In [1]:
```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, roc_auc_score, roc_curve
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import silhouette_score
from xgboost import XGBClassifier
import plotly.io as pio
pio.renderers.default = "notebook"
```

In [2]:
```
# Loading and prepairing dataset from CSV file
df = pd.read_csv(r"C:\Users\Admin\Desktop\Apply Job 2025\BankChurners.csv")
df.columns = df.columns.str.lower()
df = df.iloc[:, :-2]
df.head()
```

Out[2]:

	clientnum	attrition_flag	customer_age	gender	dependent_count	education_level	marital_status	income_category	card_category	mo
0	768805383	Existing Customer	45	M	3	High School	Married	$60K – 80K	Blue	
1	818770008	Existing Customer	49	F	5	Graduate	Single	Less than $40K	Blue	
2	713982108	Existing Customer	51	M	3	Graduate	Married	$80K – 120K	Blue	
3	769911858	Existing Customer	40	F	4	High School	Unknown	Less than $40K	Blue	
4	709106358	Existing Customer	40	M	3	Uneducated	Married	$60K – 80K	Blue	

5 rows × 21 columns

In [3]:
```
# Get the describe from dataset
df.describe()
```

Out[3]:

	clientnum	customer_age	dependent_count	months_on_book	total_relationship_count	months_inactive_12_mon	contacts_count_
count	1.012700e+04	10127.000000	10127.000000	10127.000000	10127.000000	10127.000000	10127
mean	7.391776e+08	46.325960	2.346203	35.928409	3.812580	2.341167	2
std	3.690378e+07	8.016814	1.298908	7.986416	1.554408	1.010622	1
min	7.080821e+08	26.000000	0.000000	13.000000	1.000000	0.000000	0
25%	7.130368e+08	41.000000	1.000000	31.000000	3.000000	2.000000	2
50%	7.179264e+08	46.000000	2.000000	36.000000	4.000000	2.000000	2
75%	7.731435e+08	52.000000	3.000000	40.000000	5.000000	3.000000	3
max	8.283431e+08	73.000000	5.000000	56.000000	6.000000	6.000000	6

In [4]:
```
# Check for missing values
df.isnull().sum()
```

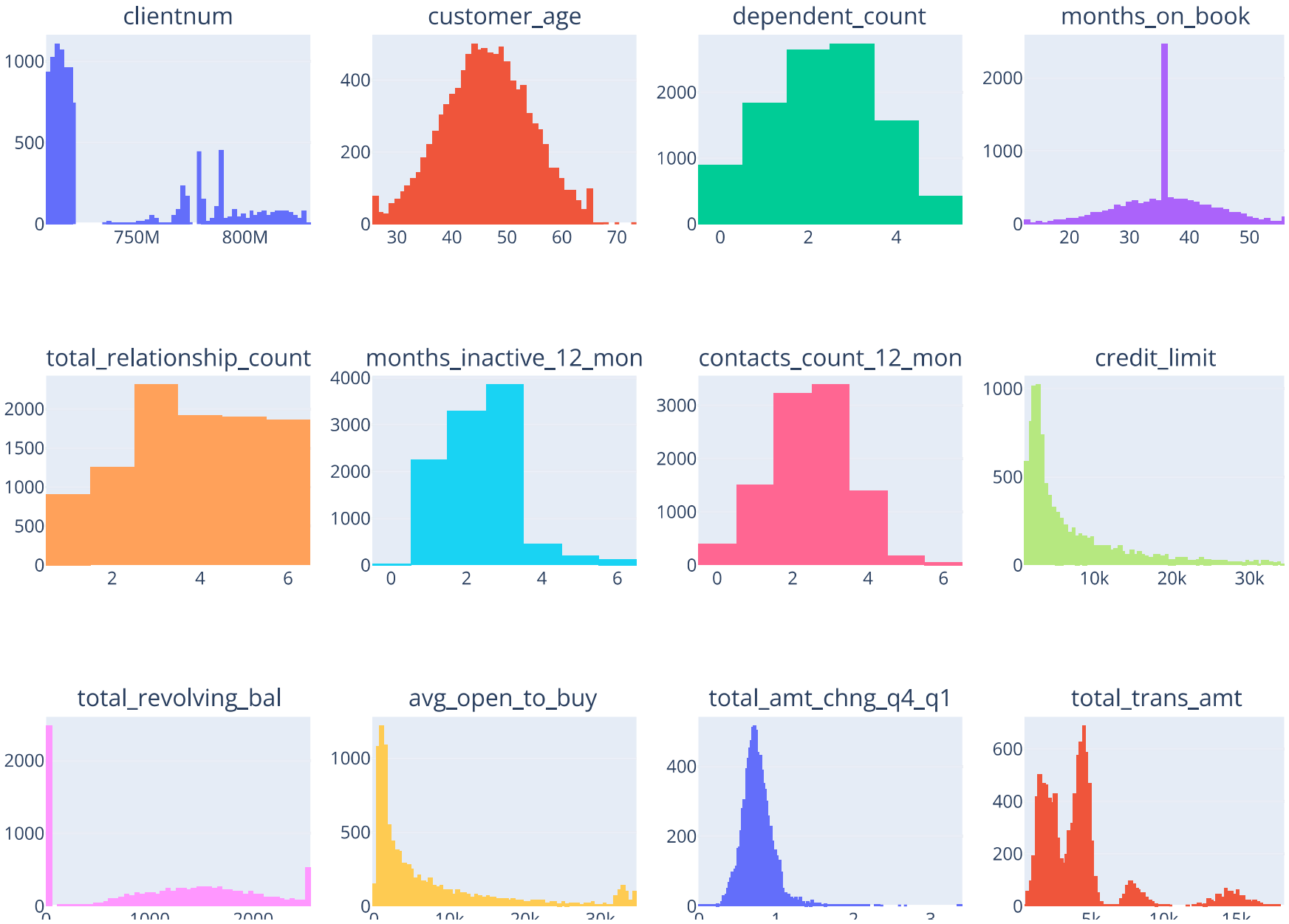
Out[4]:
```
clientnum                 0
attrition_flag            0
customer_age              0
gender                    0
dependent_count           0
education_level           0
marital_status            0
income_category           0
card_category             0
months_on_book            0
total_relationship_count  0
months_inactive_12_mon    0
contacts_count_12_mon     0
credit_limit              0
total_revolving_bal       0
avg_open_to_buy           0
total_amt_chng_q4_q1      0
total_trans_amt           0
total_trans_ct            0
total_ct_chng_q4_q1       0
avg_utilization_ratio     0
dtype: int64
```

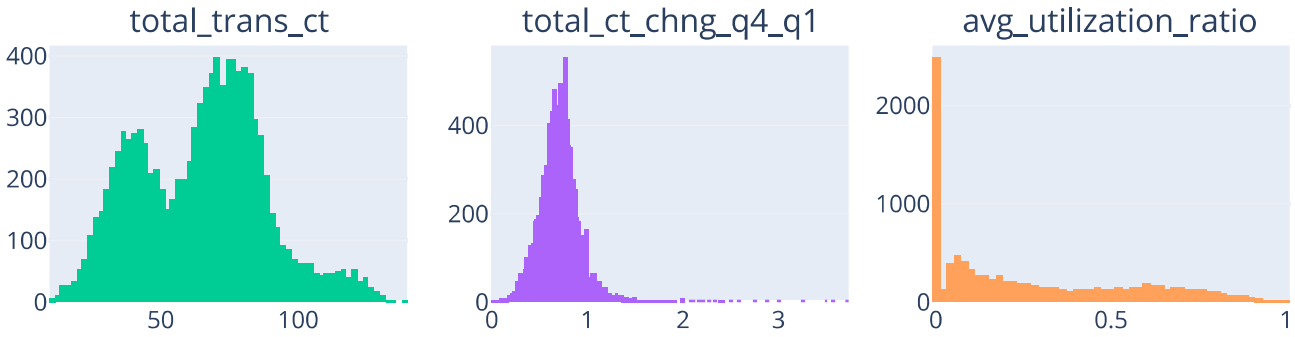
In [5]:
```
# Fill or drop missing values
df.nunique()
```

Out[5]:
```
clientnum                 10127
attrition_flag                2
customer_age                 45
gender                        2
dependent_count               6
education_level               7
marital_status                4
income_category               6
card_category                 4
months_on_book               44
total_relationship_count      6
months_inactive_12_mon        7
contacts_count_12_mon         7
credit_limit               6205
total_revolving_bal        1974
avg_open_to_buy            6813
total_amt_chng_q4_q1       1158
total_trans_amt            5033
total_trans_ct              126
total_ct_chng_q4_q1         830
avg_utilization_ratio       964
dtype: int64
```

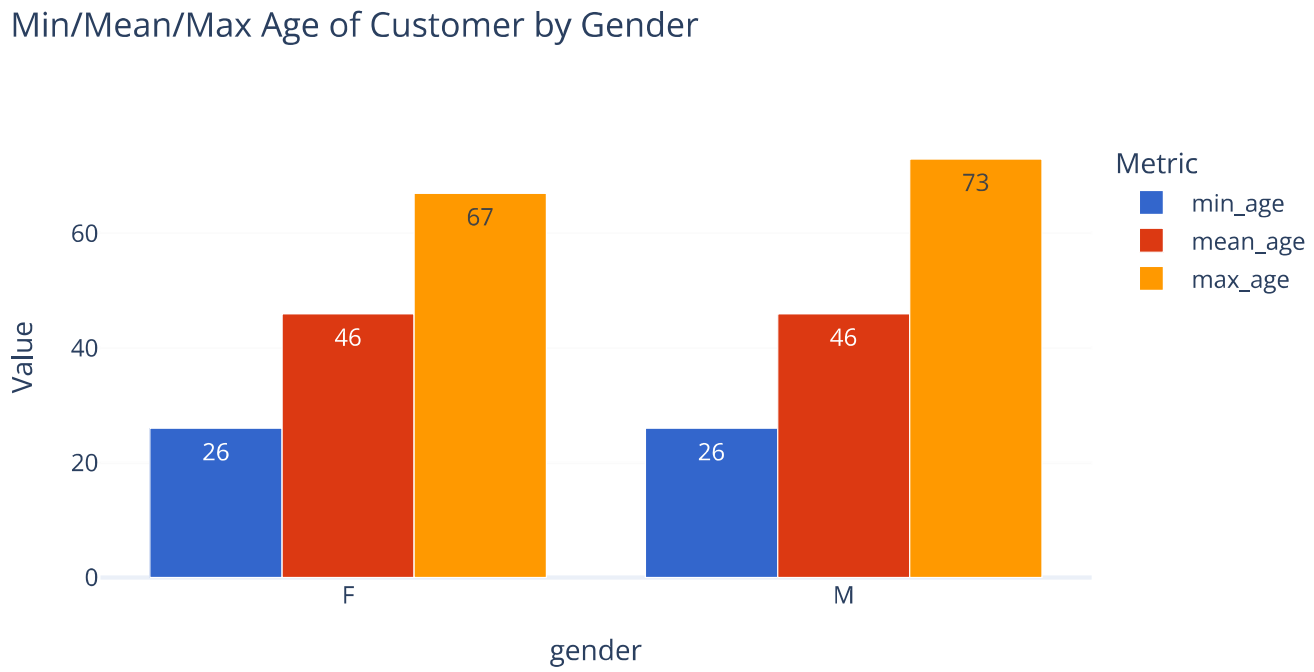
In [6]:
```python
# Get numeric columns and calculate grid size
numeric_cols = df.select_dtypes(include='number').columns
rows = cols = int(np.ceil(np.sqrt(len(numeric_cols))))

# Create histograms in a subplot
fig = make_subplots(rows=rows, cols=cols, subplot_titles=numeric_cols)

for i, col in enumerate(numeric_cols):
    fig.add_trace(go.Histogram(x=df[col], name=col), row=(i // cols) + 1, col=(i % cols) + 1)

# Adjust layout and display
fig.update_layout(height=rows*250, width=cols*250, title="Histograms", showlegend=False)
fig.show()
```

Histograms

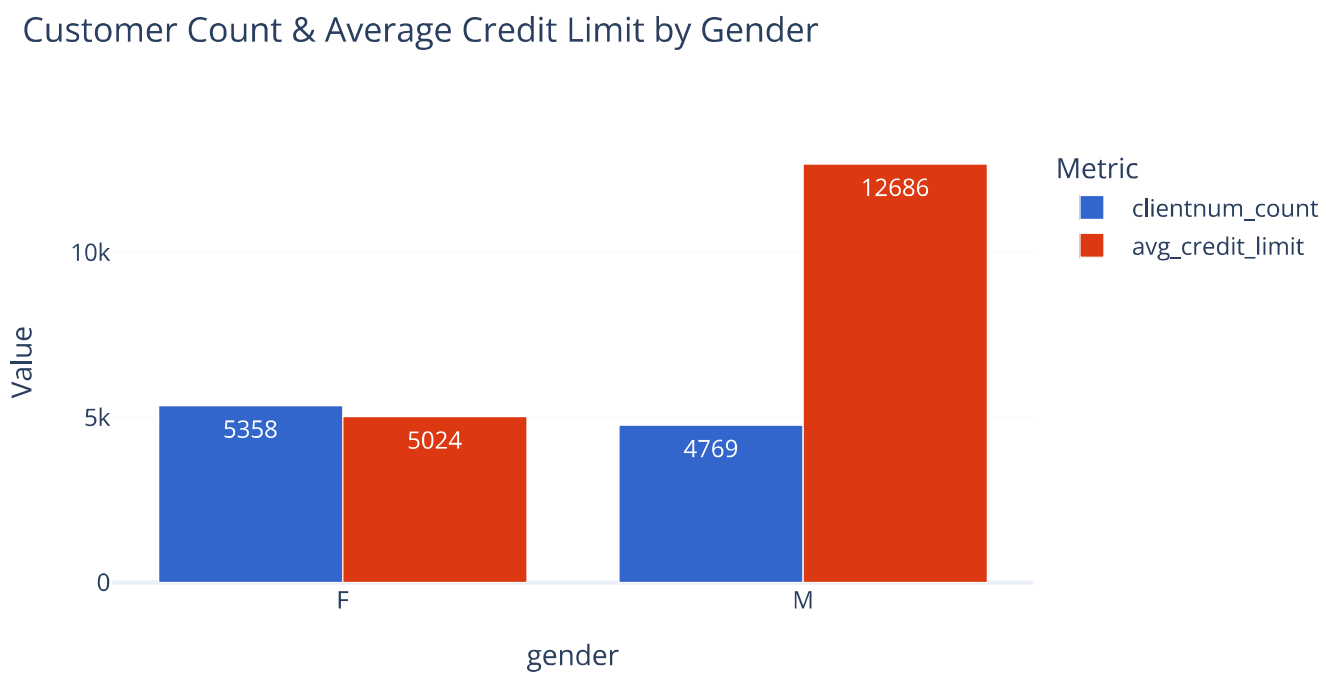

clientnum
1000
500
0
750M
800M
customer_age
400
200
0
30
40
50
60
70
dependent_count
2000
1000
0
0
2
4
months_on_book
2000
1000
0
20
30
40
50
total_relationship_count
2000
1500
1000
500
0
2
4
6
months_inactive_12_mon
4000
3000
2000
1000
0
0
2
4
6
contacts_count_12_mon
3000
2000
1000
0
0
2
4
6
credit_limit
1000
500
0
10k
20k
30k
total_revolving_bal
2000
1000
0
1000
2000
avg_open_to_buy
1000
500
0
10k
20k
30k
total_amt_chng_q4_q1
400
200
0
1
2
3
total_trans_amt
600
400
200
0
5k
10k
15k



In [7]:
```
# Plotting Age of Customer by Gender
df_melted = df.groupby('gender').agg( min_age=('customer_age', 'min'),
                                      mean_age=('customer_age', lambda x: round(x.mean())),
                                      max_age=('customer_age', 'max')).reset_index().melt(id_vars='gender', var_name='Metric',

fig = px.bar(df_melted, x='gender', y='Value', color='Metric', barmode='group', text='Value',
             template='plotly_white', color_discrete_sequence=px.colors.qualitative.G10,
             title='Min/Mean/Max Age of Customer by Gender', height=400, width=700)
fig.show()
```



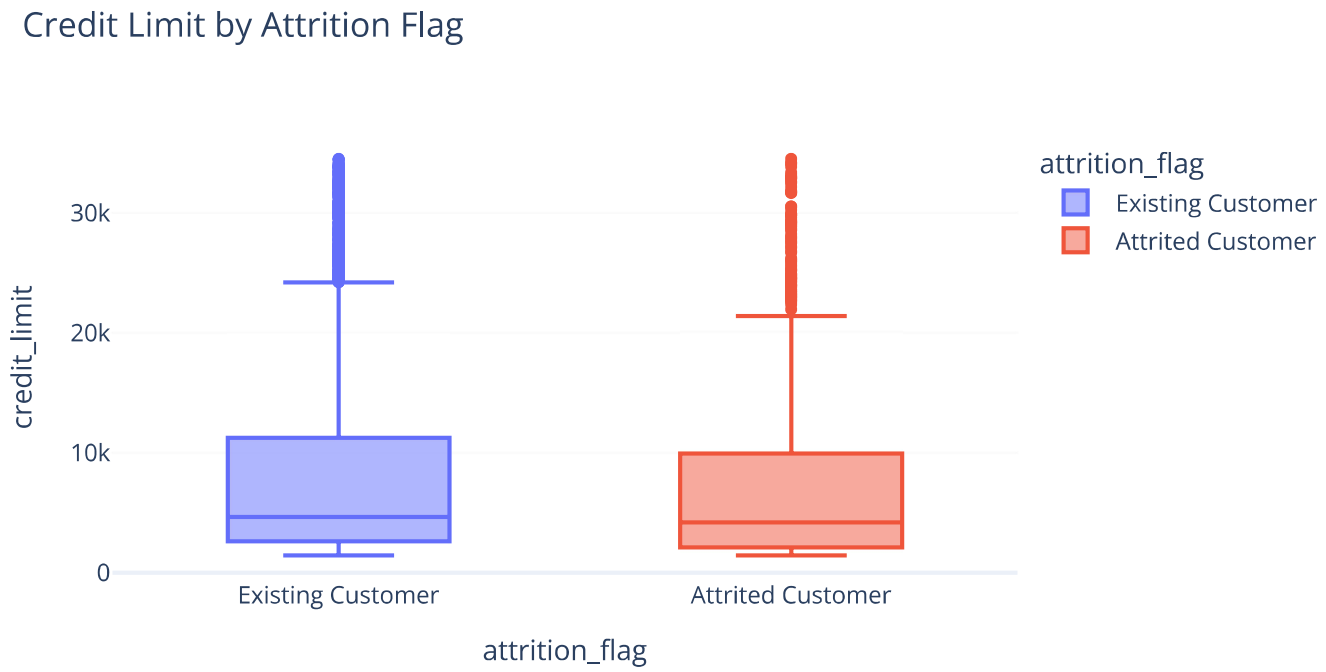
In [8]:
```
# Plotting Customer Count & Average Credit Limit by Gender
df_melted = df.groupby('gender').agg(clientnum_count=('clientnum','count'),
                                     avg_credit_limit=('credit_limit', lambda x: round(x.mean()))
                                    ).reset_index().melt(id_vars='gender', var_name='Metric', value_name='Value')

fig = px.bar(df_melted, x='gender', y='Value', color='Metric', barmode='group',
             text='Value', template='plotly_white', height=400, width=700,
             color_discrete_sequence=px.colors.qualitative.G10, title='Customer Count & Average Credit Limit by Gender')

fig.show()
```



Observations

- Male customers, while fewer in number of clients, have a significantly higher average credit limit (12686) compared to female customers (5024). This suggests a notable Gender-based difference in credit allocation or financial profiles.
- Since both Genders have a similar mean age, age does not seem to be a determining factor in the disparity of credit limits.
- Understanding why a larger proportion of male customers receive higher credit limits could help in devising strategies to balance this disparity, keep the loyal from customer and avoid churn.

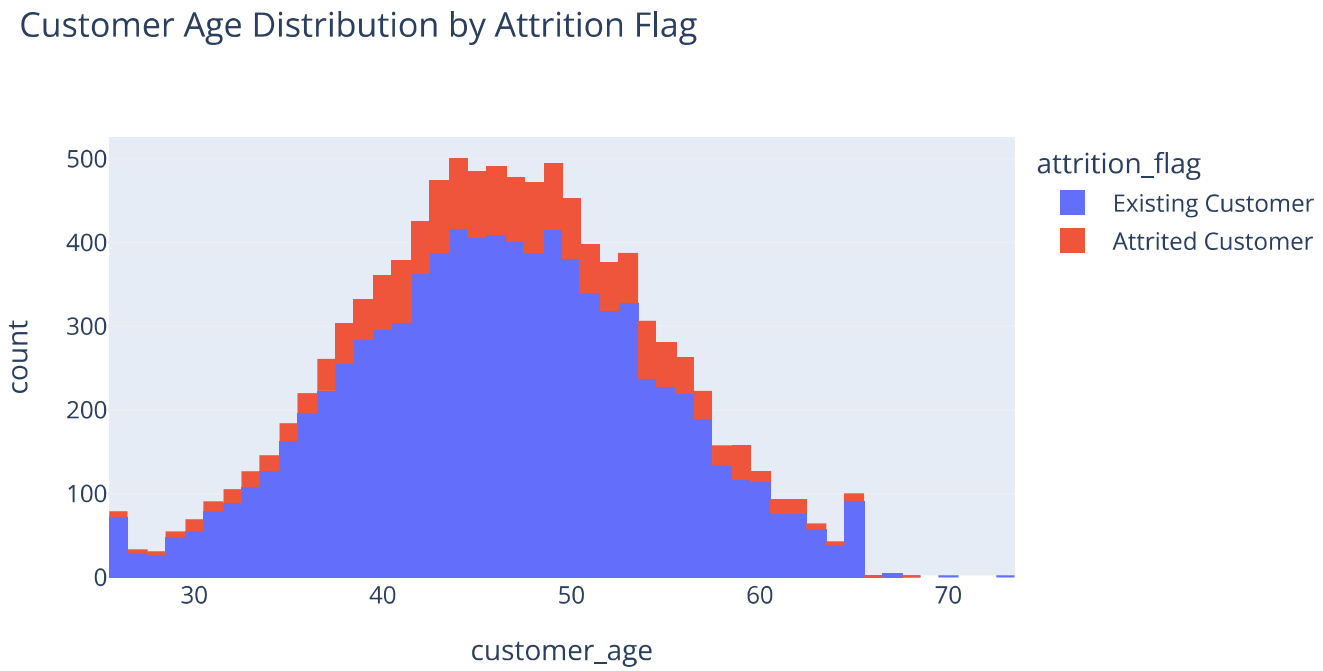
In [9]:
```
# Create a box plot for Credit Limit by Attrition Flag
fig = px.box(df, x='attrition_flag', y='credit_limit', height=400, width=700,
             title="Credit Limit by Attrition Flag",
             color='attrition_flag', template='plotly_white')
fig.show()
```



Observations

- The credit limit setting process for ECs might be less stringent than that for ACs, leading to more outliers and a higher dispersion of credit limits.
- ACs might have removed some customers with unusual credit limits, leading to fewer outliers and a lower dispersion of credit limits.
- ECs tend to have higher credit limits than ACs, but also come with higher risks due to the higher dispersion and outliers.
- Further analysis of other factors that might affect credit limits and customer attrition rates is needed to get a more accurate assessment.

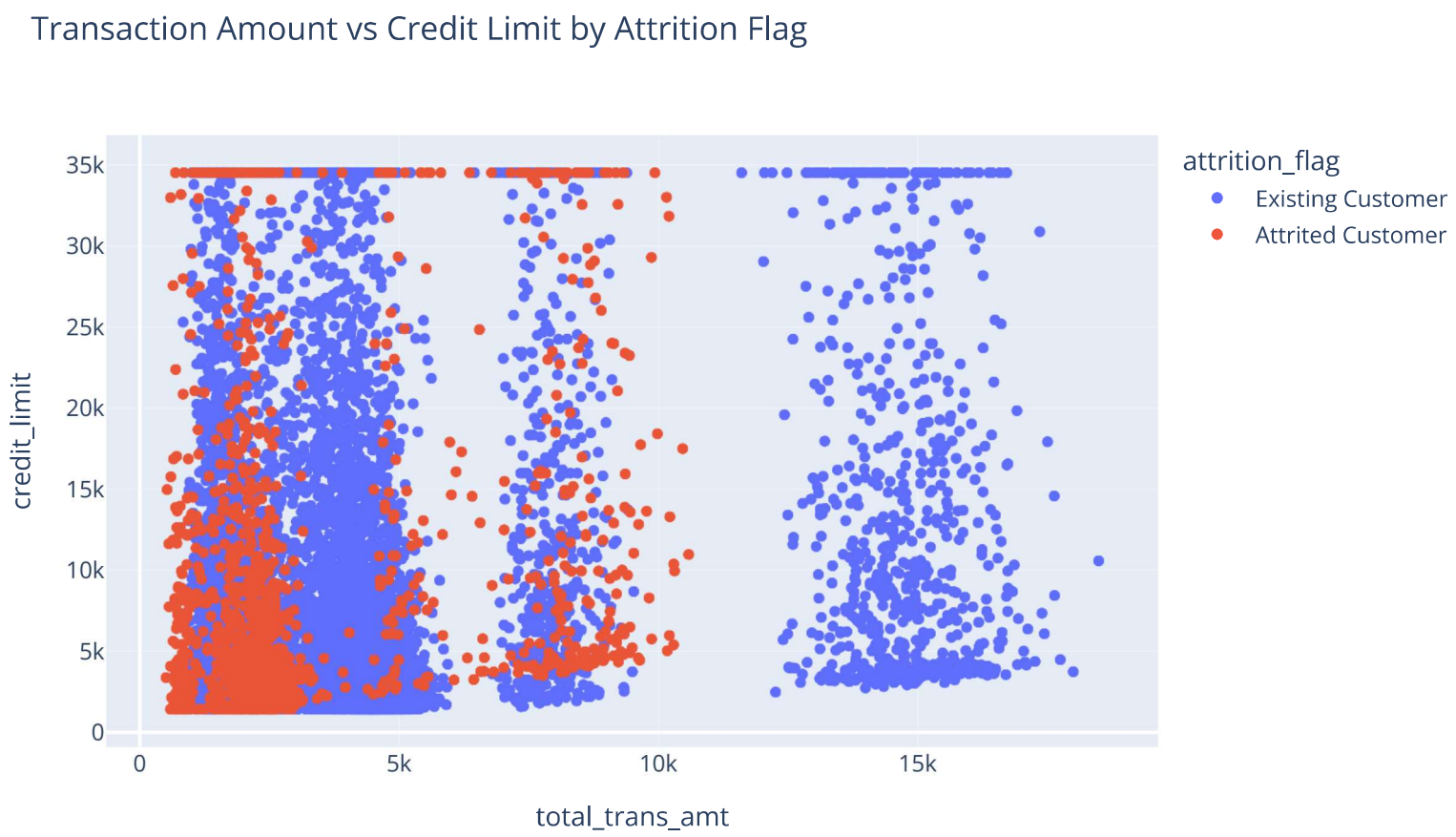
In [10]:
```
# Plotting Customer Age Distribution by Attrition Status
fig = px.histogram(df, x='customer_age', color='attrition_flag', title='Customer Age Distribution by Attrition Flag', height=4
fig.show()
```



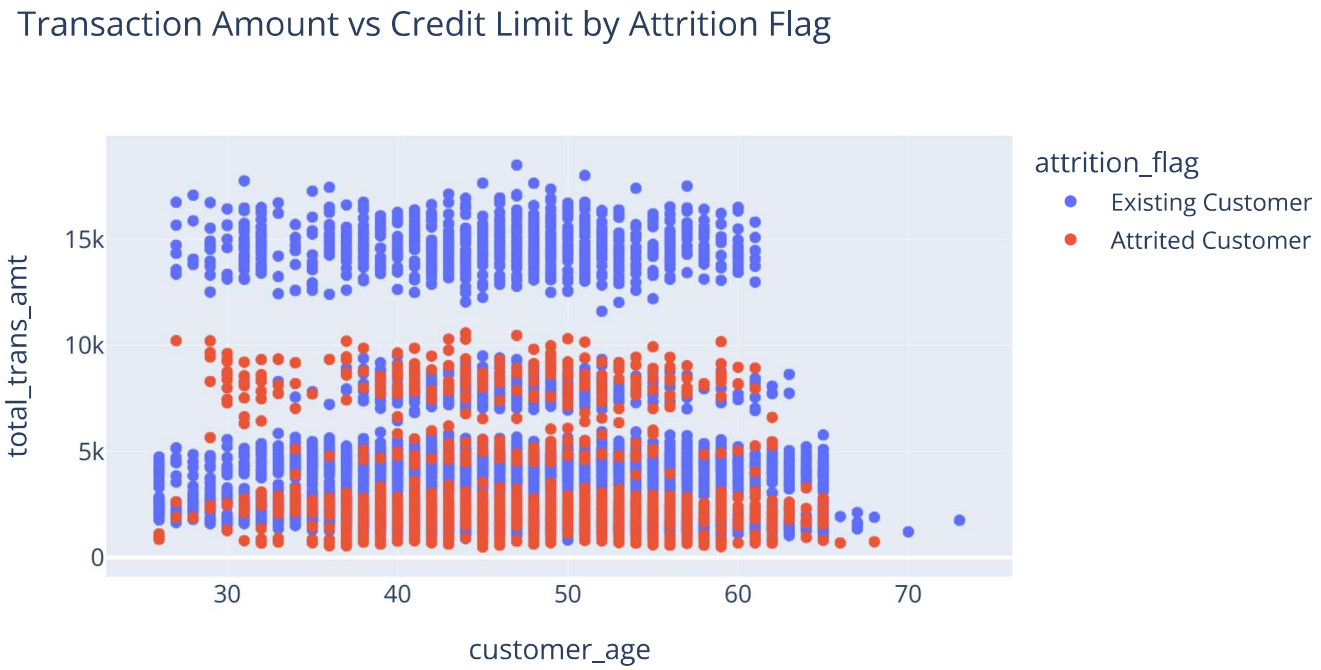
Observations

We can see a same silhouette of age for 2 types of the customer. Based on the boxplot chart, we can suppose that the credit limit affects the customer churn.

In [11]:
```
# Plotting Transaction Amount versus Credit Limit
fig = px.scatter(df, x='total_trans_amt', y='credit_limit', color='attrition_flag', title='Transaction Amount vs Credit Limit
fig.show()
```



In [12]:
```
# Plotting Customer Age versus Total Transaction Amount
fig = px.scatter(df, x='customer_age', y='total_trans_amt', color='attrition_flag', title='Transaction Amount vs Credit Limit
fig.show()
```



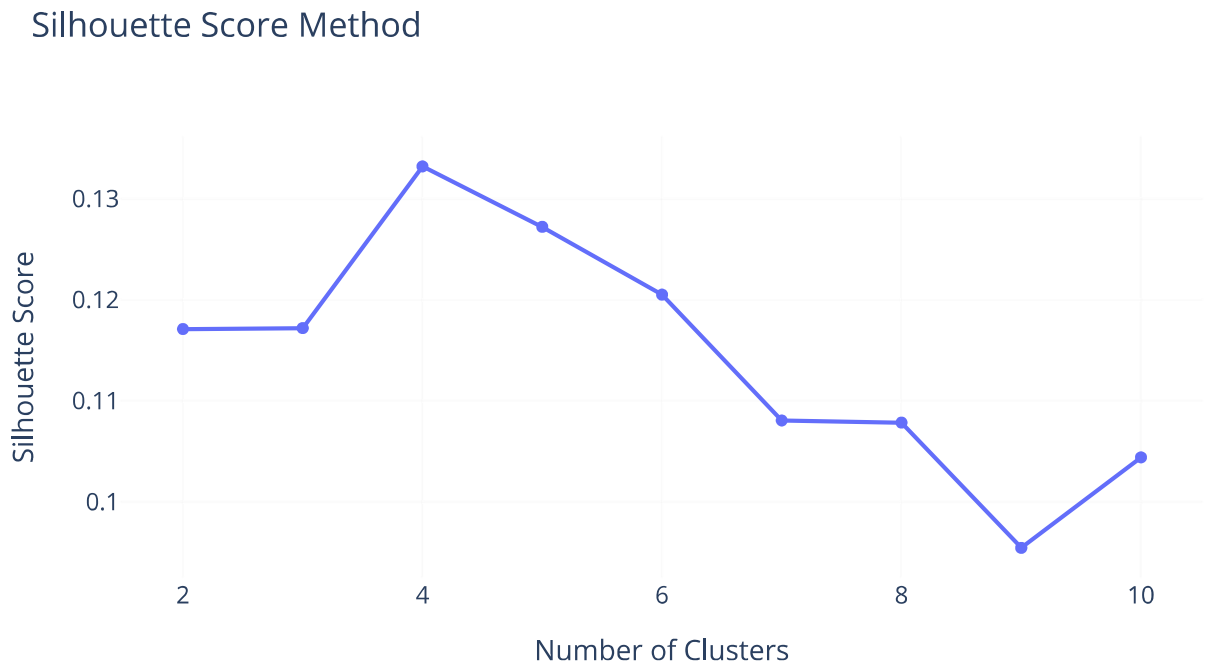
Observations

- We can see 3 groups of custommers around under 5K, around 7K, and 15K following total transaction amount.
- A significant number of attrited customers have high credit limit but low transaction volumes. This indicates that simply offering high credit limits may not be sufficient to retain customers if they are not actively using their accounts.
- A considerable of ACs have transaction amounts around 15K but it did not appear for ECs.

In [13]:
```
# Calculate churn and retention rates
churn_rate = df['attrition_flag'].eq('Attrited Customer').mean()
retention_rate = 1 - churn_rate

print("Churn Rate:", churn_rate)
print("Retention Rate:", retention_rate)
```

```
Churn Rate: 0.1606596227905599
Retention Rate: 0.8393403772094401
```

In [14]:
```python
# Standardize relevant features
features = ['customer_age','dependent_count','months_on_book','total_relationship_count',
            'months_inactive_12_mon','contacts_count_12_mon','credit_limit',
            'total_revolving_bal','avg_open_to_buy','total_amt_chng_q4_q1',
            'total_trans_amt','total_trans_ct','total_ct_chng_q4_q1','avg_utilization_ratio']
df_scaled = StandardScaler().fit_transform(df[features])

# Calculate silhouette scores for different numbers of clusters
silhouette_scores = [silhouette_score(df_scaled, KMeans(n_clusters=k, random_state=42).fit_predict(df_scaled)) for k in range(

# Plot silhouette scores
fig = px.line(
    x=range(2, 11),
    y=silhouette_scores,
    title='Silhouette Score Method',
    labels={'x': 'Number of Clusters', 'y': 'Silhouette Score'},
    markers=True,
    template='plotly_white',
    height=400,
    width=700
)
fig.show()
```



Observations

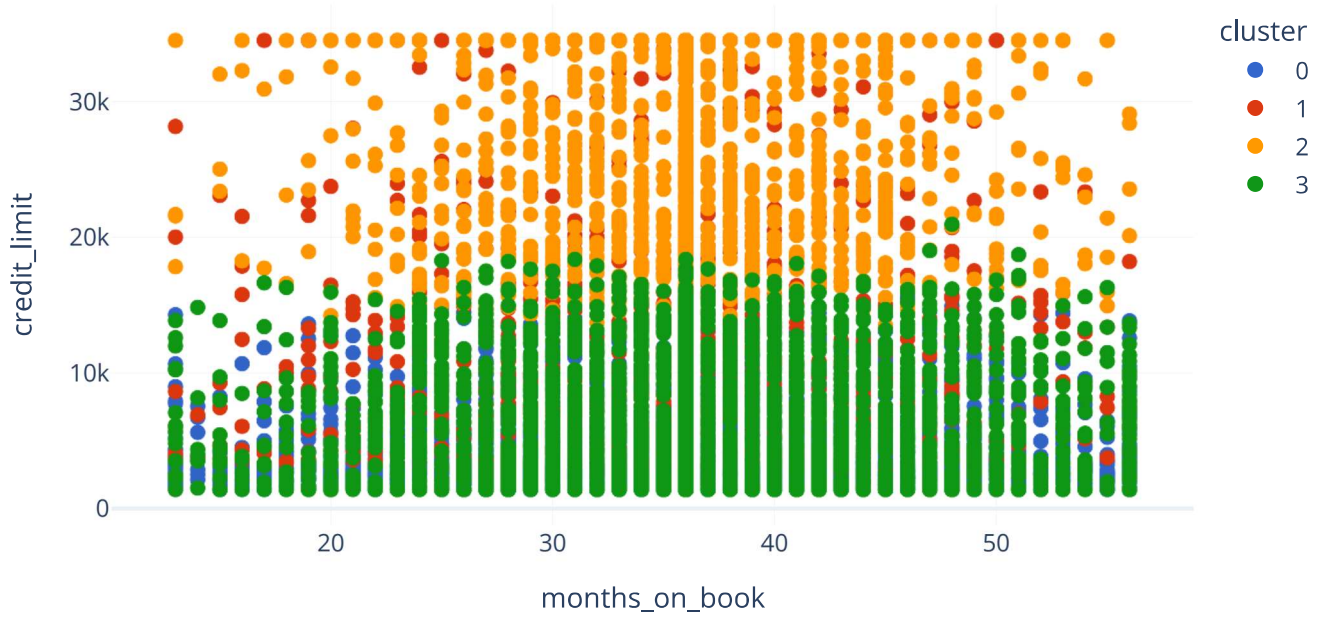
Choose the number of clusters is 4.

In [15]:
```
# Perform K-means clustering
kmeans = KMeans(n_clusters=4, random_state=42, n_init=10)
df['cluster'] = kmeans.fit_predict(df_scaled)
print(df.head())
```

```
   clientnum     attrition_flag  customer_age gender  dependent_count  \
0  768805383  Existing Customer            45      M                3
1  818770008  Existing Customer            49      F                5
2  713982108  Existing Customer            51      M                3
3  769911858  Existing Customer            40      F                4
4  709106358  Existing Customer            40      M                3

  education_level marital_status income_category card_category  \
0     High School        Married     $60K - $80K          Blue
1        Graduate         Single  Less than $40K          Blue
2        Graduate        Married    $80K - $120K          Blue
3     High School        Unknown  Less than $40K          Blue
4      Uneducated        Married     $60K - $80K          Blue

   months_on_book  ...  contacts_count_12_mon  credit_limit  \
0              39  ...                      3       12691.0
1              44  ...                      2        8256.0
2              36  ...                      0        3418.0
3              34  ...                      1        3313.0
4              21  ...                      0        4716.0

   total_revolving_bal  avg_open_to_buy  total_amt_chng_q4_q1  \
0                  777          11914.0                 1.335
1                  864           7392.0                 1.541
2                    0           3418.0                 2.594
3                 2517            796.0                 1.405
4                    0           4716.0                 2.175

   total_trans_amt  total_trans_ct  total_ct_chng_q4_q1  \
0             1144              42                1.625
1             1291              33                3.714
2             1887              20                2.333
3             1171              20                2.333
4              816              28                2.500

   avg_utilization_ratio  cluster
0                  0.061        0
1                  0.105        0
2                  0.000        0
3                  0.760        0
4                  0.000        0
```

[5 rows x 22 columns]

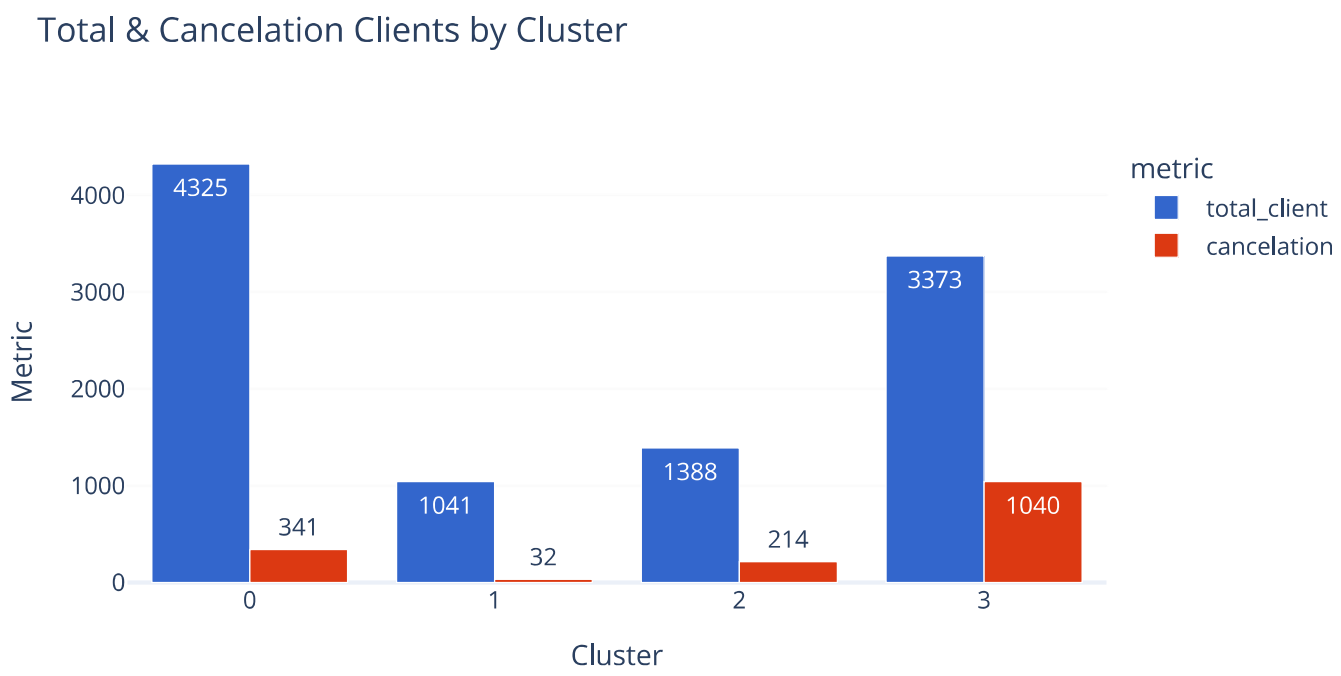
In [16]:
```
#Convert 'cluster' to string and sort DataFrame
df['cluster'] = df['cluster'].astype(str)
df = df.sort_values(by='cluster')

#Create scatter plot
fig = px.scatter(df, x='months_on_book', y='credit_limit', color='cluster',
                 template='plotly_white', height=400, width=700,
                color_discrete_sequence=px.colors.qualitative.G10)

fig.update_traces(marker_size=8)
fig.show()
```



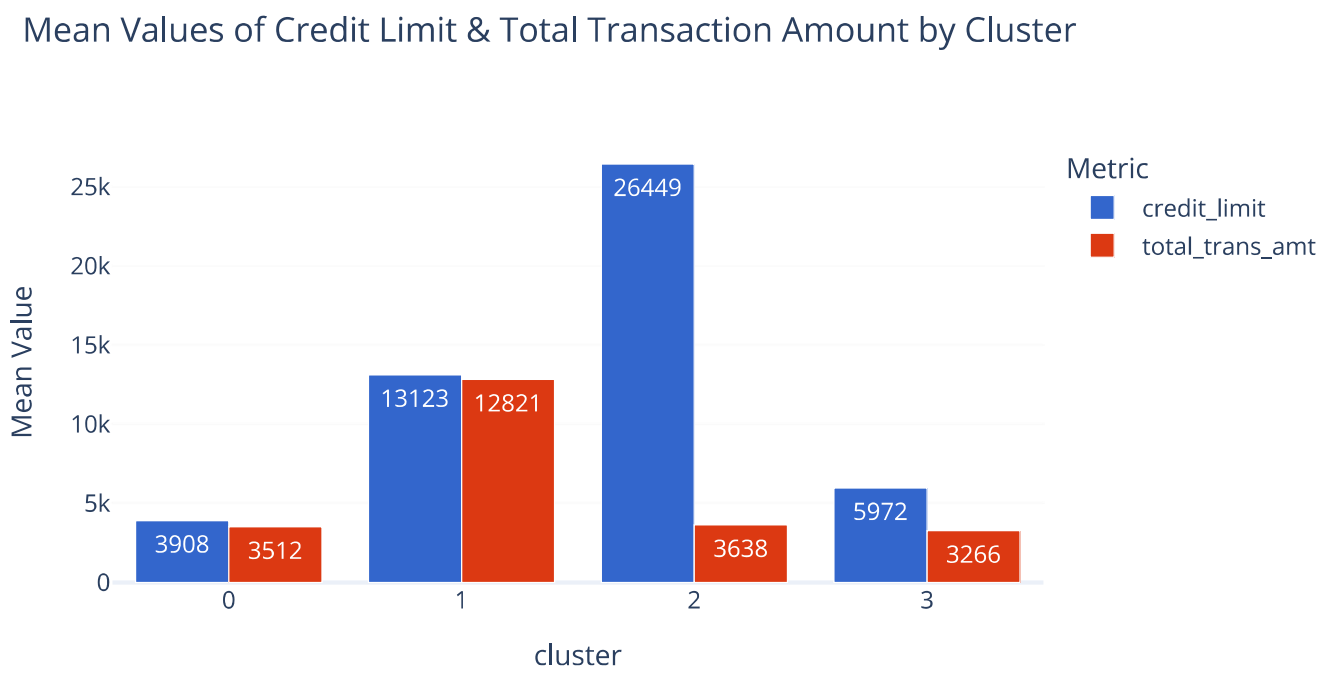
Observations

- Higher credit limits tend to be more prevalent in Clusters 1 and 2, while lower credit limits are more common in Cluster 0 and 3.
- There is no clear linear relationship between 'Months on Book' and 'Credit Limit' within each cluster.

In [17]:
```
# Group by 'cluster' and calculate 'total' and 'cancelation' in one step
cluster_summary = df.groupby('cluster').agg(
    total_client=('attrition_flag', 'size'),
    cancelation=('attrition_flag', lambda x: (x == 'Attrited Customer').sum())
).reset_index().melt(id_vars='cluster', var_name='metric', value_name='value')

# Plot the percentage of Attrited Customers by cluster
fig = px.bar(cluster_summary, x='cluster', y='value', text='value', color='metric', barmode='group',
             title='Total & Cancelation Clients by Cluster', template='plotly_white',
             color_discrete_sequence=px.colors.qualitative.G10)

fig.update_layout(height=400, width=700, xaxis_title='Cluster', yaxis_title='Metric')
fig.show()
```



In [18]:
```
# Create a new DataFrame with client numbers and their corresponding cluster labels
client_cluster_labels = df[['credit_limit','total_trans_amt','cluster']]

# Group by cluster and calculate mean values for income_category, credit_limit, and customer_age
cluster_summary = client_cluster_labels.groupby('cluster').agg(
    {'credit_limit': 'mean', 'total_trans_amt': 'mean'}
    ).round().reset_index().melt(id_vars='cluster', var_name='Metric', value_name='Mean Value')

fig= px.bar(cluster_summary, x='cluster', y='Mean Value', color='Metric', text='Mean Value', barmode='group',
            color_discrete_sequence=px.colors.qualitative.G10, template='plotly_white',
            height=400, width=700, title='Mean Values of Credit Limit & Total Transaction Amount by Cluster')
fig.show()
```

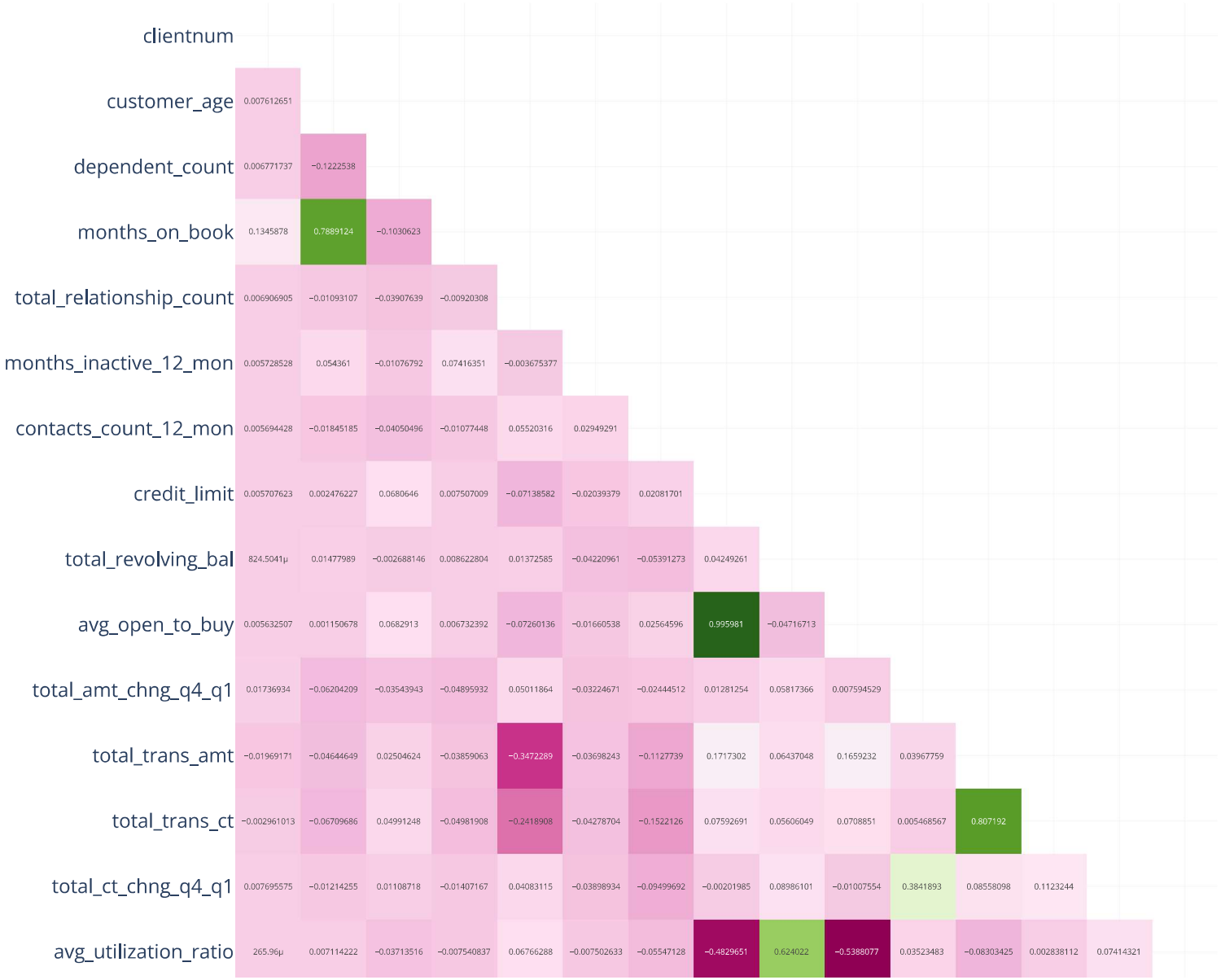


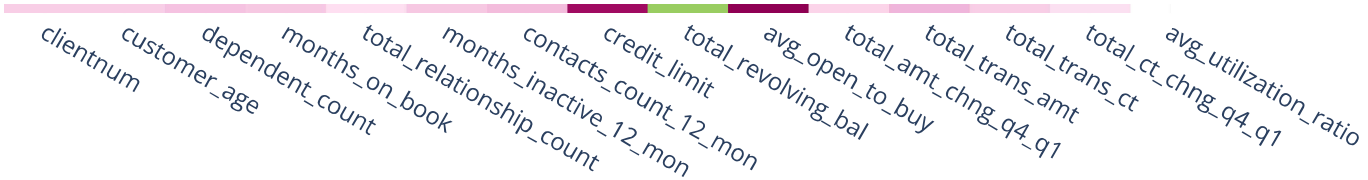
Observation

- Cluster 0: Represents low-value customers with the largest client base. Focus on minimizing churn through cost-effective retention strategies like personalized communication or incentives.
- Cluster 1: Shows the highest transaction amounts. Offering personalized services and loyalty programs could strengthen retention. Additionally, analyzing and potentially increasing credit limits might enhance revenue potential.
- Cluster 2: Has high potential with a large credit limit but moderate transactions. Engaging these customers with targeted promotions or incentives to increase spending could unlock further value.
- Cluster 3: Exhibits the highest churn rate. Address this with detailed churn analysis, including transaction trends, engagement metrics, and customer feedback to devise specific loyalty and retention strategies.

In [19]:
```
# Compute correlation matrix for numeric columns
corr = df.select_dtypes(include=["number"]).corr()

# Mask the upper triangle
mask = np.triu(np.ones_like(corr, dtype=bool))
corr = corr.mask(mask)

# Plot the heatmap
fig = px.imshow(corr, text_auto=True, color_continuous_scale="PiYG",
    title="Correlation Matrix for Numeric Columns (Lower Triangle Only)", template="plotly_white", height=800, width=1000)
fig.show()
```

Correlation Matrix for Numeric Columns (Lower Triangle Only)





Observation

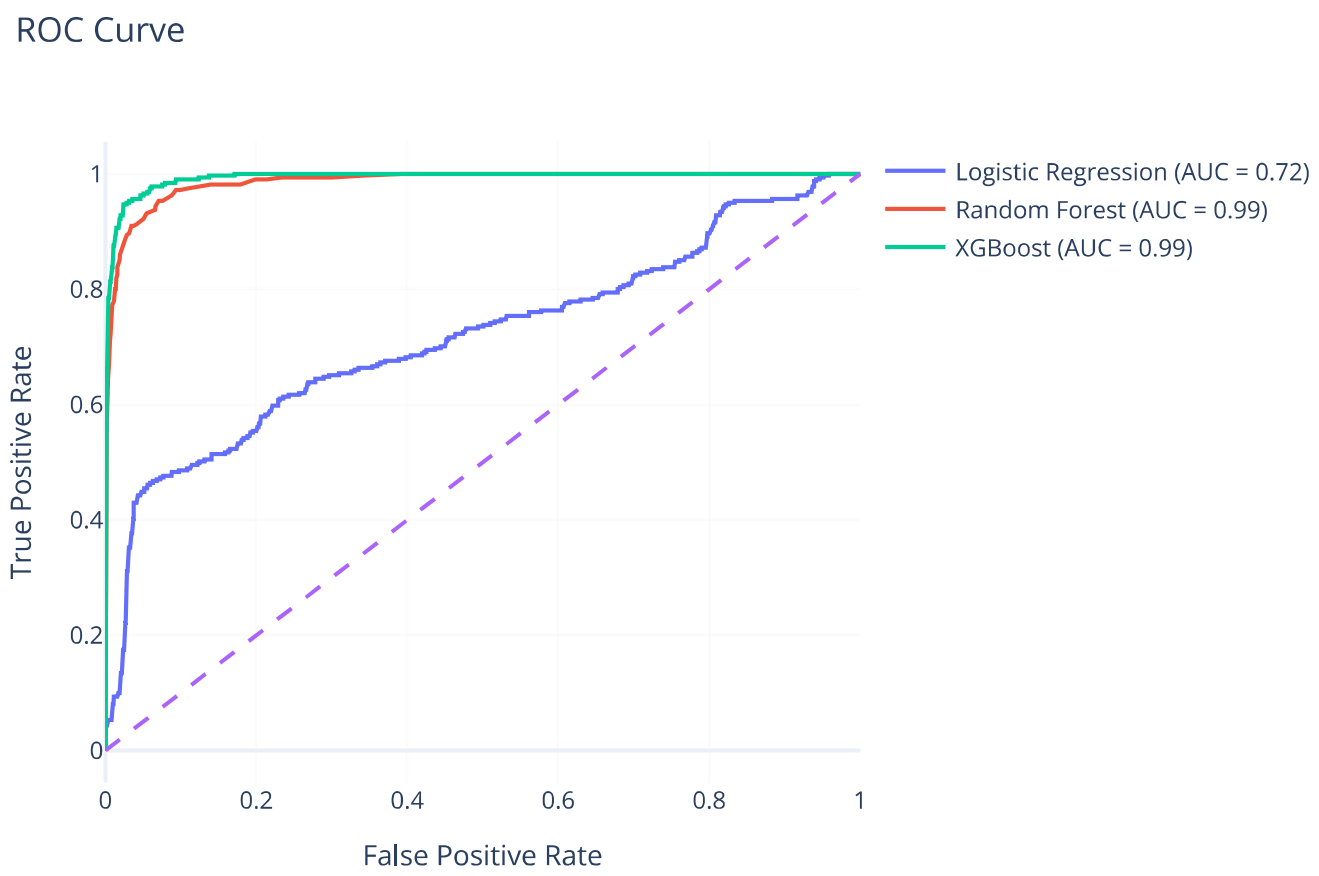
- Correlation Matrix chart shows that "Avg_Open_To_Buy" and "Credit_Limit" features have highest correlated (0.99). They provide redundant information. Keeping both in the model does not add value and can lead to multicollinearity issue.
- Credit_Limit feature is easily understandable feature for stakeholders, which enhances the practical applicability of our model's insight.
- We choose to drop Avg_Open_To_Buy feature.

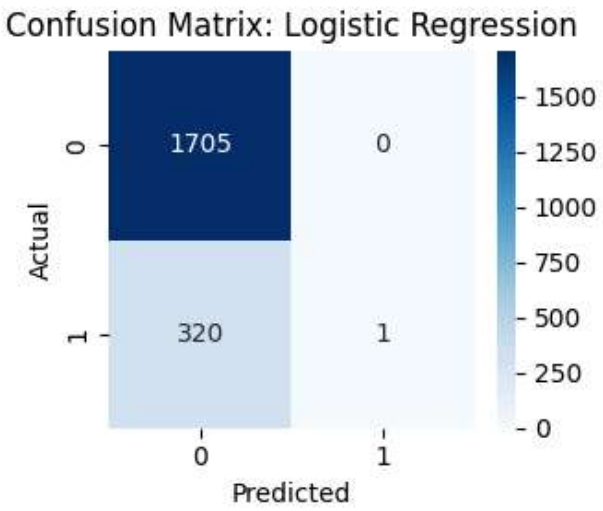
In [20]:
```
# Define features and target
X = df.drop(columns=["attrition_flag", "avg_open_to_buy"])
y = df["attrition_flag"].map({"Attrited Customer": 1, "Existing Customer": 0})

# One-hot encode categorical variables
X = pd.get_dummies(X)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize models
models = {
    "Logistic Regression": LogisticRegression(max_iter=1000),
    "Random Forest": RandomForestClassifier(),
    "XGBoost": XGBClassifier(eval_metric="logloss"),
}

# Train models and get predictions and probabilities
probs = {}
preds = {}
for name, model in models.items():
    model.fit(X_train, y_train)
```

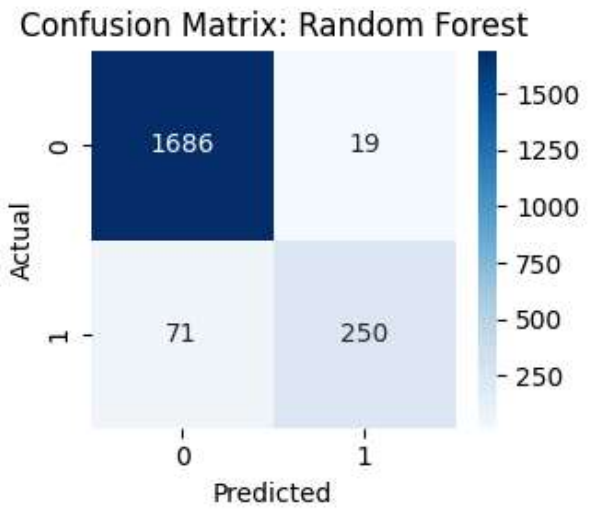
```
    probs[name] = model.predict_proba(X_test)[:, 1]
    preds[name] = model.predict(X_test)
```

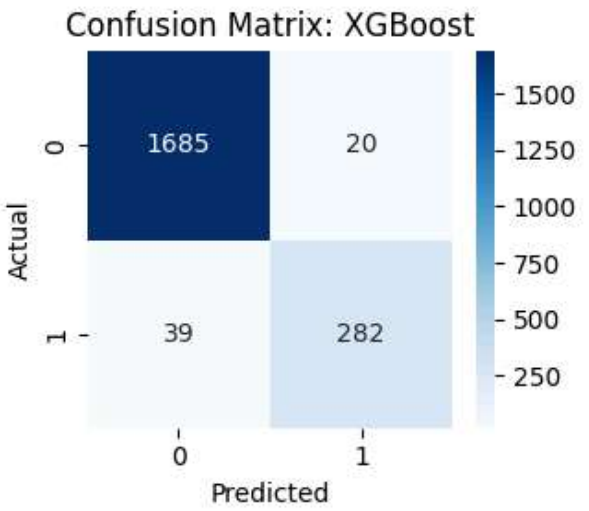
In [21]:

```
# Plot ROC curves
fig = go.Figure([
    go.Scatter(x=fpr, y=tpr, mode="lines", name=f"{name} (AUC = {roc_auc_score(y_test, prob):.2f})")
    for name, prob in probs.items()
    for fpr, tpr, _ in [roc_curve(y_test, prob)]
])
fig.add_trace(go.Scatter(x=[0, 1], y=[0, 1], mode="lines", line=dict(dash="dash"), showlegend=False))
fig.update_layout(title="ROC Curve", xaxis_title="False Positive Rate", yaxis_title="True Positive Rate",
                  height=500, width=700, template="plotly_white")
fig.show()

# Plot cofusion matrices
fig, axes = plt.subplots(1, 3, figsize=(10,3))
for ax, (model_name, y_pred) in zip(axes, preds.items()):
    sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt="d", cmap="Blues", ax=ax)
    ax.set_title(f"Confusion Matrix: {model_name}")
    ax.set_xlabel("Predicted")
    ax.set_ylabel("Actual")

plt.tight_layout()
plt.show()

# Print classification reports
for model_name, y_pred in preds.items():
    print(f"\nClassification Report for {model_name}")
    print(classification_report(y_test, y_pred))
```


ROC Curve
True Positive Rate
False Positive Rate
0
0.2
0.4
0.6
0.8
1
Logistic Regression (AUC = 0.72)
Random Forest (AUC = 0.99)
XGBoost (AUC = 0.99)


Confusion Matrix: Logistic Regression
Actual
Predicted
0
1
1705
0
320
1
1500
1250
1000
750
500
250
0


Confusion Matrix: Random Forest
Actual
Predicted
0
1
1686
19
71
250
1500
1250
1000
750
500
250


Confusion Matrix: XGBoost
Actual
Predicted
0
1
1685
20
39
282
1500
1250
1000
750
500
250

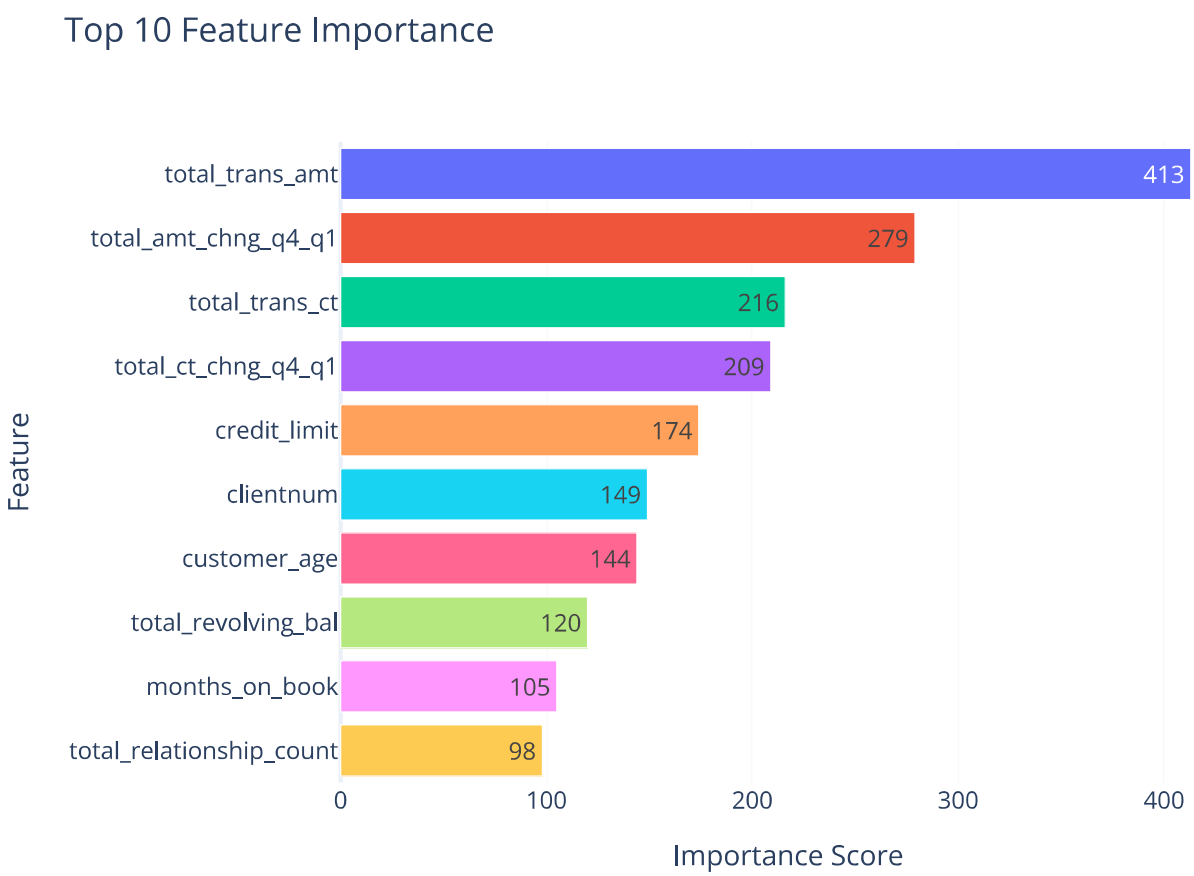
```
Classification Report for Logistic Regression
              precision    recall  f1-score   support

           0       0.84      1.00      0.91      1705
           1       1.00      0.00      0.01       321

    accuracy                           0.84      2026
   macro avg       0.92      0.50      0.46      2026
weighted avg       0.87      0.84      0.77      2026


Classification Report for Random Forest
              precision    recall  f1-score   support

           0       0.96      0.99      0.97      1705
           1       0.93      0.78      0.85       321

    accuracy                           0.96      2026
   macro avg       0.94      0.88      0.91      2026
weighted avg       0.95      0.96      0.95      2026


Classification Report for XGBoost
              precision    recall  f1-score   support

           0       0.98      0.99      0.98      1705
           1       0.93      0.88      0.91       321

    accuracy                           0.97      2026
   macro avg       0.96      0.93      0.94      2026
weighted avg       0.97      0.97      0.97      2026
```

Observation

- Based on the comprehensive comparison of key performance metrics, XGBoost outperforms both Logistic Regression and Random Forest models in most of rates.
- XGBoost has the lowest number of false negatives, which is particularly important for identifying churned customers accurately.
- Therefore, the XGBoost model is chosen for the final deployment in the churn prediction project.

In [22]:

```
# Training an XGBoost classification model
xgb_model = XGBClassifier(eval_metric='logloss')
xgb_model.fit(X_train, y_train)

# Extract feature importance scores
importance_scores = xgb_model.get_booster().get_score(importance_type='weight')

# Convert the scores to a DataFrame for better readability
importance_df = pd.DataFrame(importance_scores.items(), columns=['Feature', 'Importance']).sort_values(by='Importance', ascend

# Plot feature importance using Plotly
fig = px.bar(importance_df.head(10),
             x='Importance', y='Feature',
             title='Top 10 Feature Importance',
             template='plotly_white',
             color=px.colors.qualitative.G10,
            text_auto=True)

fig.update_layout(
    xaxis_title='Importance Score',
    yaxis_title='Feature',
    yaxis={'categoryorder':'total ascending'},
    height=500,
    width=700,
    showlegend=False)
fig.show()
```

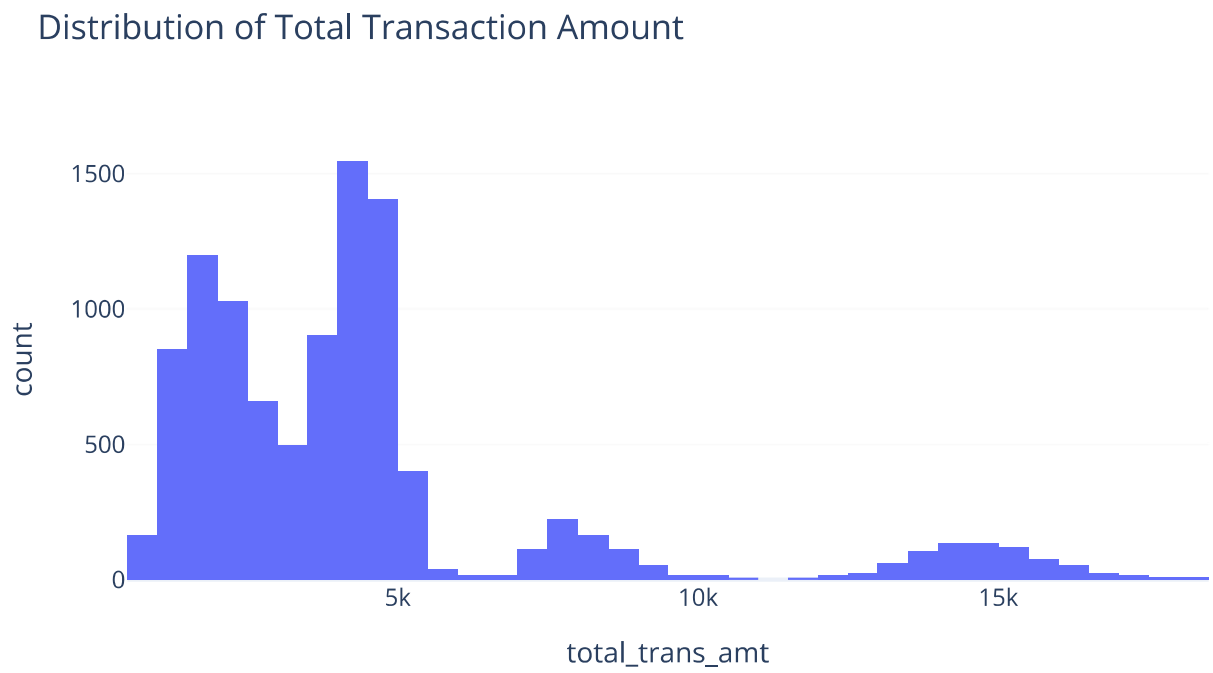


Observation

- From top 10 feature importance, we can develop targeted strategies to improve customer retention and satisfaction.
- For example:
  - Total Transaction Amount (Total_Trans_Amt): High transaction amounts indicate active usage of services, which correlates with customer retention.

- Transaction Amount from Q4 to Q1 (Total_amt_chng_q4_q1): A significant changes in transaction amounts can signal changes in customer behavior or satisfaction.
- Total Revolving Balance (Total_Revolving_Bal): High revolving balances might indicate financial stress, which could lead to churn.

In [23]:
```
# Count number of customer by Total Transaction Amount
fig = px.histogram(df, x='total_trans_amt', nbins=50, title='Distribution of Total Transaction Amount', template='plotly_white
fig.show()
```



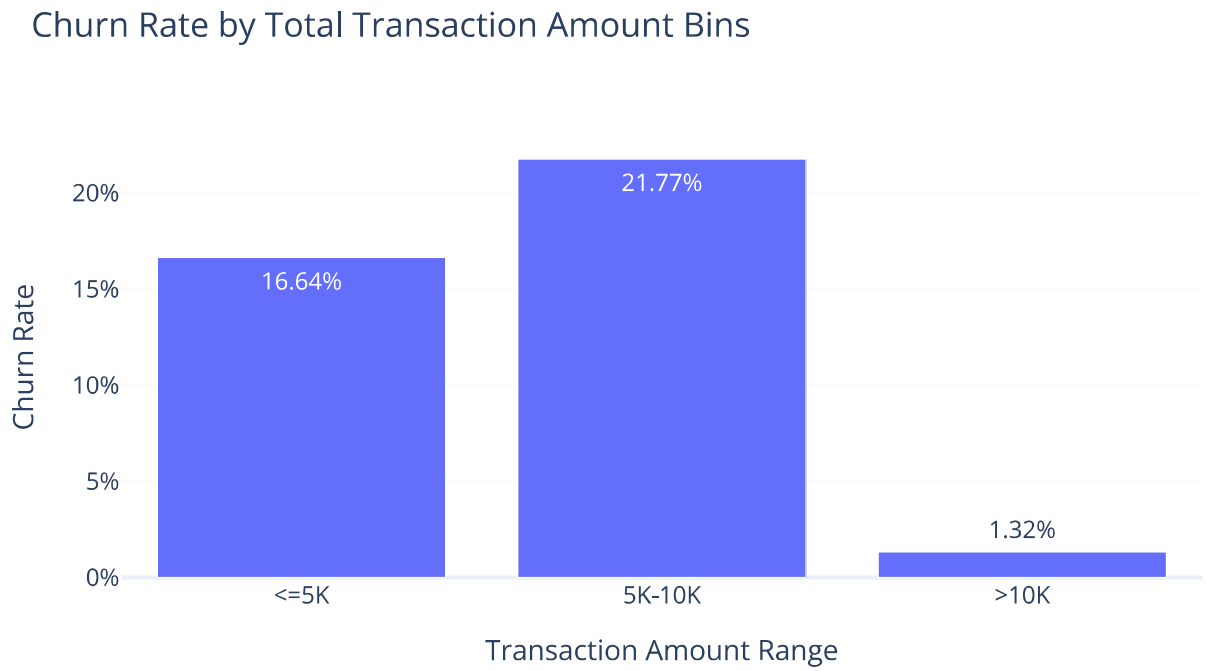
Observation

From this chart, we determine that we have 3 customer groups by total transaction amount that:

- <= 5k

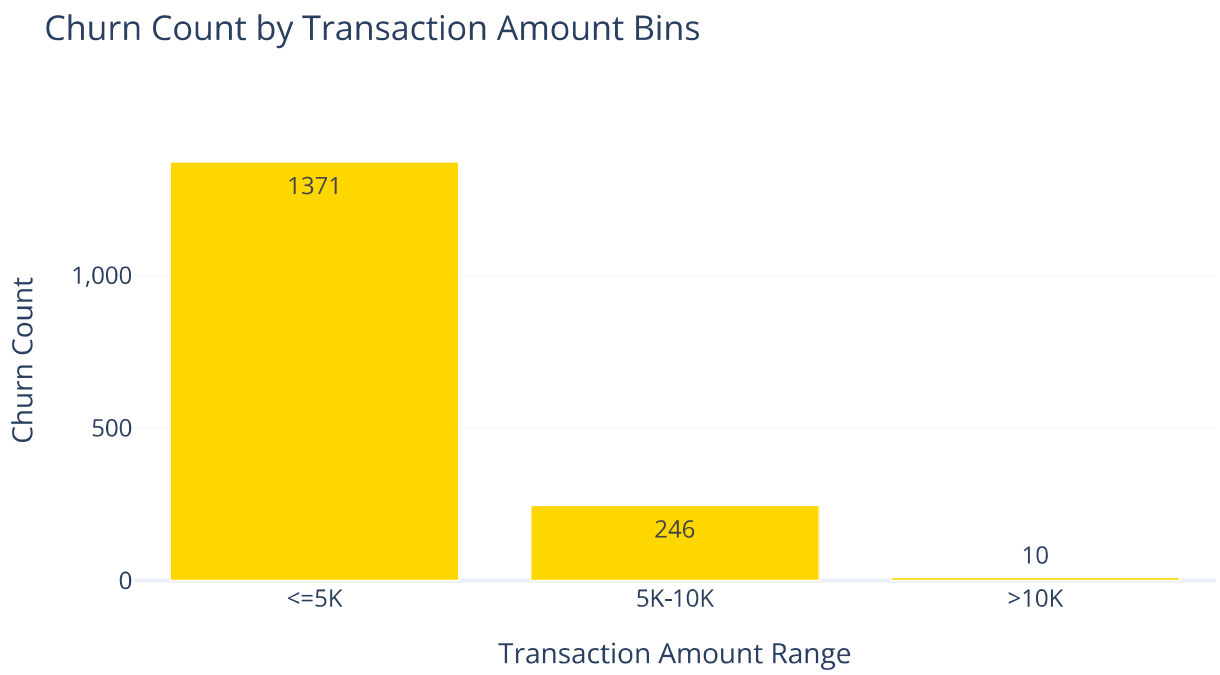
- 5K-10K
- >10k

In [24]:
```
# Calculate churn rate and group values
churn_rate = df.groupby(pd.cut(df['total_trans_amt'], bins=[0, 5000, 10000, df['total_trans_amt'].max()],
                               labels=['<=5K', '5K-10K', '>10K']), observed=False)['attrition_flag'] \
               .apply(lambda x: (x == 'Attrited Customer').mean()) \
               .reset_index(name='churn_rate')

# Plot the chart
fig = px.bar(churn_rate, x='total_trans_amt', y='churn_rate', height=400, width=700,
             title='Churn Rate by Total Transaction Amount Bins',
             text_auto='.2%', template='plotly_white')

fig.update_layout(xaxis_title='Transaction Amount Range',
                  yaxis_title='Churn Rate',
                  yaxis_tickformat='.0%')
fig.show()
```



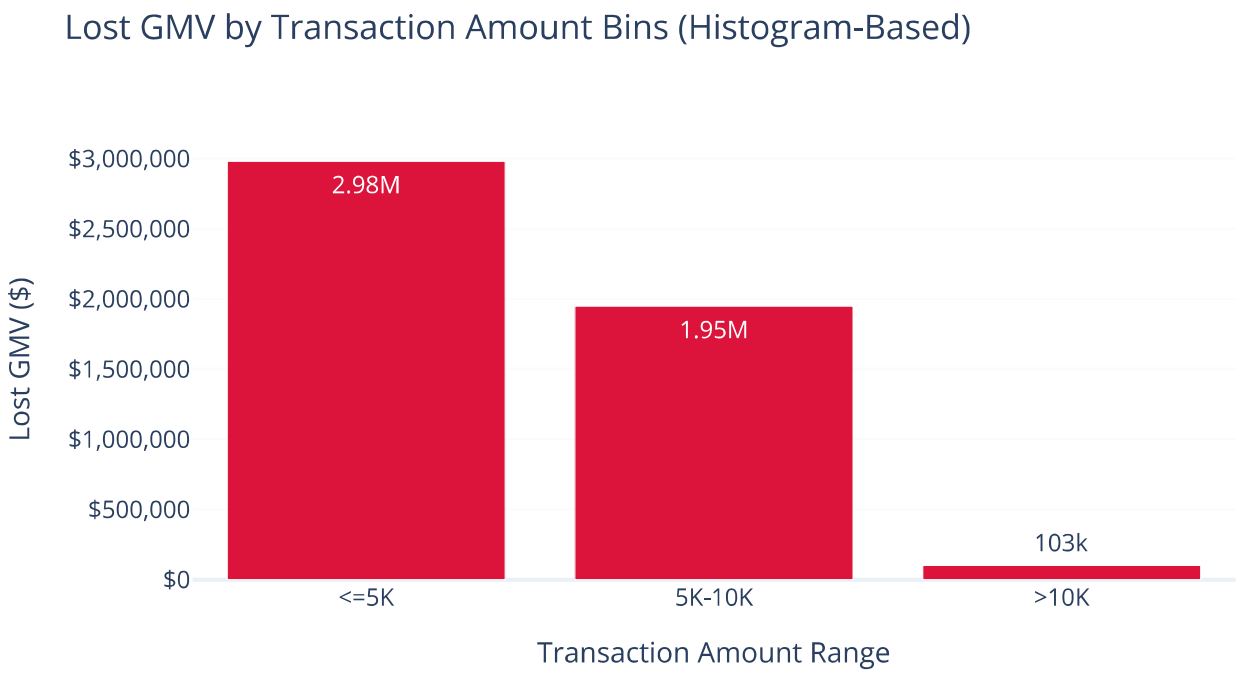
In [25]:
```
# Create trans_amt_bin column
df['trans_amt_bin'] = pd.cut(df['total_trans_amt'],
                             bins=[0, 5000, 10000, df['total_trans_amt'].max()],
                             labels=['<=5K', '5K-10K', '>10K'],
                             include_lowest=True)

# Calculate churn by group
churn_count = df[df['attrition_flag'] == 'Attrited Customer'] \
    .groupby('trans_amt_bin', observed=False)['attrition_flag'] \
    .count().reset_index(name='churn_count')

# Plot the chart
import plotly.express as px
fig = px.bar(churn_count, x='trans_amt_bin', y='churn_count', height=400, width=700,
             title='Churn Count by Transaction Amount Bins',
             text_auto='.0f', template='plotly_white')
```

```
fig.update_traces(marker_color='gold')
fig.update_layout(xaxis_title='Transaction Amount Range',
                  yaxis_title='Churn Count',
                  yaxis_tickformat=',')
fig.show()
```



In [26]:
```
# Calculate GMV lose by group
lost_gmv = df[df['attrition_flag'] == 'Attrited Customer'] \
    .groupby('trans_amt_bin', observed=False)['total_trans_amt'] \
    .sum().reset_index()

lost_gmv.columns = ['trans_amt_bin', 'lost_gmv']

# Plot the chart
fig = px.bar(lost_gmv, x='trans_amt_bin', y='lost_gmv', height=400, width=700,
             title='Lost GMV by Transaction Amount Bins (Histogram-Based)',
```

```
             text_auto='.3s', template='plotly_white')
fig.update_traces(marker_color='crimson')
fig.update_layout(xaxis_title='Transaction Amount Range',
                  yaxis_title='Lost GMV ($)',
                  yaxis_tickformat='$,.0f')
fig.show()
```



Observation

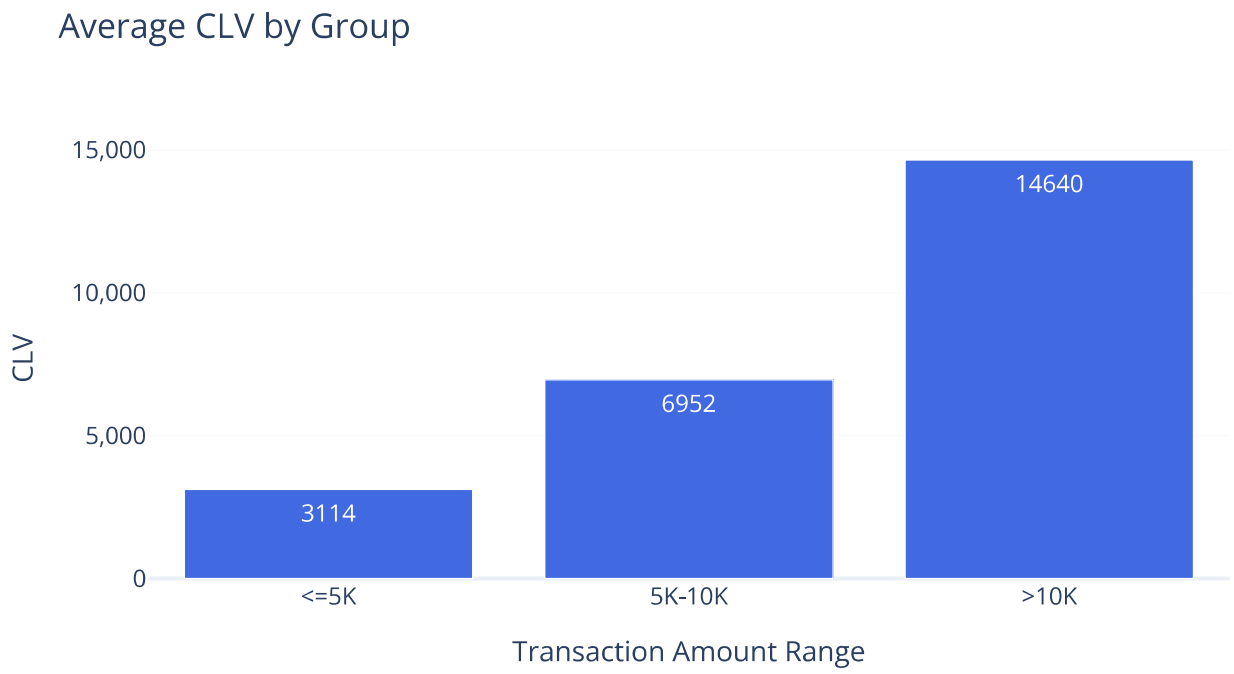
The <=5K group has higher churn (1371 customers, $2.98M GMV loss) compared to 5K-10K (246 customers, $1.95M GMV loss). However, 5K-10K customers have a greater GMV impact per churn ($7926 vs. $2175, 3.6x higher), a higher churn rate (21.77% vs. 16.64%), and a more cost-efficient retention opportunity (65% of GMV loss with only 18% of churn volume).

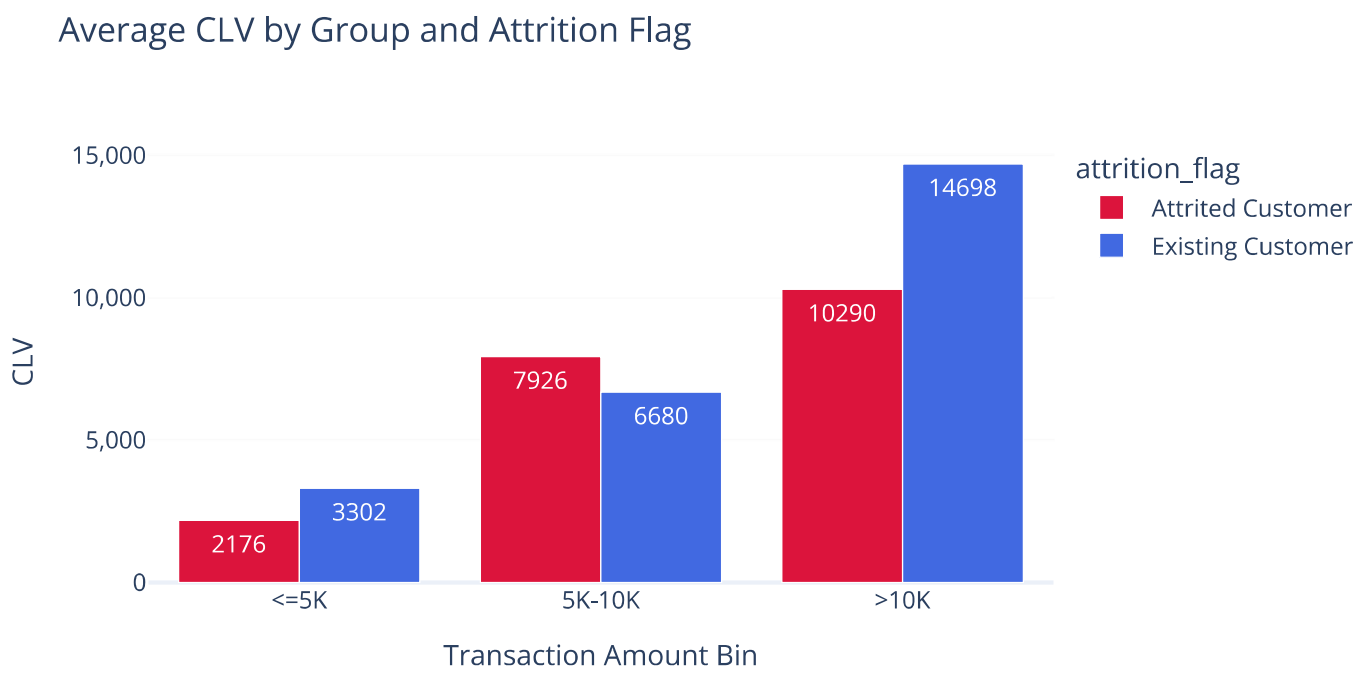
Recommendations:

- **For the 5K-10K group (High-Value, High-Risk):** Priortities retention strategies, focusing on personalized incentives (cashback, loyalty programs, exclusive offers, fee waivers) predictive analytics to identify at-risk customers (declining transaction frequency, occasionally using service) and engage them.
- **For the <=5K group (High-Volume, Moderate-Risk):** Implement scalable, low-cost retention strategies to address the high churn volume, such as automated email/sms campaigns with small incentives ($5 credit on their next transaction) or referral programs.
- **For the >10K group (High-Value, Low-Risk):** Maintain interaction, focus on loyalty programs and enhance personalization.

In [27]:
```
# Calculate Average tracsaction value
df['avg_transaction_value'] = df['total_trans_amt'] / df['total_trans_ct']

# Customer lifespan by years
df['lifespan_years'] = df['months_on_book'] / 12

# Transaction frequency per year
df['transaction_frequency_per_year'] = df['total_trans_ct'] / df['lifespan_years']

# Calculate CLV
df['clv'] = df['avg_transaction_value'] * df['transaction_frequency_per_year'] * df['lifespan_years']

# Remove NaN or infinity values (if months_on_book = 0 or total_trans_ct = 0)
df = df.dropna(subset=['clv']).replace([float('inf'), -float('inf')], 0)

# Plot Average CLV by Group
clv_summary = df.groupby('trans_amt_bin', observed=False)['clv'].mean().reset_index()

fig = px.bar(clv_summary, x='trans_amt_bin', y='clv', height=400, width=700,
             title='Average CLV by Group', text_auto='.0f', template='plotly_white')
fig.update_traces(marker_color='royalblue')
fig.update_layout(xaxis_title='Transaction Amount Range',
                  yaxis_title='CLV',yaxis_tickformat=',')
fig.show()

# Plot Average CLV by Group and Attrition Flag
clv_by_attrition = df.groupby(['trans_amt_bin', 'attrition_flag'], observed=False)['clv'].mean().round().reset_index()
color_map = {'Attrited Customer': 'crimson', 'Existing Customer': 'royalblue'}

fig= px.bar(clv_by_attrition, x='trans_amt_bin', y='clv', color='attrition_flag', text='clv', barmode='group',
            template='plotly_white', height=400, width=700, title='Average CLV by Group and Attrition Flag', color_discrete_ma
```

```
fig.update_layout(xaxis_title='Transaction Amount Bin',
                  yaxis_title='CLV',
                  yaxis_tickformat=',')
fig.show()
```





Observation

- CLV by Group
  - **<=5K:** $3114 (Lowest CLV, potential for upsell)
  - **5K-10K:** $6952 (2.2x higher than <=5K group, high-value group)
  - **>10K:** $14640 (highest CLV, 2.1x higher than 5K-10K group)
- CLV by Group and Attrition:
  - **<=5K:** Attrited ($2176) vs Existing ($3302) - 34% lower CLV for attrited, indicating potential to increase CLV by retaining longer.
  - **5K-10K:** Attrited ($7926) vs Existing ($6980) - Atrited have 18.7% higher CLV, showing sighnificant loss when they churn.
  - **>10K:** Attrited ($10290) vs Existing ($14698) - 42.8% higher CLV for exisiting, but low churn number (7 customers).

Recommendations:

- **Prioritize 5K-10K group:** High CLV and high loss from attrited customers.
- **Target <=5K group for retention and upsell:** Low CLV but potential for retention. Use cost-effective campaign (offer credit by emails) and upsell programs (Reach $5000 for extra bonus) to move them to 5K-10K group.
- **Maintain >10K group:** Highest CLV, low churn. Sustain loyalty with VIP programs to protect long-term value.
- **Micro Analysis:**
  - Analyze transaction trends to segment customers with decrasing activity and derive retention insights, focusing on high-risk groups.
  - Apply the Intra-Group Segmentation or RFM model (Recency: months_inactive_12_mon, Frequency: total_trans_ct, Monetary: total_trans_amt) to identify high-risk and high-value target within each trans_amt_bin cluster for tailored strategies.