```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC, LinearSVC
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, roc_auc_score

def preprocess_data(data: pd.DataFrame):
    dummy_cols = ['Location', 'CardType']
    scale_cols = ['Age', 'RiskScore', 'LoyaltyYears', 'AccountBalance',
'IncomeEstimate',
                  'ComplaintSatisfaction', 'CreditCardPoints']
    df = pd.get_dummies(data, columns=dummy_cols)
    scaler = MinMaxScaler()
    df[scale_cols] = scaler.fit_transform(df[scale_cols])
    X = df.drop(['Retention', 'RecordNumber', 'CustomerId', 'LastName'],
axis=1)
    y = df['Retention']
    return train_test_split(X, y, test_size=0.3, random_state=42)


def evaluate_model(model, y_test, prediction, X_test, isSVM=False):
    print("Accuracy:", accuracy_score(y_test, prediction))
    print("Precision:", precision_score(y_test, prediction))
    print("Recall:", recall_score(y_test, prediction))
    print("F1-score:", f1_score(y_test, prediction))
    # if not isSVM:
    print("ROC-AUC:", roc_auc_score(y_test, model.predict_proba(X_test)[:, -
1]), "\n")
    # else:
    #     print("\n")

if __name__ == '__main__':
    # 1-2 Loading and preprocessing
    all_data = pd.read_csv('RetentionBusinessCustomers.csv')
    X_train, X_test, y_train, y_test = preprocess_data(all_data)
    columns = X_train.columns
    # 3.a creating the models
    knn = KNeighborsClassifier()
    dt = DecisionTreeClassifier()
    svm = LinearSVC()
    # 3.b training model and evaluating
    # fitting
    knn.fit(X_train, y_train)
    dt.fit(X_train, y_train)
    svm.fit(X_train, y_train)
    # predict
    knn_preds = knn.predict(X_test)
    dt_preds = dt.predict(X_test)
    svm_preds = svm.predict(X_test)
    # performance
    print("knn evaluation: ")
    evaluate_model(knn, y_test, knn_preds, X_test)
    print("Decision Tree Evaluation: ")
    evaluate_model(dt, y_test, dt_preds, X_test)

    print("SVM Evalutation: ")
    evaluate_model(svm, y_test, svm_preds, X_test, isSVM=True)

    print("The best model was SVM")
    # 4 Model Optimazation
    # fine tuning
    new_svm = LinearSVC()
    param_grid = {'C': [0.001,0.025,0.05, 0.075, 0.1, 0.125, 0.15, 0.25, 1.0,
2.0]}
    grid_search = GridSearchCV(new_svm, param_grid, scoring='accuracy')
    grid_search.fit(X_train, y_train)
    best_params = grid_search.best_params_
    print("Best Param for finetuning are: ", best_params)
    final_svm_model = LinearSVC(**best_params)
    final_svm_model.fit(X_train, y_train)
    final_svm_prediction = final_svm_model.predict(X_test)
    evaluate_model(final_svm_model, y_test, final_svm_prediction, X_test,
isSVM=True)
    # 5 Modle interpretatio
    coefficients = svm.coef_
    absolute_coefficients = abs(coefficients)
    most_significant_features = np.argsort(absolute_coefficients)[::-1]
    print(most_significant_features)
    print(columns)
    print(np.array(columns))
```