Current State Analysis Plan

This notebook outlines the strategy to assess the current effectiveness of Credit-X's payment reminder communication patterns. The aim is to identify key pain points and failure modes, which will support the redesign of a more efficient and customer-centric reminder system.

Load the tables

In [1]:
```
import pandas as pd
import plotly.express as px
import plotly.io as pio
import plotly.graph_objects as go
import numpy as np
from statsmodels.nonparametric.smoothers_lowess import lowess
from sklearn.ensemble import IsolationForest
from sklearn.preprocessing import StandardScaler

pio.renderers.default = 'iframe_connected'

"from data.generate_data import feedback_df, customers_df, accounts_df,payments_df, reminders_df, schedules_df"

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')
```

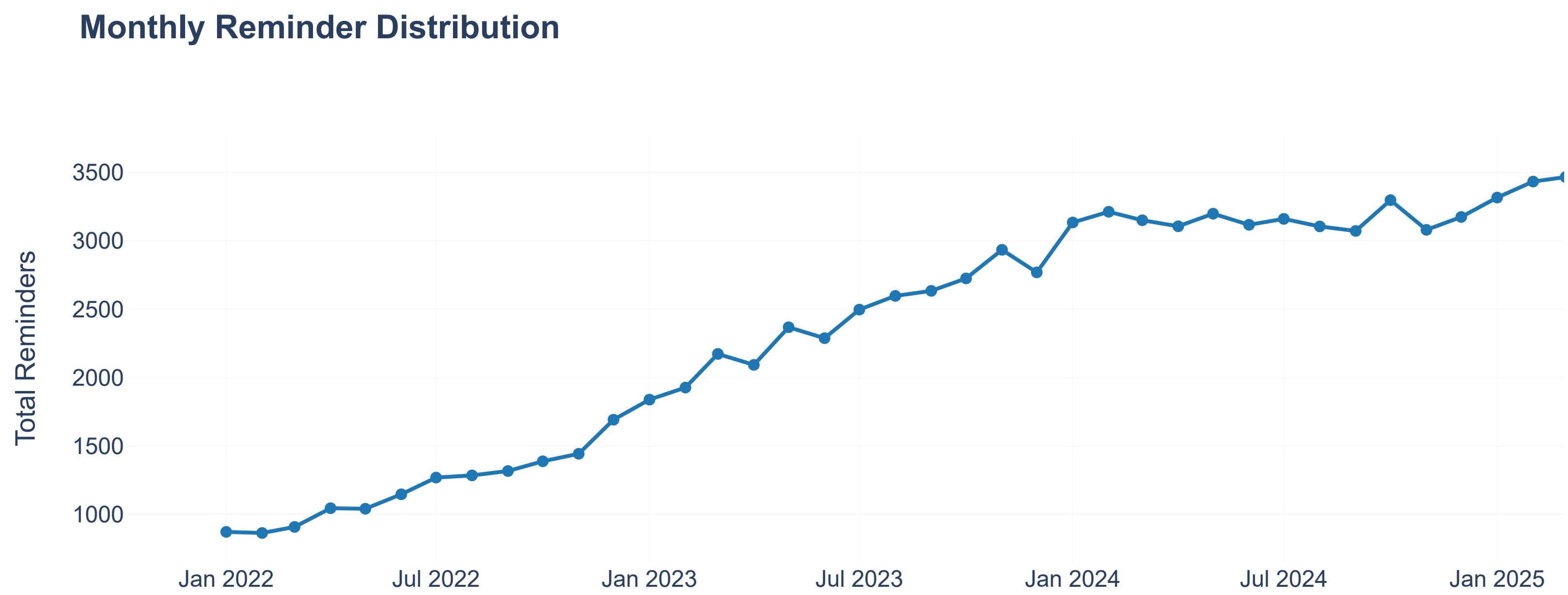
```
C:\Users\gaby\AppData\Local\Temp\ipykernel_12200\2594808351.py:19: DtypeWarning:

Columns (4) have mixed types. Specify dtype option on import or set low_memory=False.
```

Data preprocessing

In [2]:
```
reminders['sent_at'] = pd.to_datetime(reminders['sent_at'])
schedules['due_date'] = pd.to_datetime(schedules['due_date'])
payments['payment_date'] = pd.to_datetime(payments['payment_date'])
customers['signup_date'] = pd.to_datetime(customers['signup_date'])
reminders = pd.merge(reminders, accounts[['account_id', 'customer_id']], on='account_id', how='left')

reminders['month'] = reminders['sent_at'].dt.to_period('M').astype(str)
monthly_reminders = reminders.groupby('month').size().reset_index(name='reminder_count')
```

Urgent Need for Reminder Optimization

In [3]:
```
fig = px.line(
    monthly_reminders,
    x='month',
    y='reminder_count',
    markers=True,
    title='<b>Monthly Reminder Distribution</b>',
    labels={'month': '', 'reminder_count': 'Number of Reminders'},
    color_discrete_sequence=['#1f77b4'],  # Professional blue
    template='plotly_white'
).update_layout(
    yaxis_title='Total Reminders',
    hovermode='x unified',
    font=dict(family="Arial"),
    height=400
)

fig.show()
```



Our monthly reminder distribution shows consistent volume increases. This unsustainable growth demands immediate action to:

Maintain payment effectiveness - More reminders ≠ better results without strategy

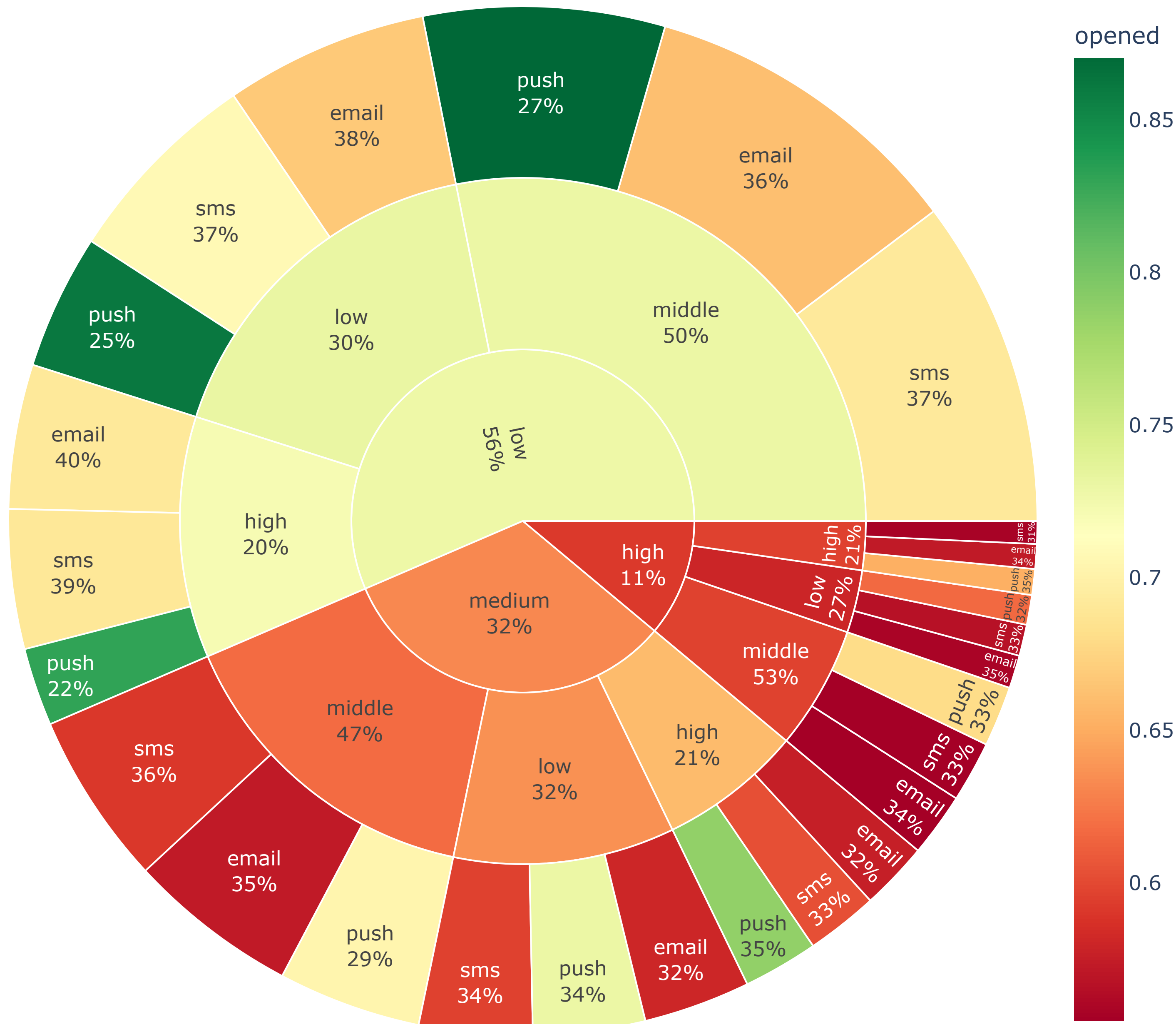
Sunburst Chart: Hierarchical visualization showing reminder effectiveness by customer segment and channel, with:

- Concentric rings for: `risk_tier` → `income_bracket` → `channel`
- Color-coded by open rate
- Sized by payment conversion rate

In [4]:
```
def create_sunburst_chart():
    df_sunburst = pd.merge(
```

```
        reminders.groupby(['customer_id', 'channel']).agg({
            'opened': 'mean',
            'clicked': 'mean',
            'payment_triggered': 'mean'
        }).reset_index(),
        customers[['customer_id', 'risk_tier', 'income_bracket']],
        on='customer_id',
        how='left'
    )

    fig = px.sunburst(
        df_sunburst,
        path=['risk_tier', 'income_bracket', 'channel'],
        values='payment_triggered',
        color='opened',
        color_continuous_scale='RdYlGn',
        title='<b>Reminder Effectiveness by Customer Segment & Channel</b>',
        width=800,
        height=800
    )

    fig.update_traces(
        textinfo="label+percent parent",
        hovertemplate="<b>Segment:</b> %{label}<br>" +
                      "<b>Payment Trigger Rate:</b> %{value:.2f}<br>" +
                      "<b>Open Rate:</b> %{color:.2f}"
    )

    fig.show()

create_sunburst_chart()
```

Reminder Effectiveness by Customer Segment & Channel



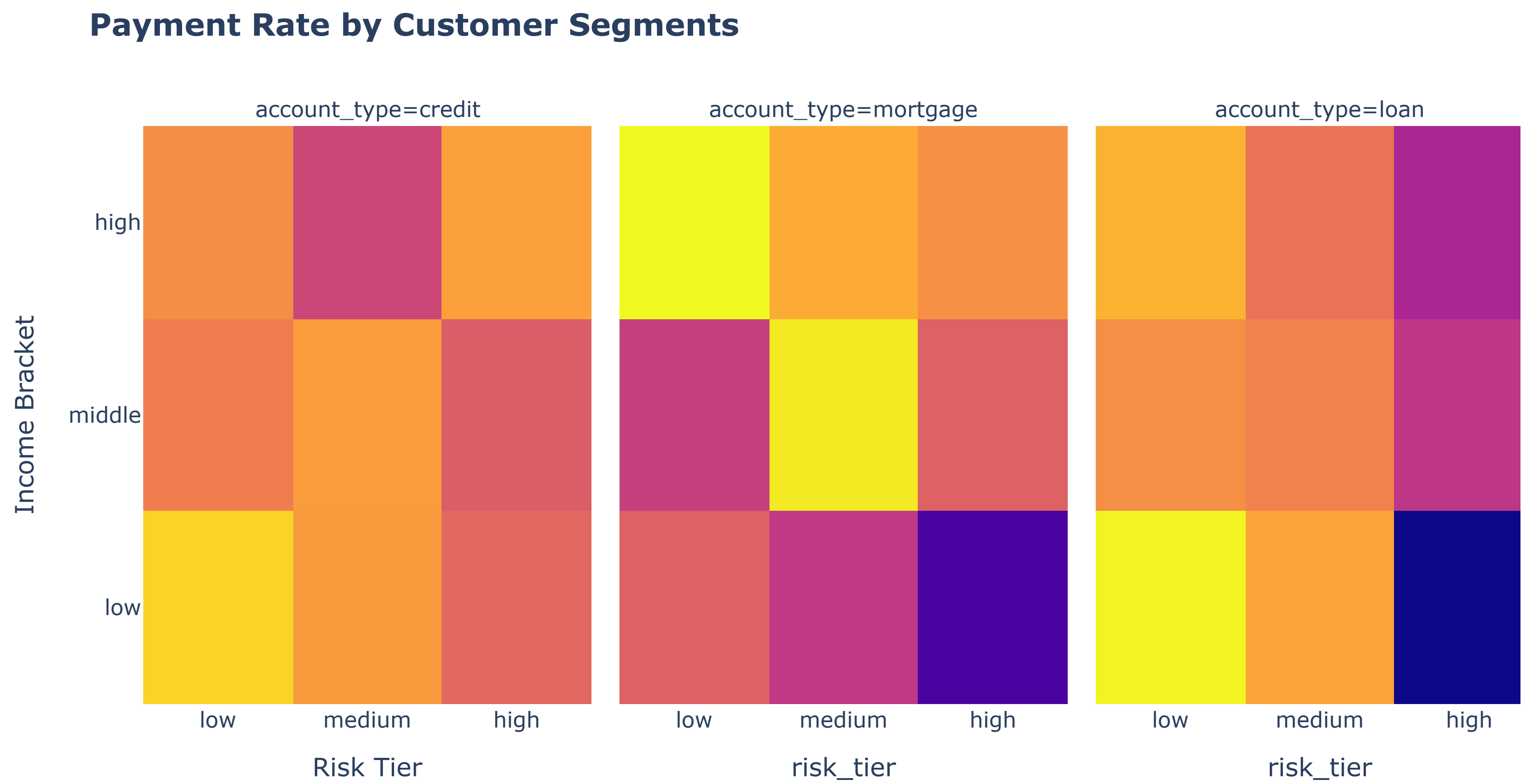
A significant observation from the chart is the notably poorer performance within the "high" risk tier. Customers in this segment consistently exhibit low `opened` rates, shown by the prevalence of red coloring, and consequently, lower rates of `payment_triggered`. This pattern is evident across various income brackets and through channels such as email and SMS. This indicates that the current, undifferentiated reminder strategy is largely failing to engage these higher-risk customers and effectively prompt their payments.

In contrast, "push" notifications generally show strong engagement, characterized by dark green segments indicating high `opened` rates, and appear more effective in leading to payments, particularly for customers in the "low" and "medium" risk tiers. The effectiveness of email and SMS, however, is highly inconsistent; they often demonstrate lower engagement, especially with higher-risk segments. This suggests that while push notifications could be a valuable asset to leverage more broadly, there's a clear need to re-evaluate the content, frequency, and specific targeting of email and SMS communications for various customer groups.

Delinquency Risk Heatmap

In [5]:
```
def create_risk_heatmap():
    # Prepare data
    df_risk = pd.merge(
        accounts,
        customers,
        on='customer_id',
        how='left'
    )

    df_risk = pd.merge(
        df_risk,
        schedules.groupby('account_id')['is_paid'].mean().reset_index(name='payment_rate'),
        on='account_id',
        how='left'
    )

    df_risk = pd.merge(
```

```
        df_risk,
        reminders.groupby('account_id')['payment_triggered'].mean().reset_index(name='reminder_response_rate'),
        on='account_id',
        how='left'
    )

    # Create heatmap
    fig = px.density_heatmap(
        df_risk,
        x='risk_tier',
        y='income_bracket',
        z='payment_rate',
        histfunc="avg",
        facet_col='account_type',
        title='<b>Payment Rate by Customer Segments</b>',
        width=1000,
        height=500
    )

    fig.update_layout(
        xaxis_title="Risk Tier",
        yaxis_title="Income Bracket",
        coloraxis_colorbar=dict(title="Payment Rate")
    )

    fig.show()
create_risk_heatmap()
```



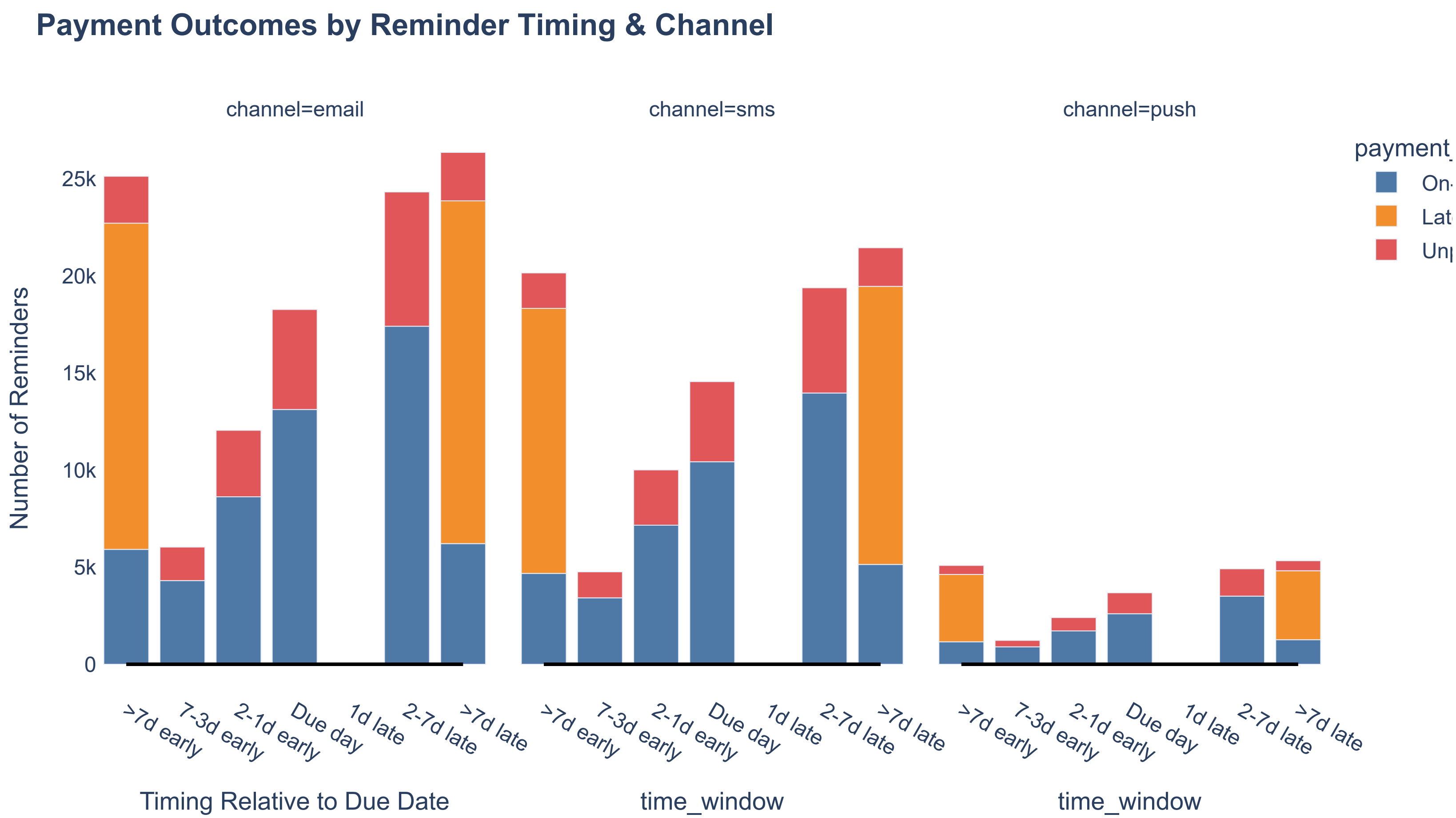
- **Low-risk** customers in **low income brackets** show **high payment rates**.
- **High-risk** customers (any income) show **lower payment rates**. **High-risk + low income** segment shows **very low payment rate low-risk + low income** performs best.
- **Risk Tier** is a strong predictor of payment behavior across all account types.
- **Loan accounts** show the most extreme drop in payment performance among high-risk, low-income customers.
- **Low-income + low-risk** customers are consistently reliable, suggesting potential for targeting or support programs.

Timing

In [6]:
```
#Merge data with  keys
analysis_df = (
    reminders.merge(
        schedules,
        left_on='account_id',
        right_on='account_id',
        how='left'
    )
    .merge(
        payments[['schedule_id', 'days_late']],
        on='schedule_id',
        how='left'
    )
)

# Calculate timing and payment status
analysis_df['days_before_due'] = (
    pd.to_datetime(analysis_df['due_date']) -
    pd.to_datetime(analysis_df['sent_at'])
).dt.days

#determine payment status using days_late and payments_schedule.is_paid
analysis_df['payment_status'] = np.select(
    [
        analysis_df['days_late'] > 0,  # Late payments
        analysis_df['is_paid'] == True  # On-time payments
    ],
    ['Late', 'On-time'],
    default='Unpaid'  # Not paid
)

# 3.bin timing with observed=True
time_bins = [-30, -7, -3, -1, 0, 1, 7, 30]
time_labels = ['>7d early', '7-3d early', '2-1d early', 'Due day', '1d late', '2-7d late', '>7d late']
analysis_df['time_window'] = pd.cut(
    analysis_df['days_before_due'],
    bins=time_bins,
    labels=time_labels
```

```
)

plot_data = (
    analysis_df
    .groupby(
        ['channel', 'time_window', 'payment_status'],
        observed=True
    )
    .size()
    .reset_index(name='count')
)

#visualization
fig = px.bar(
    plot_data,
    x='time_window',
    y='count',
    color='payment_status',
    facet_col='channel',
    title='<b>Payment Outcomes by Reminder Timing & Channel</b>',
    labels={'count': 'Number of Reminders'},
    category_orders={
        'time_window': time_labels,
        'payment_status': ['On-time', 'Late', 'Unpaid'],
        'channel': ['email', 'sms', 'push']
    },
    color_discrete_map={
        'On-time': '#4E79A7',  # Blue
        'Late': '#F28E2B',     # Orange
        'Unpaid': '#E15759'    # Red
    }
)

# trend lines
for i, channel in enumerate(analysis_df['channel'].unique()):
    trend_data = (
        analysis_df[analysis_df['channel'] == channel]
        .groupby('time_window', observed=True)['payment_status']
        .apply(lambda x: (x == 'On-time').mean())
        .reset_index()
    )
```

```
    fig.add_trace(
        px.line(trend_data, x='time_window', y='payment_status')
        .update_traces(
            line_color='black',
            line_width=2,
            showlegend=False
        ).data[0],
        row=1, col=i+1
    )
fig.update_layout(
    height=500,
    width=900,
    plot_bgcolor='white',
    hovermode='x unified',
    font=dict(family="Arial"),
    xaxis_title="Timing Relative to Due Date"
)
fig.show()
```



Email

- High volume of reminders, especially when sent more than 7 days early (>7d early) and exactly on the due date.
- Reminders sent on the "Due day" and "1d late" result in more on-time payments
- Mailings >7d early and >7d late have many unpaid or late payments.

SMS

- Fewer reminders overall than email.
- The trend is similar: higher effectiveness between the due date and 1-2 days after.
- Many unpaid items are also observed if sent too early or too late.
- SMS is somewhat less effective than email but shows a similar trend.

Push

- Very few reminders sent via push, compared to email and SMS.
- Although data is scarce, the results are similar: reminders sent close to the due date have more on-time payments. Could be underutilized or less effective in this sample.

In conclusion:

- Sending reminders close to the due date (especially on the same day or one day after) improves on-time payment.
- Emails are the most used and also the most effective channel, followed by SMS.
- Reminders sent too early or too late have less positive impact.
- Push notifications appear to be marginal or less used.

Reminders Frecuency:

In [7]:
```
reminders.groupby('customer_id')['sent_at'].count().describe()
```

Out[7]:
```
count    4902.000000
mean       24.804570
std        23.621837
min         1.000000
25%         9.000000
50%        17.000000
75%        33.000000
max       147.000000
Name: sent_at, dtype: float64
```

The frequency of reminders exhibits high dispersion, with a standard deviation (std = 23.6) approximately equal to the mean. This indicates a wide variability in the number of reminders received by clients. While the majority of clients receive between 9 and 33 reminders, extreme outliers are present, such as one instance recording 147 reminders. Only 25% of clients receive more than 33 reminders. Therefore, accounts exceeding this threshold warrant immediate attention and should be rigorously analyzed as potentially anomalous or suspicious cases.

Implications:

This quantitative analysis corroborates the insights derived from the histogram: the distribution displays a pronounced long tail. This pattern strongly suggests either systemic inefficiencies in the reminder process, potential misuse of the system, or the presence of automated and/or abusive behaviors. Further investigation into these high-frequency segments is crucial for identifying and mitigating underlying issues.

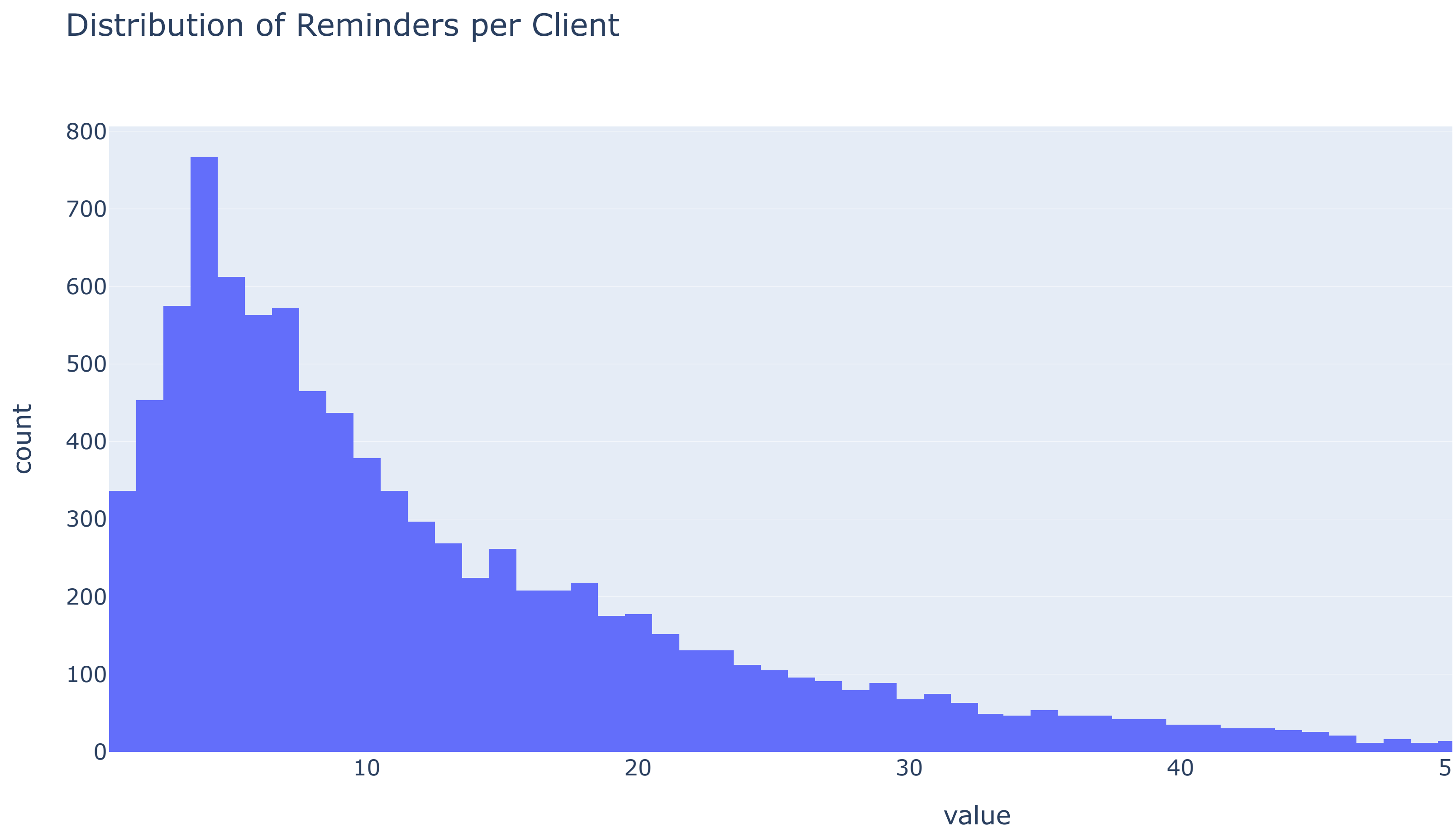
INTERVAL BETWEEN REMINDERS

In [8]:
```
reminders.sort_values(['account_id', 'sent_at']).groupby('account_id')['sent_at'].diff().dt.days.mean()
```

Out[8]:
```
np.float64(54.70440122642518)
```

To understand the frequency with which reminders are sent to accounts, we analyzed the time interval between successive messages. The average difference between consecutive reminders sent to the same account_id is 54.7 days. This suggests that, in general, reminders are spaced out by a considerable period, which could influence the effectiveness of the reminders.

In [9]:
```
px.histogram(reminders.groupby('account_id').size(), title='Distribution of Reminders per Client').update_layout(show
```



A significant portion of our client base receives between 5 and 10 reminders. However, a notable right-skewed long tail indicates a subset of clients receiving in excess of 40, and in some extreme cases, over 60 reminders. This highly anomalous volume of reminders for a minority of accounts could be indicative of sophisticated fraudulent activities. These may include phantom accounts, bot-driven interactions, or abusive users who systematically exploit the reminder infrastructure with no genuine intent of payment. Further investigation into these high-frequency recipients is warranted to identify and mitigate such exploitative patterns.

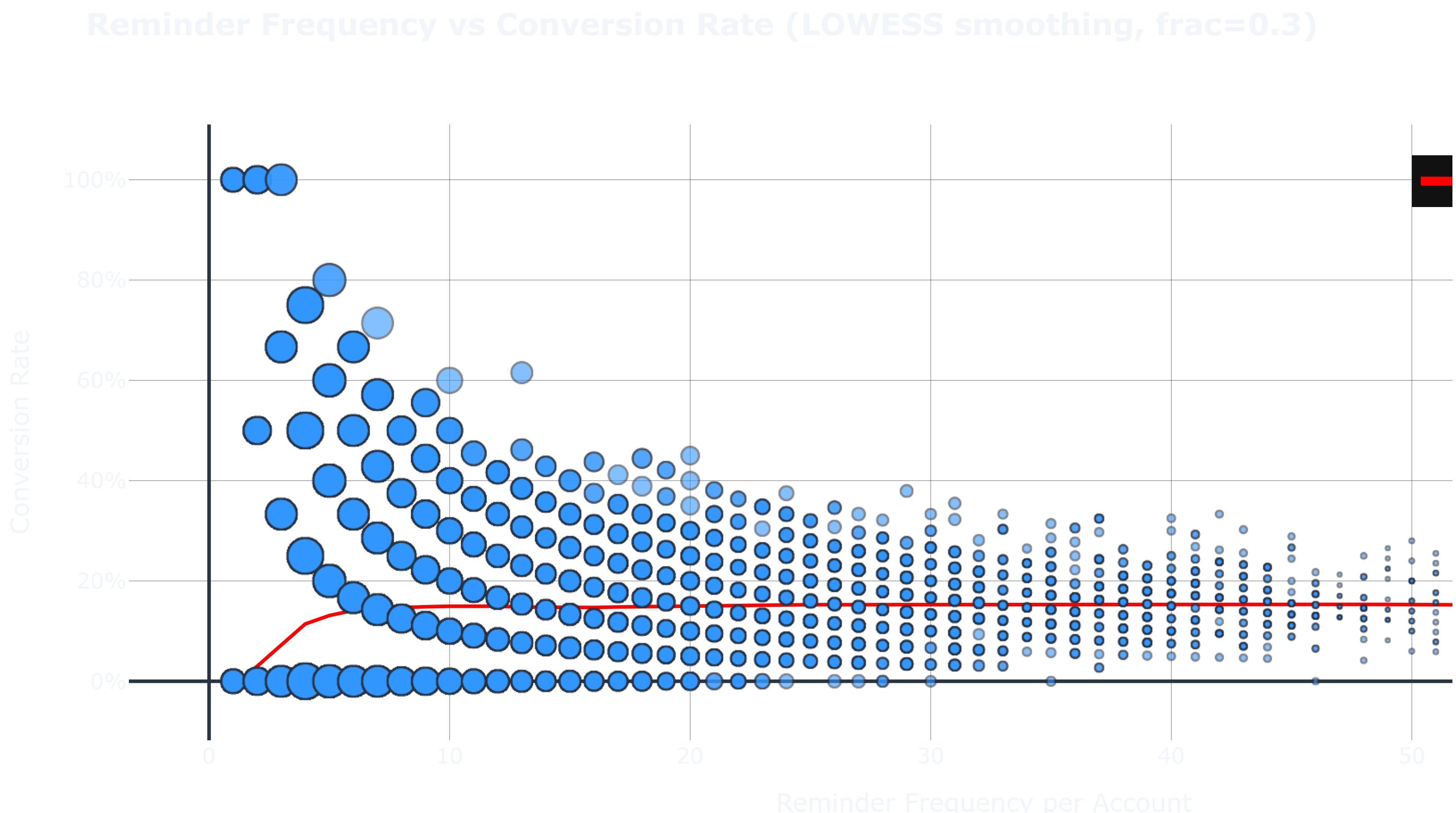
Such inherent inconsistencies within our communication protocols create exploitable windows for malicious actors. Periods characterized by a lack of reminders could be deliberately leveraged to avoid charges, accumulate significant outstanding balances, or simply to remain undetected. The absence of personalization renders the reminder system highly predictable and thus more susceptible to manipulation. This predictability facilitates the development of automated scripts or bots designed to systematically ignore or filter specific message types, thereby undermining the effectiveness of our dunning processes.

Reminder Frequency vs Conversion Rate

In [10]:

```
window_frac = 0.3
# Frequency and conversion rate per account
df = reminders.groupby('account_id').agg(
    frequency=('reminder_id', 'count'),
    conversion_rate=('payment_triggered', 'mean')
).sort_values('frequency').reset_index()

# LOWESS smoothing
smoothed = lowess(df['conversion_rate'], df['frequency'], frac=window_frac)
df['smoothed'] = smoothed[:, 1]

# Count how many accounts per frequency
freq_counts = df['frequency'].value_counts().reset_index()
freq_counts.columns = ['frequency', 'account_count']
df = df.merge(freq_counts, on='frequency', how='left')

fig = px.scatter(
    df,
    x='frequency',
    y='conversion_rate',
    size='account_count',
    title=f'<b>Reminder Frequency vs Conversion Rate (LOWESS smoothing, frac={window_frac})</b>',
    labels={
        'frequency': 'Reminder Frequency per Account',
        'conversion_rate': 'Conversion Rate',
        'account_count': 'Number of Accounts'
    },
    opacity=0.6,
    color_discrete_sequence=['#3399FF']
)
```

```
# Smoothed trend line
fig.add_trace(
    go.Scatter(
        x=df['frequency'],
        y=df['smoothed'],
        mode='lines',
        line=dict(color='red', width=2),
        name='Smoothed Trend (LOWESS)'
    )
)

fig.update_layout(
    yaxis_tickformat=',.0%',
    hovermode='x unified',
    template='plotly_dark',
    legend=dict(x=0.75, y=0.95)
)

fig.show()
```



The red LOWESS line clearly indicates that beyond approximately 5–10 reminders, the conversion rate does not exhibit significant improvement. In fact, it tends to stabilize around 15–18%, even when clients receive upwards of 50 reminders. This suggests a clear saturation point in our reminder strategy. Sending an excessive number of messages beyond this threshold does not lead to enhanced client behavior or increased conversions. The size of the data points in the visualization indicates that the bulk of our client accounts fall within the 1 to 25 reminder range. This segment represents our largest opportunity for impact. An optimized strategy focusing on this core group can yield the most significant improvements in overall conversion.

What Works:

- Initial Reminders (1–5): These early reminders can be highly effective for specific client segments.
- A highly personalized strategy during these initial contacts is likely to be far more effective than an indiscriminate or indefinitely persistent approach.

What Does Not Work:

- *Persisting with more than 25 reminders rarely yields an increase in conversions.
- Such excessive communication could, as indicated by the problem's background, negatively impact the customer experience and foster frustration.

FRAUD ANALYSIS

Isolation Forest

In our context, Isolation Forest is instrumental in detecting customers whose payment behavior is significantly different from the general population, potentially indicating anomalous or fraudulent activities.

In [11]:
```
# CLIENT BEHAVIOR
df = payments.merge(customers, on="customer_id")
df = df.merge(accounts[['account_id', 'customer_id']], on="customer_id", how="left")

features = df.groupby('customer_id').agg({
    'amount_paid': ['mean', 'std', 'sum'],
    'days_late': ['mean', 'std', 'max'],
    'credit_score': 'mean',
    'account_id': 'nunique',
}).reset_index()

# RENAME COLUMNS
features.columns = ['customer_id', 'amount_paid_mean', 'amount_paid_std', 'amount_paid_sum',
                    'days_late_mean', 'days_late_std', 'days_late_max',
                    'credit_score', 'n_accounts']

features = features.fillna(0)  #
```

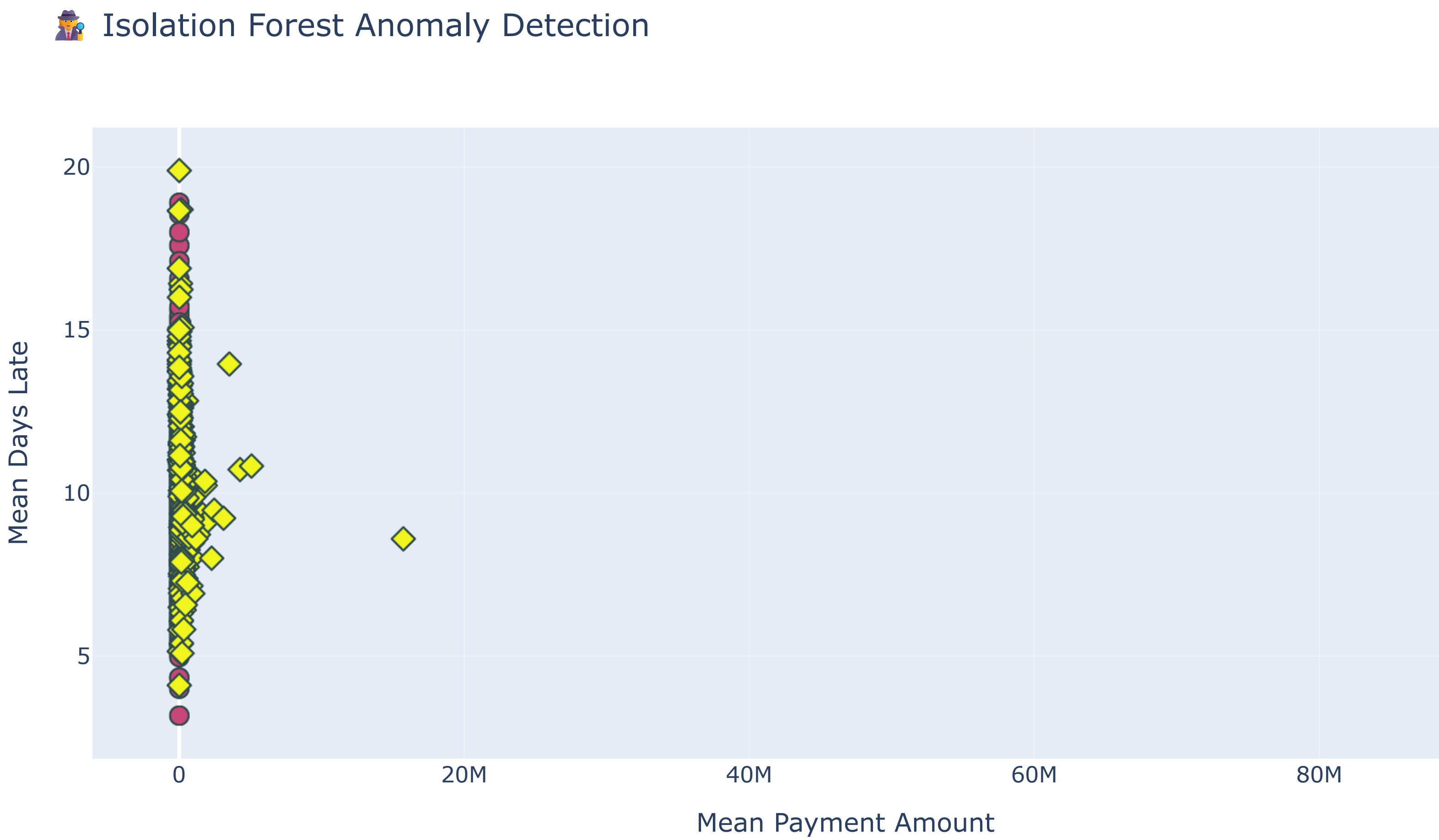
In [12]:
```python
# Normalize
scaler = StandardScaler()
X = scaler.fit_transform(features.drop('customer_id', axis=1))

# Modelling
iso = IsolationForest(contamination=0.05, random_state=42)
features['anomaly_score'] = iso.fit_predict(X)
features['anomaly'] = features['anomaly_score'].apply(lambda x: 1 if x == -1 else 0)
```

In [13]:
```python
suspicious = features[features['anomaly'] == 1].sort_values(by='days_late_mean', ascending=False)
display(suspicious.head(10))
```

	customer_id	amount_paid_mean	amount_paid_std	amount_paid_sum	days_late_mean	days_late_std	days_late_max	credit_
141	CUST_00141	4912.721111	1454.380020	132643.47	19.888889	25.404926	128	
1750	CUST_01750	136499.781923	47334.528476	3548994.33	18.692308	24.427475	106	
4957	CUST_04957	1833.413437	623.059364	58669.23	18.656250	27.991484	135	
3706	CUST_03706	2908.730278	899.392794	104714.29	16.888889	24.770694	134	
479	CUST_00479	94585.656154	30229.930950	2459227.06	16.423077	18.891634	60	
1299	CUST_01299	128394.357600	45677.698030	3209858.94	16.240000	23.315374	106	
2502	CUST_02502	14254.671176	2440.802601	242329.41	16.000000	36.250862	154	
4659	CUST_04659	203457.379231	52235.217581	5289891.86	15.076923	24.994276	114	
207	CUST_00207	89084.147692	34994.364038	1158093.92	15.076923	18.404988	55	
83	CUST_00083	39344.871461	43008.739889	10505080.68	15.056180	20.572644	123	

In [14]:
```python
fig = px.scatter(
    features,
    x='amount_paid_mean',
    y='days_late_mean',
    color='anomaly',
    symbol='anomaly',
    hover_data=['customer_id', 'credit_score', 'n_accounts'],
```

```
    color_discrete_map={0: 'blue', 1: 'red'},
    title='🕵️ Isolation Forest Anomaly Detection',
    labels={
        'amount_paid_mean': 'Mean Payment Amount',
        'days_late_mean': 'Mean Days Late',
        'anomaly': 'Anomaly Detected'
    }
)

fig.update_traces(marker=dict(size=10, line=dict(width=1, color='DarkSlateGrey')))
fig.update_layout(legend_title_text='Anomaly (1 = Suspicious)')
fig.show()
```



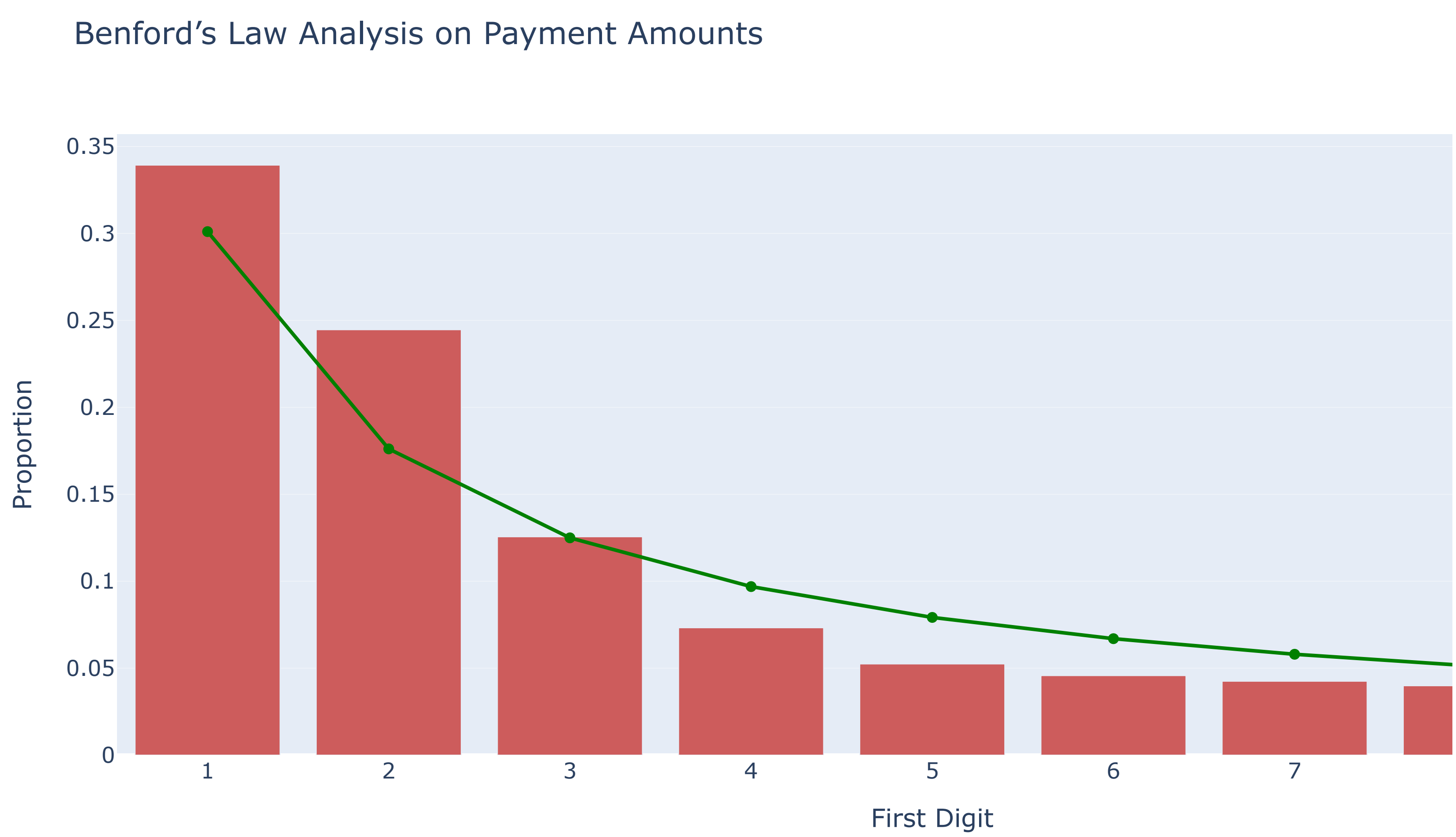
Most customers cluster on the left side, indicating moderate to low payment amounts and delays. These are likely typical, non-risky clients. A few bright yellow outliers are positioned far to the right, signifying exceptionally high average payment amounts. These customers exhibit an unusual spending pattern that sharply deviates from the majority. Some anomalous customers also demonstrate higher than average payment delays, which may be indicative of intentional payment manipulation or elevated risk behavior.

BENFORD LAW

Benford's Law states that in natural datasets, the leading digits of numbers are not uniformly distributed. Specifically, the number 1 appears as the first digit approximately 30% of the time, 2 appears around 17%, and the frequency decreases for higher digits.

This expected distribution is typically not observed in fabricated or manipulated numbers, making Benford's Law a valuable tool for detecting potential fraud.

In [15]:
```
# make sure we have positive numbers
payments_clean = payments[payments['amount_paid'] > 0].copy()

# first digit
payments_clean['first_digit'] = payments_clean['amount_paid'].astype(str).str.replace('.', '').str.lstrip('0').str[0

observed = payments_clean['first_digit'].value_counts(normalize=True).sort_index()


# what Benford expect
benford_dist = {d: np.log10(1 + 1/d) for d in range(1, 10)}
benford_df = pd.DataFrame({
    'Digit': list(benford_dist.keys()),
    'Benford': list(benford_dist.values()),
    'Observed': [observed.get(d, 0) for d in range(1, 10)]
})

fig = go.Figure()
fig.add_trace(go.Bar(x=benford_df['Digit'], y=benford_df['Observed'],
                     name='Observed', marker_color='indianred'))
fig.add_trace(go.Scatter(x=benford_df['Digit'], y=benford_df['Benford'],
                         mode='lines+markers', name='Benford', line=dict(color='green')))

fig.update_layout(
    title='Benford’s Law Analysis on Payment Amounts',
    xaxis_title='First Digit',
    yaxis_title='Proportion',
    barmode='group',
    legend_title='Distribution'
)
fig.show()
```



Problem Formulation

We aim to:

- **Maximize** payment compliance (minimize delinquency).

- **Minimize** customer dissatisfaction (measured via feedback scores).
- **Subject to** operational and business constraints.

Let's define the key decision variables:

- $x_{i,j,k,t}$: Binary variable (1 if reminder j is sent to customer i via channel k at time t, else 0).
- f_i: Reminder frequency (number of reminders sent to customer i per billing cycle).
- τ_i: Timing of reminders relative to the due date (e.g., days before/after the due date).
- c_i: Channel mix (SMS, email, push) for customer i.

3. Objective Function

We employ a **multi-objective function** to balance competing priorities:

Maximize Payment Compliance (Primary Objective)

$$\text{Maximize} \sum_{i=1}^{N} (P(\text{Payment}_i \mid x_{i,j,k,t}))$$

Where $P(\text{Payment}_i)$ represents the probability customer i pays on time given the reminders.

Minimize Customer Dissatisfaction (Secondary Objective)

$$\text{Minimize} \sum_{i=1}^{N} (\text{Dissatisfaction}_i(f_i, \tau_i, c_i))$$

Dissatisfaction increases with:

- Excessive reminders ($f_i > f_{\text{max}}$).
- Poorly timed reminders (too early/late).
- Non-preferred channels.

Operational Cost Control (Tertiary Objective)

$$\text{Minimize} \sum_{i=1}^{N} (\text{Cost}_i(c_i))$$

Here, cost depends on the channel (e.g., SMS typically costs more than email).

Combined Objective (Weighted Approach)

$$\text{Maximize} \alpha \cdot \text{Payment Compliance} - \beta \cdot \text{Dissatisfaction} - \gamma \cdot \text{Cost}$$

Weights α, β, γ can be tuned based on business priorities.

Frequency Constraints

$$f_i \leq f_{\max} \quad \forall i \quad \text{(e.g., max 5-10 reminders per cycle)}$$

Timing Constraints

$$\tau_i \in [\tau_{\min}, \tau_{\max}] \quad \text{(e.g., -3 to +7 days from due date)}$$

Channel Constraints

$$\sum_k x_{i,j,k,t} \leq 1 \quad \text{(No duplicate reminders on same day)}$$

Customer Preference Constraints

$$\text{If Preference}_i = \text{SMS, then } c_i \text{ must include SMS.}$$

Budget Constraints

$$\sum_{i=1}^{N} \text{Cost}_i(c_i) \leq B \quad \text{(Total cost} \leq \text{budget)}$$

Assumptions

- **Linearity of Response**: We assume reminder effectiveness follows a **diminishing returns** curve, not strictly linear.
- **Customer Independence**: We assume one customer’s response does not affect others (no network effects).
- **Data Quality**: We assume payment and feedback data are accurate and representative.