```
import pandas as pd

df = pd.read_csv("appointments.csv")
df.head()

import pandas as pd
import numpy as np

# Load the dataset
df = pd.read_csv("appointments.csv")

# Print out the exact column names to inspect them
print("Original Column Names:", df.columns)

# Normalize column names to lowercase and replace spaces with underscores
df.columns = df.columns.str.lower().str.replace(' ', '_').str.replace('-', '_')

# Check the column names again
print("Normalized Column Names:", df.columns)

# Ensure that the column names are consistent
# If there are any mismatches, manually adjust the names here, e.g.:
# Check if 'scheduled_day' exists, if not, find the correct name and rename it
if 'scheduled_day' not in df.columns:
    # Assuming the original column name is 'ScheduledDay' (adjust if needed)
    # Find the closest match using difflib
    import difflib
    closest_match = difflib.get_close_matches('scheduled_day', df.columns, n=1, cutoff=0.8)

    if closest_match:
        df.rename(columns={closest_match[0]: 'scheduled_day'}, inplace=True)
    else:
        # If no close match is found, raise an error or handle it appropriately
        raise KeyError(f"Could not find a column name similar to 'scheduled_day' in the DataFrame. Available columns: {df.columns

# Do the same for 'appointment_day'
if 'appointment_day' not in df.columns:
    closest_match = difflib.get_close_matches('appointment_day', df.columns, n=1, cutoff=0.8)
    if closest_match:
        df.rename(columns={closest_match[0]: 'appointment_day'}, inplace=True)
    else:
        raise KeyError(f"Could not find a column name similar to 'appointment_day' in the DataFrame. Available columns: {df.colum


# Convert date columns to datetime format
df['scheduled_day'] = pd.to_datetime(df['scheduled_day'], errors='coerce')
df['appointment_day'] = pd.to_datetime(df['appointment_day'], errors='coerce')

# ... (rest of your code) ...
# Remove rows with invalid date values
df = df.dropna(subset=['scheduled_day', 'appointment_day'])

# Clean the 'age' column (remove invalid values)
df = df[(df['age'] >= 0) & (df['age'] < 120)]

# Map the 'no_show' column to binary values (0 for No, 1 for Yes)
df['no_show'] = df['no_show'].map({'No': 0, 'Yes': 1})

# Feature Engineering
df['days_waiting'] = (df['appointment_day'] - df['scheduled_day']).dt.days
df['appointment_weekday'] = df['appointment_day'].dt.day_name()
df['appointment_hour'] = df['appointment_day'].dt.hour

df['time_block'] = pd.cut(df['appointment_hour'], bins=[0, 12, 16, 24],
                          labels=['Morning', 'Afternoon', 'Evening'], right=False)

df['age_group'] = pd.cut(df['age'], bins=[0, 18, 40, 60, 100],
                         labels=['Child', 'Adult', 'MiddleAge', 'Senior'])

# Simulate external factor (bad weather)
np.random.seed(42)
df['bad_weather'] = np.random.choice([0, 1], size=len(df), p=[0.85, 0.15])

# Check the first few rows of the cleaned DataFrame
df.head()

print(df.head())
```

```
print(df.head())

from sklearn.model_selection import train_test_split

# Features and target
features = ['age', 'days_waiting', 'sms_received', 'scholarship', 'hipertension',
            'diabetes', 'alcoholism', 'handcap', 'bad_weather', 'appointment_weekday',
            'time_block', 'age_group']

# Encoding categorical variables into dummy variables
df_model = pd.get_dummies(df[features], drop_first=True)

X = df_model
y = df['no_show']

# 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)

print(f"Training set size: {len(X_train)}")
print(f"Test set size: {len(X_test)}")

from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix

# Initialize the RandomForestClassifier
model = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the model
model.fit(X_train, y_train)

# Predict on the test data
y_pred = model.predict(X_test)

# Evaluate the model's performance
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Confusion Matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

# Get feature importance from the trained model
feature_importances = model.feature_importances_

# Create a DataFrame to display feature importances
importance_df = pd.DataFrame({
    'feature': X.columns,
    'importance': feature_importances
}).sort_values(by='importance', ascending=False)

print(importance_df)

# Example function to generate recommendations for high-risk patients (predicted no-shows)
def suggest_actions(row):
    recommendations = []

    if row['sms_received'] == 0:
        recommendations.append("Send SMS reminder")
    if row['days_waiting'] > 7:
        recommendations.append("Consider rescheduling closer to appointment date")
    if row['bad_weather'] == 1:
        recommendations.append("Offer teleconsultation due to bad weather")
    if row['appointment_weekday'] in ['Monday', 'Friday']:
        recommendations.append("Mid-week appointments may reduce no-shows")

    return "; ".join(recommendations)

# Filter the high-risk patients (those predicted to not show up)
df_test = df.iloc[y_test.index].copy()
df_test['predicted_no_show'] = y_pred
high_risk_patients = df_test[df_test['predicted_no_show'] == 1]

# Apply the recommendation function
high_risk_patients['recommendations'] = high_risk_patients.apply(suggest_actions, axis=1)

# Save the high-risk patients with recommendations to a CSV file
high_risk_patients[['age', 'appointment_day', 'predicted_no_show', 'recommendations']].to_csv('high_risk_patients_with_recommenda
```

```
print(high_risk_patients.head())

# Export full cleaned dataset
df.to_csv("cleaned_appointment_data.csv", index=False)

# Export the prediction + recommendations
high_risk_patients[['age', 'appointment_day', 'predicted_no_show', 'recommendations']].to_csv(
    "high_risk_patients_with_recommendations.csv", index=False)
```

```
Original Column Names: Index(['PatientId', 'AppointmentID', 'Gender', 'ScheduledDay',
       'AppointmentDay', 'Age', 'Neighbourhood', 'Scholarship', 'Hipertension',
       'Diabetes', 'Alcoholism', 'Handcap', 'SMS_received', 'No-show'],
      dtype='object')
Normalized Column Names: Index(['patientid', 'appointmentid', 'gender', 'scheduledday',
       'appointmentday', 'age', 'neighbourhood', 'scholarship', 'hipertension',
       'diabetes', 'alcoholism', 'handcap', 'sms_received', 'no_show'],
      dtype='object')
      patientid  appointmentid gender             scheduled_day  \
0  2.987250e+13        5642903      F 2016-04-29 18:38:08+00:00
1  5.589980e+14        5642503      M 2016-04-29 16:08:27+00:00
2  4.262960e+12        5642549      F 2016-04-29 16:19:04+00:00
3  8.679510e+11        5642828      F 2016-04-29 17:29:31+00:00
4  8.841190e+12        5642494      F 2016-04-29 16:07:23+00:00

            appointment_day  age      neighbourhood  scholarship  \
0 2016-04-29 00:00:00+00:00   62    JARDIM DA PENHA            0
1 2016-04-29 00:00:00+00:00   56    JARDIM DA PENHA            0
2 2016-04-29 00:00:00+00:00   62      MATA DA PRAIA            0
3 2016-04-29 00:00:00+00:00    8  PONTAL DE CAMBURI            0
4 2016-04-29 00:00:00+00:00   56    JARDIM DA PENHA            0

   hipertension  diabetes  alcoholism  handcap  sms_received  no_show  \
0             1         0           0        0             0        0
1             0         0           0        0             0        0
2             0         0           0        0             0        0
3             0         0           0        0             0        0
4             1         1           0        0             0        0

   days_waiting appointment_weekday  appointment_hour time_block  age_group  \
0            -1              Friday                 0    Morning     Senior
1            -1              Friday                 0    Morning  MiddleAge
2            -1              Friday                 0    Morning     Senior
3            -1              Friday                 0    Morning      Child
4            -1              Friday                 0    Morning  MiddleAge

   bad_weather
0            0
1            1
2            0
3            0
4            0
Training set size: 88420
Test set size: 22106
Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.89      0.86     17715
           1       0.34      0.22      0.27      4391

    accuracy                           0.76     22106
   macro avg       0.58      0.56      0.56     22106
weighted avg       0.73      0.76      0.74     22106

Confusion Matrix:
[[15843  1872]
 [ 3439   952]]
```