```
# lead_scoring.py
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
import joblib

# Sample data loading (in practice, you'd use your actual lead data)
def load_lead_data():
    # Simulating lead data with engagement metrics
    data = {
        'page_views': [10, 5, 20, 3, 15, 8, 25, 2],
        'time_on_site': [15, 5, 30, 2, 20, 10, 40, 1],
        'downloads': [1, 0, 3, 0, 2, 1, 4, 0],
        'email_opens': [3, 1, 5, 0, 4, 2, 6, 0],
        'converted': [1, 0, 1, 0, 1, 0, 1, 0]  # Target
    }
    return pd.DataFrame(data)

def train_lead_scoring_model():
    # Load and prepare data
    df = load_lead_data()
    X = df.drop('converted', axis=1)
    y = df['converted']

    # Split data
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # Train model
    model = RandomForestClassifier(n_estimators=100, random_state=42)
    model.fit(X_train, y_train)

    # Evaluate
    y_pred = model.predict(X_test)
    print(classification_report(y_test, y_pred))

    # Save model
    joblib.dump(model, 'lead_scoring_model.pkl')
    print("Lead scoring model trained and saved.")

    return model

if __name__ == '__main__':
    train_lead_scoring_model()
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Recall is ill-defined and being
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Recall is ill-defined and being
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Recall is ill-defined and being
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
              precision    recall  f1-score   support

           0       1.00      0.50      0.67         2
           1       0.00      0.00      0.00         0

    accuracy                           0.50         2
   macro avg       0.50      0.25      0.33         2
weighted avg       1.00      0.50      0.67         2

Lead scoring model trained and saved.
```

```
# customer_analysis.py
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import classification_report
import joblib

# Sample data loading (in practice, use your actual customer data)
def load_customer_data():
    # Simulating customer data
    data = {
        'usage_frequency': [15, 2, 30, 1, 20, 5, 25, 0],
        'support_tickets': [1, 5, 0, 8, 2, 4, 1, 10],
        'payment_delays': [0, 3, 0, 5, 1, 2, 0, 6],
        'last_engagement': [7, 30, 1, 45, 5, 15, 2, 60],  # days ago
        'churned': [0, 1, 0, 1, 0, 1, 0, 1]  # Target for churn
        # For NBA, you might have different targets like 'upgrade', 'renew', 'support', etc.
    }
    return pd.DataFrame(data)

def train_churn_model():
    # Load and prepare data
    df = load_customer_data()
    X = df.drop('churned', axis=1)
    y = df['churned']

    # Split data
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # Train model
    model = GradientBoostingClassifier(n_estimators=100, random_state=42)
    model.fit(X_train, y_train)

    # Evaluate
    y_pred = model.predict(X_test)
    print(classification_report(y_test, y_pred))

    # Save model
    joblib.dump(model, 'churn_model.pkl')
    print("Churn prediction model trained and saved.")

    return model

if __name__ == '__main__':
    train_churn_model()
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         0
           1       1.00      0.50      0.67         2

    accuracy                           0.50         2
   macro avg       0.50      0.25      0.33         2
weighted avg       1.00      0.50      0.67         2

Churn prediction model trained and saved.
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Recall is ill-defined and being
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Recall is ill-defined and being
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Recall is ill-defined and being
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

```
# app.py
from flask import Flask, request, jsonify
import joblib
import pandas as pd

app = Flask(__name__)

# Load models
lead_model = joblib.load('lead_scoring_model.pkl')
churn_model = joblib.load('churn_model.pkl')

@app.route('/predict_lead', methods=['POST'])
def predict_lead():
    try:
        # Get data from request
        data = request.get_json()

        # Convert to DataFrame
        features = pd.DataFrame([data])

        # Predict
        score = lead_model.predict_proba(features)[0][1]  # Probability of conversion

        return jsonify({
            'lead_score': float(score),
            'conversion_likelihood': 'high' if score > 0.7 else 'medium' if score > 0.4 else 'low'
        })
    except Exception as e:
        return jsonify({'error': str(e)}), 400

@app.route('/predict_churn', methods=['POST'])
def predict_churn():
    try:
        # Get data from request
        data = request.get_json()

        # Convert to DataFrame
        features = pd.DataFrame([data])

        # Predict
        churn_prob = churn_model.predict_proba(features)[0][1]
        nba = "retention_offer" if churn_prob > 0.6 else "check_in" if churn_prob > 0.3 else "upsell"

        return jsonify({
            'churn_probability': float(churn_prob),
            'next_best_action': nba
        })
    except Exception as e:
        return jsonify({'error': str(e)}), 400

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=True)
```

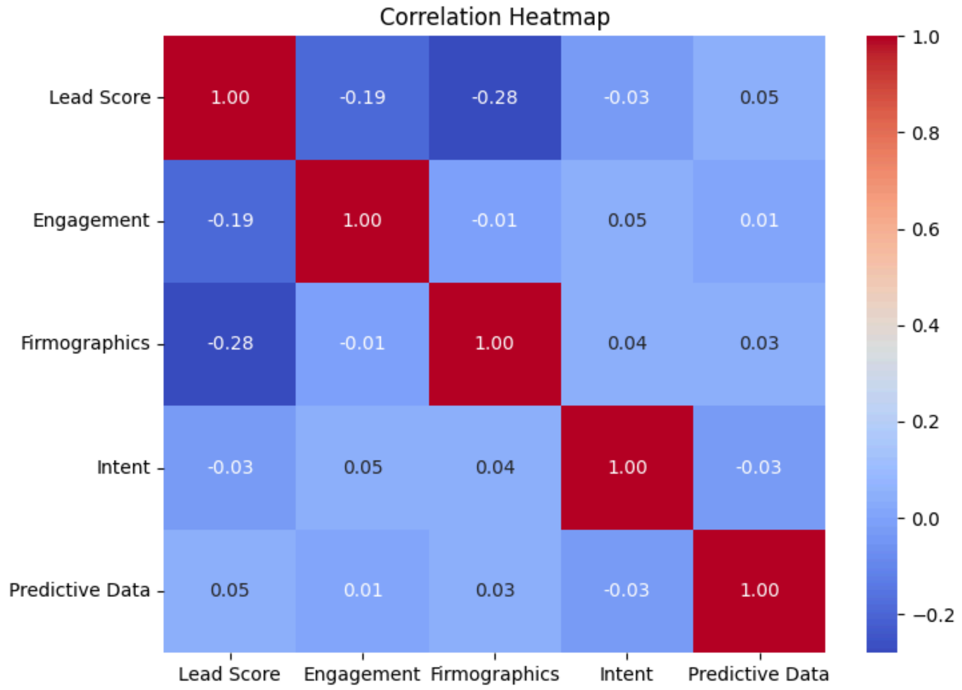
```
 * Serving Flask app '__main__'
 * Debug mode: on
INFO:werkzeug:WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.28.0.12:5000
INFO:werkzeug:Press CTRL+C to quit
INFO:werkzeug: * Restarting with stat
```

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

# Example data
data = {
    'Lead Score': np.random.normal(70, 15, 100),
    'Engagement': np.random.rand(100),
    'Firmographics': np.random.rand(100),
    'Intent': np.random.rand(100),
    'Predictive Data': np.random.rand(100)
}
df = pd.DataFrame(data)

# Correlation matrix
corr = df.corr()

# Plot heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(corr, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap')
plt.show()
```


Correlation Heatmap
Lead Score
Engagement
Firmographics
Intent
Predictive Data
1.00
-0.19
-0.28
-0.03
0.05
-0.19
1.00
-0.01
0.05
0.01
-0.28
-0.01
1.00
0.04
0.03
-0.03
0.05
0.04
1.00
-0.03
0.05
0.01
0.03
-0.03
1.00
Lead Score
Engagement
Firmographics
Intent
Predictive Data
1.0
0.8
0.6
0.4
0.2
0.0
−0.2