services, investment opportuniti... |
| **1643** | 0.005575 | Low Engagement and Awareness | Target with brand awareness campaigns and simp... |
| **167** | 0.838086 | High Value Engaged | Offer premium services, investment opportuniti... |
| **439** | 0.777550 | Potential Upsell | Suggest additional products like credit cards ... |
| **2531** | 0.065151 | Low Engagement and Awareness | Target with brand awareness campaigns and simp... |
| **1421** | 0.005575 | Low Engagement and Awareness | Target with brand awareness campaigns and simp... |
| **2834** | 0.065151 | Low Engagement and Awareness | Target with brand awareness campaigns and simp... |
| **4356** | 0.777550 | Potential Upsell | Suggest additional products like credit cards ... |
| **168** | 0.838086 | High Value Engaged | Offer premium services, investment opportuniti... |