```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, f1_score, roc_auc_score, classification_report

pd.set_option('display.max_columns', None)
pd.set_option('display.width', 1000)

DATA_FILE = "DATA SCEINCE FINALE DATA.csv"

def load_and_preprocess_data(file_path):
    df_loaded = None
    try:
        df_loaded = pd.read_csv(file_path, header=None)

        header_row_index = -1
        for i in range(min(df_loaded.shape[0], 10)):
            row_values = [str(val) for val in df_loaded.iloc[i].values]
            if 'SNo' in row_values and 'Name' in row_values:
                header_row_index = i
                break

        if header_row_index != -1:
            df_loaded.columns = df_loaded.iloc[header_row_index]
            df_loaded = df_loaded.iloc[header_row_index + 1:].reset_index(drop=True)
        else:
            raise ValueError("Header identification failed.")

    except FileNotFoundError:
        df_loaded = None
    except Exception:
        df_loaded = None

    if df_loaded is None:
        return None

    if 'SNo' in df_loaded.columns:
        df_loaded = df_loaded.drop('SNo', axis=1)

    if 'Date' in df_loaded.columns:
        df_loaded['Date'] = pd.to_datetime(df_loaded['Date'], errors='coerce')

    cols_to_convert_numeric = ['High', 'Low', 'Open', 'Close', 'Volume', 'Marketcap']
    for col in cols_to_convert_numeric:
        if col in df_loaded.columns:
            df_loaded[col] = pd.to_numeric(df_loaded[col], errors='coerce')

    if df_loaded.isnull().sum().sum() > 0:
        for col in cols_to_convert_numeric:
            if col in df_loaded.columns and df_loaded[col].isnull().any():
                df_loaded[col].fillna(df_loaded[col].median(), inplace=True)
        if 'Date' in df_loaded.columns:
            df_loaded.dropna(subset=['Date'], inplace=True)

    df_loaded.drop_duplicates(inplace=True)
    if 'Name' in df_loaded.columns and 'Date' in df_loaded.columns:
        df_loaded = df_loaded.sort_values(by=['Name', 'Date']).reset_index(drop=True)

    if 'Close' in df_loaded.columns and 'Open' in df_loaded.columns:
        df_loaded['Daily_Price_Change'] = df_loaded['Close'] - df_loaded['Open']
        df_loaded['Daily_Return'] = (df_loaded['Close'] / df_loaded['Open'].replace(0, np.nan) - 1) * 100

    return df_loaded

def create_ml_features(df):
    df = df[df['Open'] != 0].copy()
    df['Price_Direction'] = (df['Close'] > df['Open']).astype(int)

    features = ['Close', 'High', 'Low', 'Volume', 'Marketcap', 'Daily_Return']
    lagged_features = []

    for feature in features:
        df[f'{feature}_lag1'] = df.groupby('Name')[feature].shift(1)
        lagged_features.append(f'{feature}_lag1')

    df.dropna(inplace=True)

    X = df[lagged_features]
    y = df['Price_Direction']

    if len(X) != len(y):
        return None, None

    return X, y

def train_and_evaluate_models(X_train, X_test, y_train, y_test):
    models = {
        'Random Forest': RandomForestClassifier(n_estimators=100, random_state=42),
        'SVM': SVC(random_state=42, probability=True),
        'KNN': KNeighborsClassifier(n_neighbors=5)
    }

    results = {}

    for name, model in models.items():
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)

        accuracy = accuracy_score(y_test, y_pred)
        f1 = f1_score(y_test, y_pred, average='weighted')

        roc_auc = "N/A"
        if hasattr(model, "predict_proba"):
            y_pred_proba = model.predict_proba(X_test)
            try:
                roc_auc = roc_auc_score(y_test, y_pred_proba[:, 1])
            except ValueError:
                pass

        results[name] = {'Accuracy': accuracy, 'F1-Score': f1, 'AUC-ROC': roc_auc}

        print(f"--- {name} Results ---")
        print(f"Accuracy: {accuracy:.4f}")
        print(f"F1-Score: {f1:.4f}")
        print(f"AUC-ROC: {roc_auc if isinstance(roc_auc, float) else roc_auc}")
        print("\nClassification Report:")
        print(classification_report(y_test, y_pred))

    return results

if __name__ == "__main__":
    crypto_df = load_and_preprocess_data(DATA_FILE)

    if crypto_df is not None:
        X, y = create_ml_features(crypto_df)

        if X is not None and y is not None:
            split_point = int(len(X) * 0.8)
            X_train, X_test = X[:split_point], X[split_point:]
            y_train, y_test = y[:split_point], y[split_point:]

            scaler = StandardScaler()
            X_train_scaled = scaler.fit_transform(X_train)
            X_test_scaled = scaler.transform(X_test)

            model_results = train_and_evaluate_models(X_train_scaled, X_test_scaled, y_train, y_test)

            print("\n--- Final Model Comparison Summary ---")
            for name, metrics in model_results.items():
                print(f"Model: {name}")
                for metric_name, value in metrics.items():
                    print(f"  {metric_name}: {value:.4f}" if isinstance(value, float) else f"  {metric_name}
        else:
            print("\nML feature creation failed. Halting model comparison.")
    else:
        print("\nData loading and preprocessing failed. Halting model comparison.")
```

```
--- Random Forest Results ---
Accuracy: 0.5383
F1-Score: 0.5324
AUC-ROC: 0.5159588877333584

Classification Report:
              precision    recall  f1-score   support

           0       0.56      0.65      0.60      4001
           1       0.50      0.41      0.45      3411

    accuracy                           0.54      7412
   macro avg       0.53      0.53      0.53      7412
weighted avg       0.53      0.54      0.53      7412

--- SVM Results ---
Accuracy: 0.5496
F1-Score: 0.5387
AUC-ROC: 0.5637967523657051

Classification Report:
              precision    recall  f1-score   support

           0       0.57      0.69      0.62      4001
           1       0.51      0.38      0.44      3411

    accuracy                           0.55      7412
   macro avg       0.54      0.54      0.53      7412
weighted avg       0.54      0.55      0.54      7412
```

```
--- KNN Results ---
Accuracy: 0.5341
F1-Score: 0.5346
AUC-ROC: 0.5241030331687088

Classification Report:
              precision    recall  f1-score   support

           0       0.57      0.56      0.56      4001
           1       0.49      0.51      0.50      3411

    accuracy                           0.53      7412
   macro avg       0.53      0.53      0.53      7412
weighted avg       0.54      0.53      0.53      7412


--- Final Model Comparison Summary ---
Model: Random Forest
  Accuracy: 0.5383
  F1-Score: 0.5324
  AUC-ROC: 0.5160
Model: SVM
  Accuracy: 0.5496
  F1-Score: 0.5387
  AUC-ROC: 0.5638
Model: KNN
  Accuracy: 0.5341
  F1-Score: 0.5346
```