SOLUTION DESIGN

Proposed Methodology and Technical Approach

Customer Segmentation Strategy

Segmentation Framework:

- **Risk-based segmentation:** Leverage existing risk tiers (low, medium, high) but refine with behavioral data.
- **Behavioral clusters:** Use K-means clustering on:
  - Payment patterns (days late, amount variability)
  - Channel responsiveness (open/click rates by channel)
  - Reminder frequency tolerance

In [1]:
```
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import numpy as np
```

In [2]:
```
feedback = pd.read_csv(r'C:\Users\gaby\PycharmProjects\Payment-Reminder-Optimizatio\data\raw\feedback.csv')
customers = pd.read_csv(r'C:\Users\gaby\PycharmProjects\Payment-Reminder-Optimizatio\data\raw\customers.csv')
accounts = pd.read_csv(r'C:\Users\gaby\PycharmProjects\Payment-Reminder-Optimizatio\data\raw\accounts.csv')
payments = pd.read_csv(r'C:\Users\gaby\PycharmProjects\Payment-Reminder-Optimizatio\data\raw\payments.csv')
reminders = pd.read_csv(r'C:\Users\gaby\PycharmProjects\Payment-Reminder-Optimizatio\data\raw\reminders.csv')
schedules =pd.read_csv(r'C:\Users\gaby\PycharmProjects\Payment-Reminder-Optimizatio\data\raw\payment_schedules.csv')
```

```
C:\Users\gaby\AppData\Local\Temp\ipykernel_27268\3580896198.py:6: DtypeWarning: Columns (4) have mixed types. Specify
dtype option on import or set low_memory=False.
  schedules =pd.read_csv(r'C:\Users\gaby\PycharmProjects\Payment-Reminder-Optimizatio\data\raw\payment_schedules.cs
v')
```

In [3]:
```
# Enhanced segmentation using available data
def create_customer_segments(customers, payments, reminders):
    # Payment behavior features
    payment_features = payments.groupby('customer_id').agg({
```

```
        'days_late': ['mean', 'max'],
        'amount_paid': 'mean'
    }).reset_index()
    payment_features.columns = ['customer_id', 'avg_days_late', 'max_days_late', 'avg_payment']

    # Reminder responsiveness features
    reminder_features = reminders.groupby('account_id').agg({
        'opened': 'mean',
        'payment_triggered': 'mean'
    }).reset_index()

    # Merge all features
    segments = customers.merge(payment_features, on='customer_id') \
                       .merge(accounts, on='customer_id') \
                       .merge(reminder_features, on='account_id')

    # Simple rule-based segmentation (can be replaced with ML clustering)
    conditions = [
        (segments['risk_tier'] == 'high') & (segments['avg_days_late'] > 7),
        (segments['risk_tier'] == 'medium') & (segments['opened'] < 0.3),
        (segments['payment_triggered'] > 0.5),
        (segments['credit_score'] > 700)
    ]
    choices = ['high_risk_delinquent', 'low_engagement', 'high_response', 'prime']
    segments['segment'] = np.select(conditions, choices, default='standard')

    return segments

customer_segments = create_customer_segments(customers, payments, reminders)
```

---

2. Dynamic Reminder Scheduling Engine

Optimal Timing Model:

We'll use **Bayesian optimization** to determine the ideal reminder timing windows for each customer segment. This model will incorporate:

- **Historical response curves** by time-to-due-date.

- The customer's **preferred channels**.
- Their **payment method** (auto-pay vs. manual).

In [4]:
```
def calculate_optimal_reminders(account_id, segment, due_date):
    # Base rules per segment
    rules = {
        'high_risk_delinquent': {
            'channels': ['sms', 'push'],
            'timing': [-3, 0, 2, 5, 8],
            'max_reminders': 5
        },
        'low_engagement': {
            'channels': ['push', 'email'],
            'timing': [-2, 0, 3],
            'max_reminders': 3
        },
        'high_response': {
            'channels': ['email'],
            'timing': [-1],
            'max_reminders': 1
        },
        'prime': {
            'channels': ['email'],
            'timing': [-3],
            'max_reminders': 1
        },
        'standard': {
            'channels': ['email', 'sms'],
            'timing': [-5, -1],
            'max_reminders': 2
        }
    }

    # Get rules for segment
    segment_rules = rules.get(segment, rules['standard'])

    # Calculate reminder dates
    reminder_dates = []
    for days in segment_rules['timing']:
        reminder_date = due_date + pd.Timedelta(days=days)
        reminder_dates.append({
```

```
            'account_id': account_id,
            'due_date': due_date,
            'planned_date': reminder_date,
            'channel': np.random.choice(segment_rules['channels'])
        })

    return pd.DataFrame(reminder_dates).head(segment_rules['max_reminders'])
```

---

3. Channel Optimization Framework

Multi-armed Bandit Approach:

We will implement **Thompson Sampling** to dynamically optimize channel selection. This approach will balance:

- **Exploration** of new channels.
- **Exploitation** of known effective channels.

In [5]:
```
class ThompsonSampling:
    def __init__(self, channels):
        self.channels = channels
        self.alpha = {ch: 1 for ch in channels}
        self.beta = {ch: 1 for ch in channels}

    def select_channel(self):
        samples = {ch: np.random.beta(self.alpha[ch], self.beta[ch]) for ch in self.channels}
        return max(samples.items(), key=lambda x: x[1])[0]

    def update(self, channel, success):
        if success:
            self.alpha[channel] += 1
        else:
            self.beta[channel] += 1
```

Raw Data → Feature Store → Model Training → Decision Engine → Execution System |

Our model will leverage a comprehensive set of features, including:

- **Payment history metrics:** Mean and standard deviation of days late, and amount variability.

- **Customer demographics:** Risk tier and income bracket.
- **Previous reminder performance:** Open rates and conversion by channel and time.
- **Account characteristics:** Account type, credit limit, and tenure.

We'll use a robust model stack to power the optimization:

- **Payment Probability Predictor (XGBoost/LightGBM):**
  - Predicts the likelihood of on-time payment given reminder parameters.
  - Features: Customer segment, timing, channel, and historical response.
- **Dissatisfaction Predictor (Logistic Regression):**
  - Estimates the probability of negative feedback based on reminder frequency and content.
- **Optimization Engine (Constrained Optimization):**
  - Maximizes: $\Sigma(P(\text{payment}))$
  - Minimizes: $\Sigma(P(\text{dissatisfaction}))$
  - Subject to: Frequency, channel, and budget constraints.

Expected Outcomes and Performance Metrics

Key Performance Indicators

Metric	Current Baseline	Target Improvement
On-time payment rate	90%	95% (+5pp)
Customer satisfaction score	3.2/5	4.0/5
Reminder volume per customer	24.8 (mean)	Reduce by 30%
Channel effectiveness	Email: 36%	Push: 40% (+4pp)

We can use a robust validation framework to measure success:

- **Holdout Validation:**
  - 20% of customers reserved for testing.
  - Compare optimized vs. current strategy.
- **Business Metrics Monitoring:**

- Delinquency rates.
- Customer churn.
- Operational costs.

- **Feedback Analysis:**
  - Sentiment analysis on customer complaints.
  - Survey response tracking.

We'll proactively address potential risks:

- **Over-communication Risk:**
  - Hard limits on reminder frequency per segment.
  - Cool-off periods between reminders.
- **Model Decay:**
  - Automated drift detection.
  - Scheduled retraining pipeline.
- **Channel Fatigue:**
  - Content rotation strategies.
  - Channel-specific fatigue models.