```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import shap

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from xgboost import XGBClassifier
from sklearn.metrics import classification_report, confusion_matrix

# Step 1: Load CSV file
# Add the 'encoding' parameter to specify the correct encoding
file_path = "customer_churn.csv" # Define the file path

try:
    df = pd.read_csv(file_path, encoding='latin-1')  # Replace with your actual file path
    print(f"Successfully loaded {file_path}")
    print("Initial DataFrame shape:", df.shape)
    print("Initial DataFrame columns:", df.columns.tolist())

    # Check if the 'Churn' column exists immediately after loading
    if 'Churn' not in df.columns:
        raise KeyError(f"Column 'Churn' not found in the file {file_path}. Please check the file.")

except FileNotFoundError:
    print(f"Error: The file '{file_path}' was not found. Please ensure the file is in the correct directory or provide the full path.")
    exit() # Exit the script if the file is not found
except Exception as e:
    print(f"An error occurred while reading the file: {e}")
    print(f"The file content might not be a valid CSV. Please check the content of '{file_path}'.")
    exit() # Exit for other reading errors


# Step 2: Handle missing values (basic dropna, or customize)
initial_rows = df.shape[0]
df = df.dropna()
rows_after_dropna = df.shape[0]
print(f"\nRows before dropna: {initial_rows}, Rows after dropna: {rows_after_dropna}")

# Step 3: Encode categorical variables
label_encoders = {}
# Re-check columns after dropna
object_cols = df.select_dtypes(include=['object']).columns
print(f"\nObject columns before encoding: {object_cols.tolist()}")

for col in object_cols:
    # Check if column exists before encoding, as dropna might remove columns if all values are NaN
    if col in df.columns and col != 'Churn':  # Do not encode target here
        le = LabelEncoder()
        try:
            df[col] = le.fit_transform(df[col].astype(str)) # Ensure column is treated as string
            label_encoders[col] = le
        except Exception as e:
            print(f"Error encoding column {col}: {e}")
            # You might want to investigate the specific values in this column

# Step 4: Encode the target column (Churn)
print(f"\nEncoding target column: 'Churn'")
# Check if 'Churn' still exists after dropna and other potential transformations
if 'Churn' in df.columns:
    try:
        target_encoder = LabelEncoder()
        df['Churn'] = target_encoder.fit_transform(df['Churn'].astype(str)) # Ensure target is treated as string
        print("Target column 'Churn' encoded successfully.")
    except Exception as e:
        print(f"Error encoding target column 'Churn': {e}")
        # This error is less likely if the column exists and is not empty
else:
    print("Error: 'Churn' column not found after cleaning/processing steps. Cannot encode target.")
    # Exit or handle the case where Churn is missing after processing
    exit()


# Step 5: Define features and target
# *** Add this line to check columns before dropping ***
print("\nDataFrame columns before dropping 'Churn':", df.columns.tolist())

# --- Add a check here before dropping 'Churn' ---
if 'Churn' not in df.columns:
    print("Error: 'Churn' column is missing from the DataFrame before defining features and target. This may be due to dropna() removing it."
    exit() # Exit the script as 'Churn' is essential for the analysis
# --- End of added check ---

X = df.drop("Churn", axis=1)
y = df["Churn"]
print("\nDefined features (X) and target (y)")
print("X shape:", X.shape)
print("y shape:", y.shape)


# Step 6: Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
print(f"\nTrain-test split complete. Train shapes: {X_train.shape}, {y_train.shape}. Test shapes: {X_test.shape}, {y_test.shape}")

# Step 7: Scale features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
print("\nFeatures scaled successfully.")


# Step 8: Train XGBoost classifier
# Note: use_label_encoder is deprecated and removed in XGBoost 1.6+.
# It's recommended to handle label encoding before training.
# The code already does this for the target variable 'Churn'.
model = XGBClassifier(eval_metric='logloss')
model.fit(X_train_scaled, y_train)
print("\nXGBoost model trained successfully.")

# Step 9: Make predictions
y_pred = model.predict(X_test_scaled)
print("\nPredictions made successfully.")

# Step 10: Evaluate the model
print("\n=== Confusion Matrix ===")
print(confusion_matrix(y_test, y_pred))

print("\n=== Classification Report ===")
print(classification_report(y_test, y_pred))

# Step 11: Feature importance plot
# Use original column names for feature importance plotting
feature_importances = pd.DataFrame({
    'Feature': X.columns, # Use X.columns here
    'Importance': model.feature_importances_
}).sort_values(by="Importance", ascending=False)

plt.figure(figsize=(10, 6))
sns.barplot(data=feature_importances, x="Importance", y="Feature", palette="viridis")
plt.title("Feature Importance")
plt.tight_layout()
plt.show()

# Step 12: SHAP values for uncovering hidden patterns
# SHAP explainer needs a model that predicts probabilities or log-odds.
# For tree-based models like XGBoost, TreeExplainer is usually more efficient.
# explainer = shap.Explainer(model, X_train_scaled) # This might be slow
explainer = shap.TreeExplainer(model) # Use TreeExplainer for tree models
# Ensure consistent feature names/order if possible, but X_test_scaled is needed for the explainer
shap_values = explainer.shap_values(X_test_scaled)
print("\nSHAP values calculated successfully.")


# SHAP summary plot
# The first argument of shap.summary_plot should be the SHAP values array,
# and the second should be the corresponding feature data.
# Make sure X_test is used for plotting if you want original feature names.
# Ensure X_test has the same column order as the trained model's features if possible.
# Using X_test directly with the scaled data's shap values is common for plotting.
shap.summary_plot(shap_values, X_test, plot_type="bar")
shap.summary_plot(shap_values, X_test, plot_type="bar")
```

```
Successfully loaded customer_churn.csv
Initial DataFrame shape: (1880, 8)
Initial DataFrame columns: ['%PDF-1.7', 'Unnamed: 1', 'Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4', 'Unnamed: 5', 'Unnamed: 6', 'Unnamed: 7'
An error occurred while reading the file: "Column 'Churn' not found in the file customer_churn.csv. Please check the file."
The file content might not be a valid CSV. Please check the content of 'customer_churn.csv'.

Rows before dropna: 1880, Rows after dropna: 1

Object columns before encoding: ['%PDF-1.7', 'Unnamed: 1', 'Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4', 'Unnamed: 5', 'Unnamed: 6', 'Unname

Encoding target column: 'Churn'
Error: 'Churn' column not found after cleaning/processing steps. Cannot encode target.

DataFrame columns before dropping 'Churn': ['%PDF-1.7', 'Unnamed: 1', 'Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4', 'Unnamed: 5', 'Unnamed:
Error: 'Churn' column is missing from the DataFrame before defining features and target. This may be due to dropna() removing it.
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-1-dfc6bc81786f> in <cell line: 0>()
     84 # --- End of added check ---
     85 
---> 86 X = df.drop("Churn", axis=1)
     87 y = df["Churn"]
     88 print("\nDefined features (X) and target (y)")
```

3 frames

```
/usr/local/lib/python3.11/dist-packages/pandas/core/indexes/base.py in drop(self, labels, errors)
   7068         if mask.any():
   7069             if errors != "ignore":
-> 7070                 raise KeyError(f"{labels[mask].tolist()} not found in axis")
   7071             indexer = indexer[~mask]
   7072         return self.delete(indexer)

KeyError: "['Churn'] not found in axis"
```

Next steps: Explain error