CLTC_RFM_Analysis

Customer Lifetime Value Analysis

Inroduction

Customer Lifetime Value Analysis

- Customer lifetime value (CLV) analysis estimates the total value of customers to the business throughout their entire relationship.
- It assists companies in making informed decisions about customer acquisition and retention investments.
- CLV analysis helps identify high-value customers, enabling businesses to prioritize retention efforts effectively.
- By analyzing CLV, companies can optimize marketing channels and campaigns for acquiring valuable customers.
- Targeted retention strategies can be developed based on CLV analysis to foster customer engagement and loyalty.
- The task of Customer Lifetime Value analysis requires a dataset containing information about customers' relationships with the business.

Importing libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px

import warnings
warnings.filterwarnings('ignore')
```

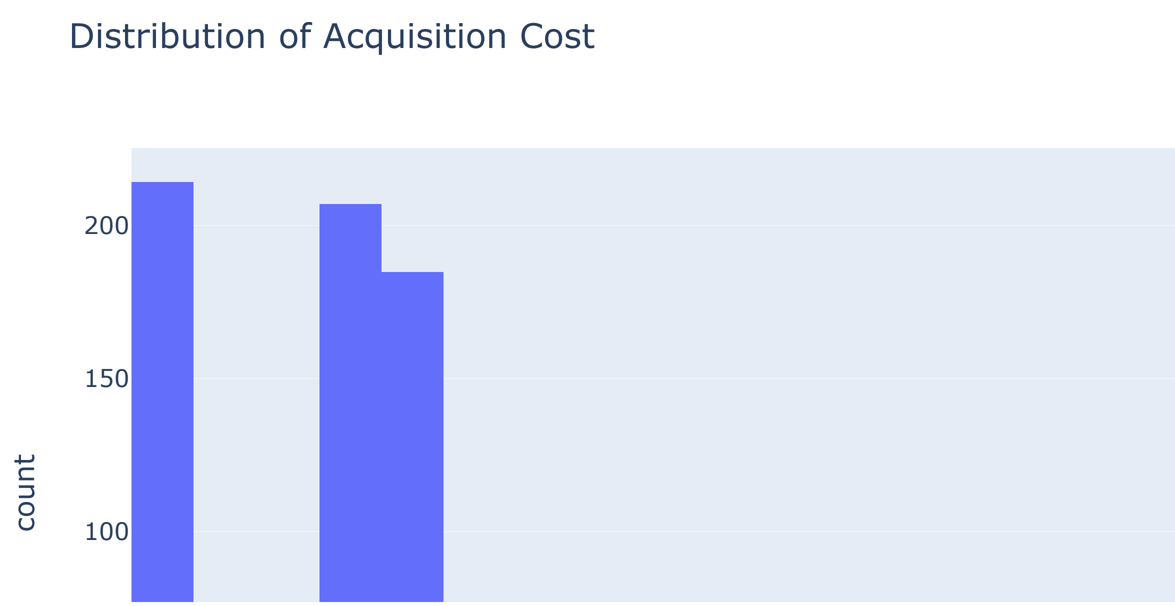
Importing dataset

In [2]:
```
data = pd.read_csv("customer_acquisition_data.csv")
print(data.head())
```

```
   customer_id            channel       cost  conversion_rate  revenue
0            1           referral   8.320327         0.123145     4199
1            2  paid advertising  30.450327         0.016341     3410
2            3    email marketing   5.246263         0.043822     3164
3            4       social media   9.546326         0.167592     1520
4            5           referral   8.320327         0.123145     2419
```

We start by visualizing the Distribution of Acquisition Cost and Customer Revenue with Histograms

In [3]:
```
fig = px.histogram(data,
                   x="cost",
                   nbins=50,
                   title='Distribution of Acquisition Cost')
fig.show()
```



Distribution of Revenue using Histogram

In [4]:
```
fig = px.histogram(data,
                   x="revenue",
                   nbins=50,
                   title='Distribution of Revenue')
fig.show()
```

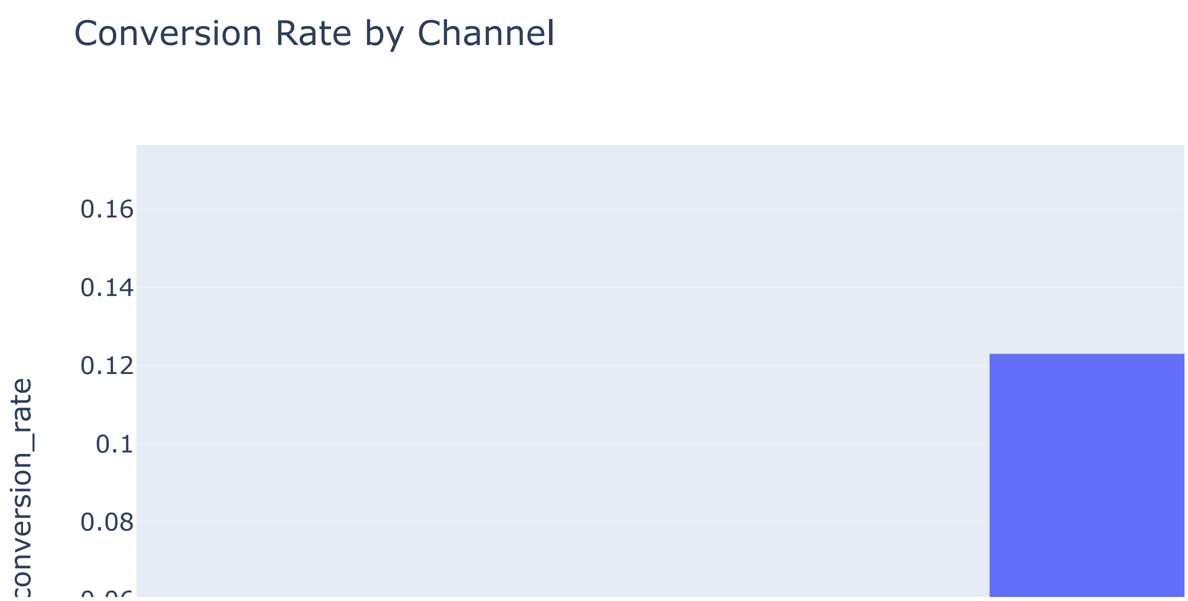
Comparing the Cost of Acquisition Across Different Channels and Identifying the Most and Least Profitable Channels

In [5]:
```
cost_by_channel = data.groupby('channel')['cost'].mean().reset_index()

fig = px.bar(cost_by_channel,
             x='channel',
             y='cost',
             title='Customer Acquisition Cost by Channel')
fig.show()
```

- Paid advertisement is the most expensive channel, while email marketing is the least expensive.

Let's now assess the effectiveness of different channels in converting customers.

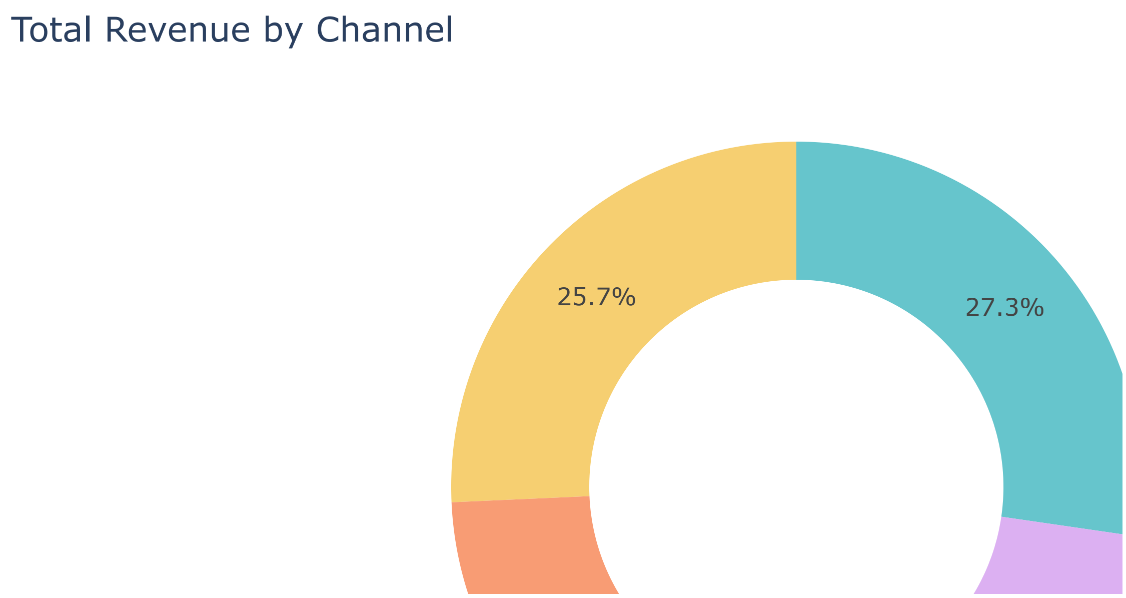
In [6]:
```
conversion_by_channel = data.groupby('channel')['conversion_rate'].mean(

fig = px.bar(conversion_by_channel, x='channel',
             y='conversion_rate',
             title='Conversion Rate by Channel')
fig.show()
```



- Social media stands out as the most effective channel for converting customers.
- On the other hand, paid advertising appears to be the least effective channel.

Next, we'll calculate the total revenue by channel to identify the most and least profitable channels in terms of revenue generation.

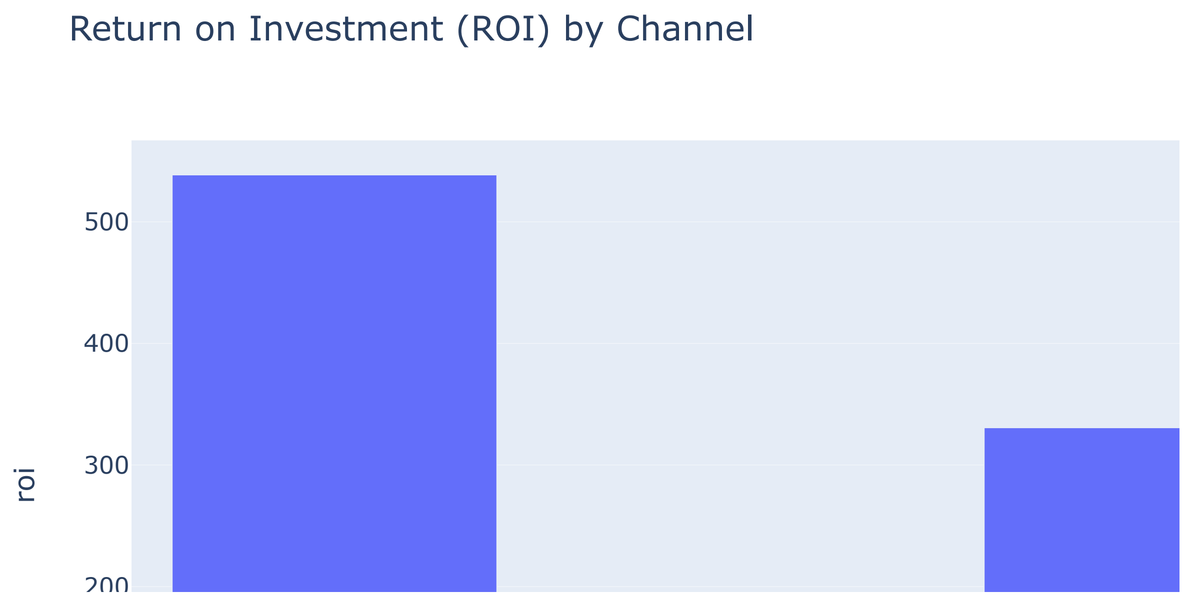
In [7]:
```
revenue_by_channel = data.groupby('channel')['revenue'].sum().reset_inde

fig = px.pie(revenue_by_channel,
             values='revenue',
             names='channel',
             title='Total Revenue by Channel',
             hole=0.6, color_discrete_sequence=px.colors.qualitative.Pas

fig.show()
```



- Email marketing stands out as the most profitable channel for revenue generation.
- While there are differences in revenue generation percentages across channels, no channel can be declared the least profitable.

Next, we will calculate the return on investment (ROI) for each channel to further assess their performance.

In [8]:
```
data['roi'] = data['revenue'] / data['cost']
roi_by_channel = data.groupby('channel')['roi'].mean().reset_index()

fig = px.bar(roi_by_channel,
             x='channel',
             y='roi', title='Return on Investment (ROI) by Channel')
fig.show()
```



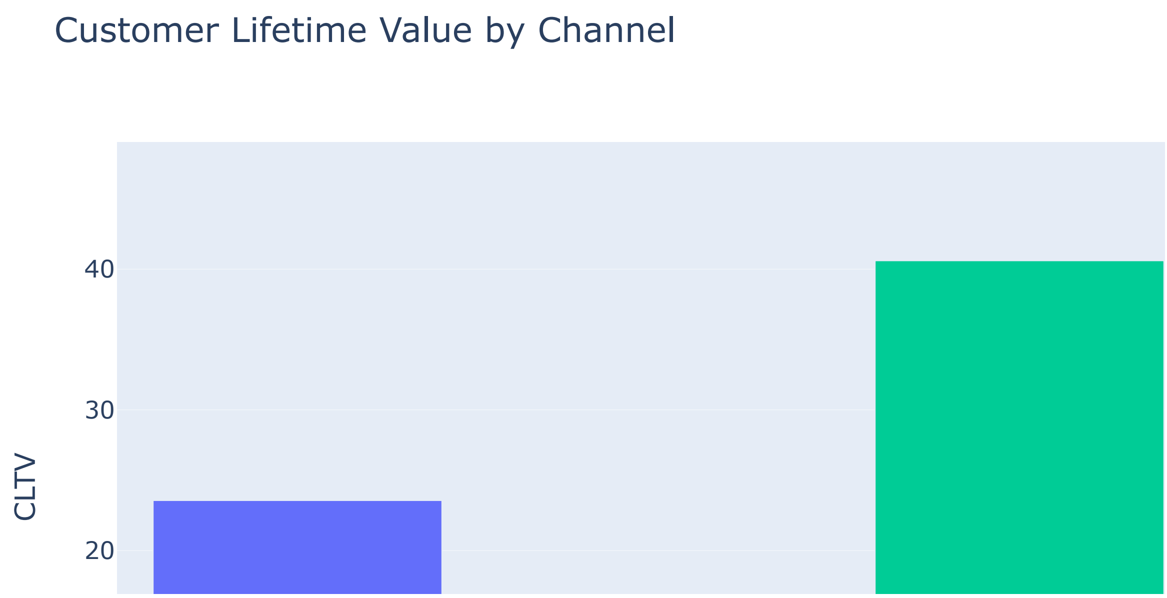
- The ROI from email marketing surpasses that of all other channels, making it highly effective.
- On the other hand, the ROI from paid advertising is the lowest among the channels.

Now, we will calculate the customer lifetime value (CLTV) for each channel.

Utilizing the data available, we can employ the provided formula to calculate CLTV.

CLTV = (revenue – cost) * conversion_rate / cost

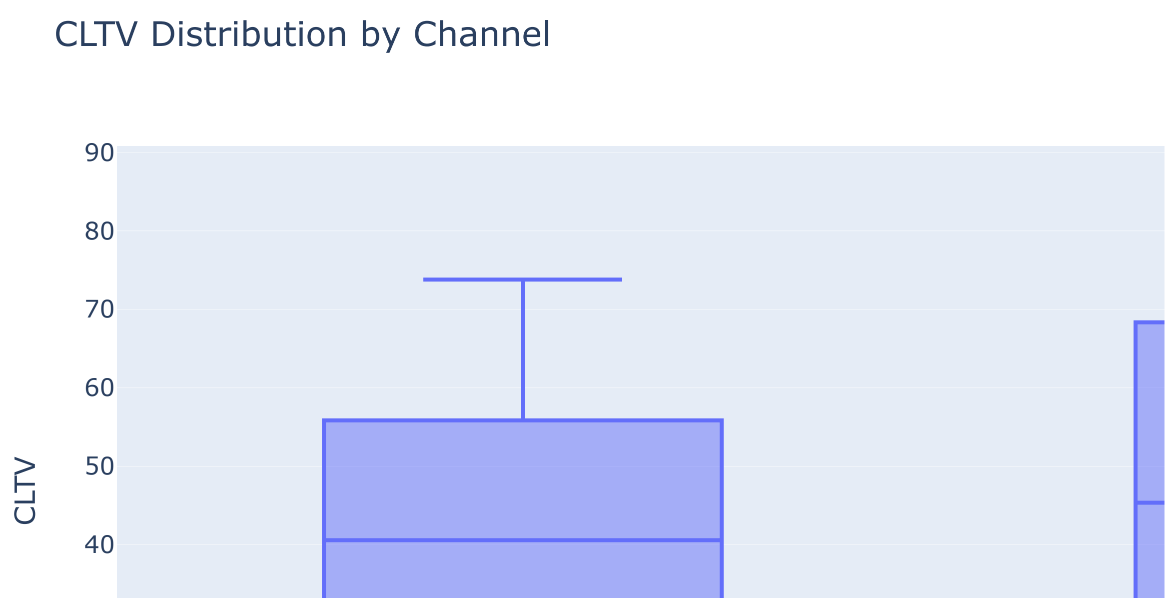
In [9]:
```
data['cltv'] = (data['revenue'] - data['cost']) * data['conversion_rate']

channel_cltv = data.groupby('channel')['cltv'].mean().reset_index()

fig = px.bar(channel_cltv, x='channel', y='cltv', color='channel',
             title='Customer Lifetime Value by Channel')

fig.update_xaxes(title='Channel')
fig.update_yaxes(title='CLTV')

fig.show()
```



- The customer lifetime value (CLV) from Social Media and referral channels is the highest.

Next, we will compare the CLTV distributions of the social media and referral channels.

In [10]:
```
subset = data.loc[data['channel'].isin(['social media', 'referral'])]

fig = px.box(subset, x='channel', y='cltv', title='CLTV Distribution by (

fig.update_xaxes(title='Channel')
fig.update_yaxes(title='CLTV')
fig.update_layout(legend_title='Channel')

fig.show()
```



- The Customer Lifetime Value from the Social Media channel is slightly better than the referral channel.

Summary

- Customer lifetime value analysis estimates the total value of customers to the business throughout their relationship.
- It aids companies in making investment decisions for customer acquisition and retention.
- Identifying the most valuable customers allows for prioritized retention efforts.
- This article provides insights into Customer Lifetime Value Analysis using Python.

RFM Analysis using Python

- RFM Analysis is a concept used in Data Science, particularly in marketing, to understand and segment customers based on their buying behavior.
- RFM stands for Recency, Frequency, and Monetary value, three key metrics that provide insights into customer engagement, loyalty, and value to a business.
- Recency refers to the date of a customer's last purchase, Frequency measures how often they make purchases, and Monetary value represents the amount spent on purchases.
- By performing RFM analysis using Python, we can use a dataset containing customer IDs, purchase dates, and transaction amounts to calculate RFM values for each customer and analyze their patterns and behaviors.

Importing libraries

In [11]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
from datetime import datetime
import plotly.graph_objects as go
```

Importing dataset

In [12]:
```python
data = pd.read_csv("rfm_data.csv")
print(data.head())
```

```
   CustomerID PurchaseDate  TransactionAmount ProductInformation  Order
ID  \
0        8814   2023-04-11             943.31          Product C   8900
75
1        2188   2023-04-11             463.70          Product A   1768
19
2        4608   2023-04-11              80.28          Product A   3400
62
3        2559   2023-04-11             221.29          Product A   2391
45
4        9482   2023-04-11             739.56          Product A   1945
45

   Location
0     Tokyo
1    London
2  New York
3    London
4     Paris
```

Calculating RFM Values

- Convert 'PurchaseDate' to datetime using `pd.to_datetime(data['PurchaseDate'])`.
- Calculate Recency by subtracting the current date from the 'PurchaseDate' and extracting days using `.dt.days`.
  - `data['Recency'] = (datetime.now().date() - data['PurchaseDate'].dt.date).dt.days`
- Calculate Frequency by grouping data by 'CustomerID' and counting the number of 'OrderID' for each customer.
  - `frequency_data = data.groupby('CustomerID')['OrderID'].count().reset_index()`
  - Rename the column to 'Frequency' using `frequency_data.rename(columns={'OrderID': 'Frequency'}, inplace=True)`.
  - Merge the frequency data back to the main data using `data = data.merge(frequency_data, on='CustomerID', how='left')`.
- Calculate Monetary Value by grouping data by 'CustomerID' and summing the 'TransactionAmount' for each customer.
  - `monetary_data = data.groupby('CustomerID')['TransactionAmount'].sum().reset_index()`
  - Rename the column to 'MonetaryValue' using `monetary_data.rename(columns={'TransactionAmount': 'MonetaryValue'}, inplace=True)`.
  - Merge the monetary data back to the main data using `data = data.merge(monetary_data, on='CustomerID', how='left')`.

These steps demonstrate how to perform RFM analysis in Python using the given dataset.

In [13]:
```
# Convert 'PurchaseDate' to datetime and handle missing values
data['PurchaseDate'] = pd.to_datetime(data['PurchaseDate'], errors='coer

# Drop rows with missing PurchaseDate (if any)
data = data.dropna(subset=['PurchaseDate'])

# Calculate Recency
data['Recency'] = (datetime.now() - data['PurchaseDate']).dt.days


# Calculate Frequency
frequency_data = data.groupby('CustomerID')['OrderID'].count().reset_ind
frequency_data.rename(columns={'OrderID': 'Frequency'}, inplace=True)
data = data.merge(frequency_data, on='CustomerID', how='left')

# Calculate Monetary Value
monetary_data = data.groupby('CustomerID')['TransactionAmount'].sum().re
monetary_data.rename(columns={'TransactionAmount': 'MonetaryValue'}, inp
data = data.merge(monetary_data, on='CustomerID', how='left')
```

In [14]:
```
print(data.head())
```

```
   CustomerID PurchaseDate  TransactionAmount ProductInformation  Order
ID  \
0        8814   2023-04-11             943.31          Product C   8900
75
1        2188   2023-04-11             463.70          Product A   1768
19
2        4608   2023-04-11              80.28          Product A   3400
62
3        2559   2023-04-11             221.29          Product A   2391
45
4        9482   2023-04-11             739.56          Product A   1945
45

   Location  Recency  Frequency  MonetaryValue
0     Tokyo      386          1         943.31
1    London      386          1         463.70
2  New York      386          1          80.28
3    London      386          1         221.29
4     Paris      386          1         739.56
```

Calculating RFM Scores

- Define scoring criteria for each RFM value:
  - Recency scores: `[5, 4, 3, 2, 1]` (Higher score for lower recency, i.e., more recent purchases).
  - Frequency scores: `[1, 2, 3, 4, 5]` (Higher score for higher frequency, i.e., more frequent purchases).
  - Monetary scores: `[1, 2, 3, 4, 5]` (Higher score for higher monetary value, i.e., higher spending).

- Calculate RFM scores for each customer using `pd.cut` to create bins and assign corresponding scores:
  - Recency scores: `data['RecencyScore'] = pd.cut(data['Recency'], bins=5, labels=recency_scores)`.
  - Frequency scores: `data['FrequencyScore'] = pd.cut(data['Frequency'], bins=5, labels=frequency_scores)`.
  - Monetary scores: `data['MonetaryScore'] = pd.cut(data['MonetaryValue'], bins=5, labels=monetary_scores)`.

These steps show how to define scoring criteria and calculate RFM scores for each customer in the given dataset using Python

In [15]:
```
# Define scoring criteria for each RFM value
recency_scores = [5, 4, 3, 2, 1]  # Higher score for lower recency (more
frequency_scores = [1, 2, 3, 4, 5]  # Higher score for higher frequency
monetary_scores = [1, 2, 3, 4, 5]  # Higher score for higher monetary va

# Calculate RFM scores
data['RecencyScore'] = pd.cut(data['Recency'], bins=5, labels=recency_sc
data['FrequencyScore'] = pd.cut(data['Frequency'], bins=5, labels=freque
data['MonetaryScore'] = pd.cut(data['MonetaryValue'], bins=5, labels=mon
```

- Convert RFM scores to numeric type:
  - `data['RecencyScore'] = data['RecencyScore'].astype(int)`
  - `data['FrequencyScore'] = data['FrequencyScore'].astype(int)`
  - `data['MonetaryScore'] = data['MonetaryScore'].astype(int)`

These lines of code will convert the RFM scores in the 'RecencyScore', 'FrequencyScore', and 'MonetaryScore' columns to integers, allowing you to perform further numerical calculations and analysis with these scores.

In [16]:
```
# Convert RFM scores to numeric type
data['RecencyScore'] = data['RecencyScore'].astype(int)
data['FrequencyScore'] = data['FrequencyScore'].astype(int)
data['MonetaryScore'] = data['MonetaryScore'].astype(int)
```

In [17]:
```
print(data.head())
```

```
   CustomerID PurchaseDate  TransactionAmount ProductInformation  Order
ID  \
0        8814   2023-04-11             943.31          Product C   8900
75
1        2188   2023-04-11             463.70          Product A   1768
19
2        4608   2023-04-11              80.28          Product A   3400
62
3        2559   2023-04-11             221.29          Product A   2391
45
4        9482   2023-04-11             739.56          Product A   1945
45

   Location  Recency  Frequency  MonetaryValue  RecencyScore  Frequency
Score  \
0     Tokyo      386          1         943.31             1
1
1    London      386          1         463.70             1
1
2  New York      386          1          80.28             1
1
3    London      386          1         221.29             1
1
4     Paris      386          1         739.56             1
1

   MonetaryScore
0              2
1              1
2              1
3              1
4              2
```

RFM Value Segmentation

- Calculate RFM score by combining the individual scores:
  - `data['RFM_Score'] = data['RecencyScore'] + data['FrequencyScore'] + data['MonetaryScore']`
- Create RFM segments based on the RFM score using quantiles (q=3) to divide the data into three segments:
  - `segment_labels = ['Low-Value', 'Mid-Value', 'High-Value']`
  - `data['Value Segment'] = pd.qcut(data['RFM_Score'], q=3, labels=segment_labels)`

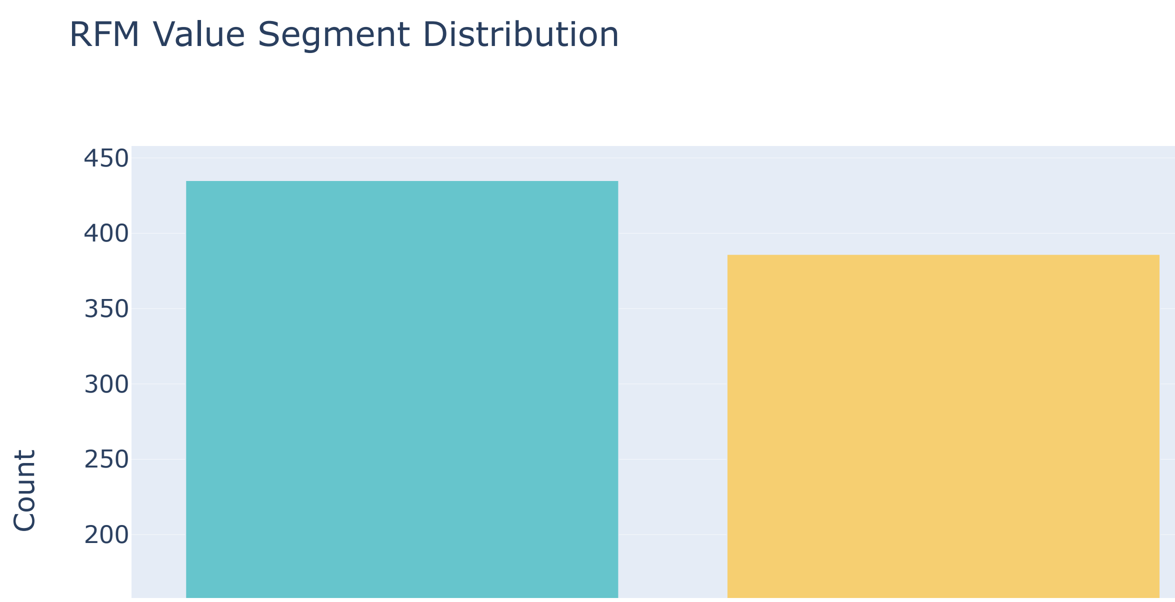
These lines of code will calculate the RFM score by adding the individual scores and then create RFM segments based on the score using quantiles, dividing the data into three segments labeled as 'Low-Value', 'Mid-Value', and 'High-Value'.

In [18]:
```
# Calculate RFM score by combining the individual scores
data['RFM_Score'] = data['RecencyScore'] + data['FrequencyScore'] + data

# Create RFM segments based on the RFM score
segment_labels = ['Low-Value', 'Mid-Value', 'High-Value']
data['Value Segment'] = pd.qcut(data['RFM_Score'], q=3, labels=segment_la
```

In [19]:
```
print(data.head())
```

```
   CustomerID PurchaseDate  TransactionAmount ProductInformation  Order
ID  \
0        8814   2023-04-11             943.31          Product C   8900
75
1        2188   2023-04-11             463.70          Product A   1768
19
2        4608   2023-04-11              80.28          Product A   3400
62
3        2559   2023-04-11             221.29          Product A   2391
45
4        9482   2023-04-11             739.56          Product A   1945
45

   Location  Recency  Frequency  MonetaryValue  RecencyScore  Frequency
Score  \
0     Tokyo      340          1         943.31             1
1
1    London      340          1         463.70             1
1
2  New York      340          1          80.28             1
1
3    London      340          1         221.29             1
1
4     Paris      340          1         739.56             1
1

   MonetaryScore  RFM_Score Value Segment
0              2          4     Low-Value
1              1          3     Low-Value
2              1          3     Low-Value
3              1          3     Low-Value
4              2          4     Low-Value
```

In [19]:
```
# RFM Segment Distribution
segment_counts = data['Value Segment'].value_counts(dropna=False).reset_
segment_counts.columns = ['Value Segment', 'Count']

pastel_colors = px.colors.qualitative.Pastel

# Create the bar chart
fig_segment_dist = px.bar(segment_counts, x='Value Segment', y='Count',
                          color='Value Segment', color_discrete_sequence
                          title='RFM Value Segment Distribution')

# Update the layout
fig_segment_dist.update_layout(xaxis_title='RFM Value Segment',
                               yaxis_title='Count',
                               showlegend=False)

# Show the figure
fig_segment_dist.show()
```



RFM Customer Segments

- Create a new column for RFM Customer Segments:

    - `data['RFM Customer Segments'] = ''`
- Assign RFM segments based on the RFM score using `loc`:
    - For 'Champions' (RFM Score >= 9): `data.loc[data['RFM_Score'] >= 9, 'RFM Customer Segments'] = 'Champions'`
    - For 'Potential Loyalists' (6 <= RFM Score < 9): `data.loc[(data['RFM_Score'] >= 6) & (data['RFM_Score'] < 9), 'RFM Customer Segments'] = 'Potential Loyalists'`
    - For 'At Risk Customers' (5 <= RFM Score < 6): `data.loc[(data['RFM_Score'] >= 5) & (data['RFM_Score'] < 6), 'RFM Customer Segments'] = 'At Risk Customers'`
    - For "Can't Lose" (4 <= RFM Score < 5): `data.loc[(data['RFM_Score'] >= 4) & (data['RFM_Score'] < 5), 'RFM Customer Segments'] = "Can't Lose"`
    - For "Lost" (3 <= RFM Score < 4): `data.loc[(data['RFM_Score'] >= 3) & (data['RFM_Score'] < 4), 'RFM Customer Segments'] = "Lost"`
- Print the updated data with RFM segments:
    - `print(data[['CustomerID', 'RFM Customer Segments']])`

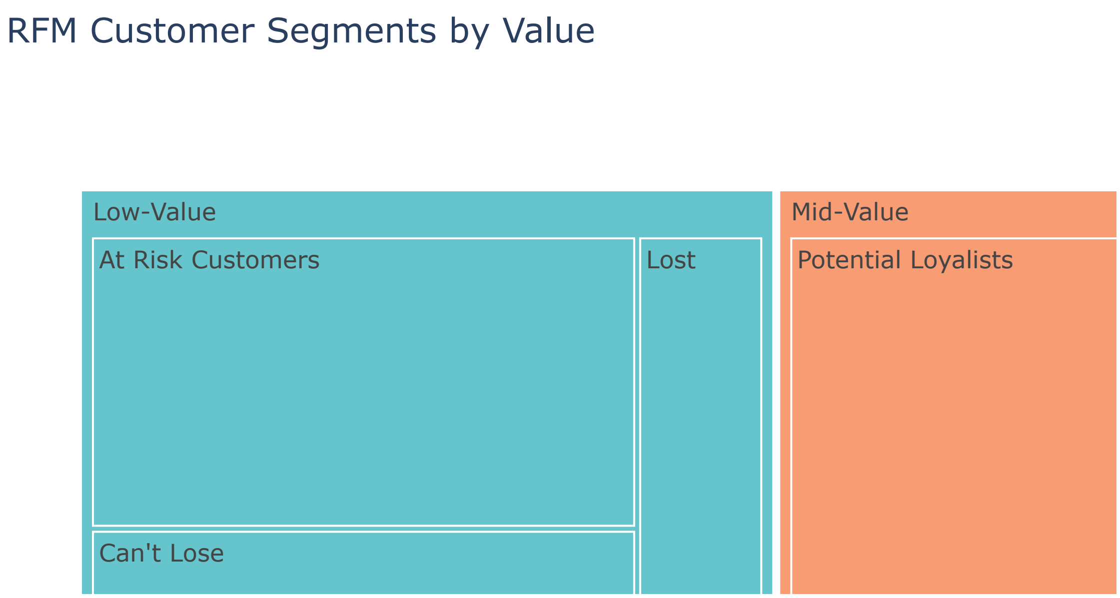
This code will create a new column 'RFM Customer Segments' and assign the appropriate segment label based on the RFM score for each customer in the dataset. The printed output will display the CustomerID along with their corresponding RFM Customer Segment.

In [20]:
```
# Create a new column for RFM Customer Segments
data['RFM Customer Segments'] = ''

# Assign RFM segments based on the RFM score
data.loc[data['RFM_Score'] >= 9, 'RFM Customer Segments'] = 'Champions'
data.loc[(data['RFM_Score'] >= 6) & (data['RFM_Score'] < 9), 'RFM Custom
data.loc[(data['RFM_Score'] >= 5) & (data['RFM_Score'] < 6), 'RFM Custom
data.loc[(data['RFM_Score'] >= 4) & (data['RFM_Score'] < 5), 'RFM Custom
data.loc[(data['RFM_Score'] >= 3) & (data['RFM_Score'] < 4), 'RFM Custom

# Print the updated data with RFM segments
print(data[['CustomerID', 'RFM Customer Segments']])
```

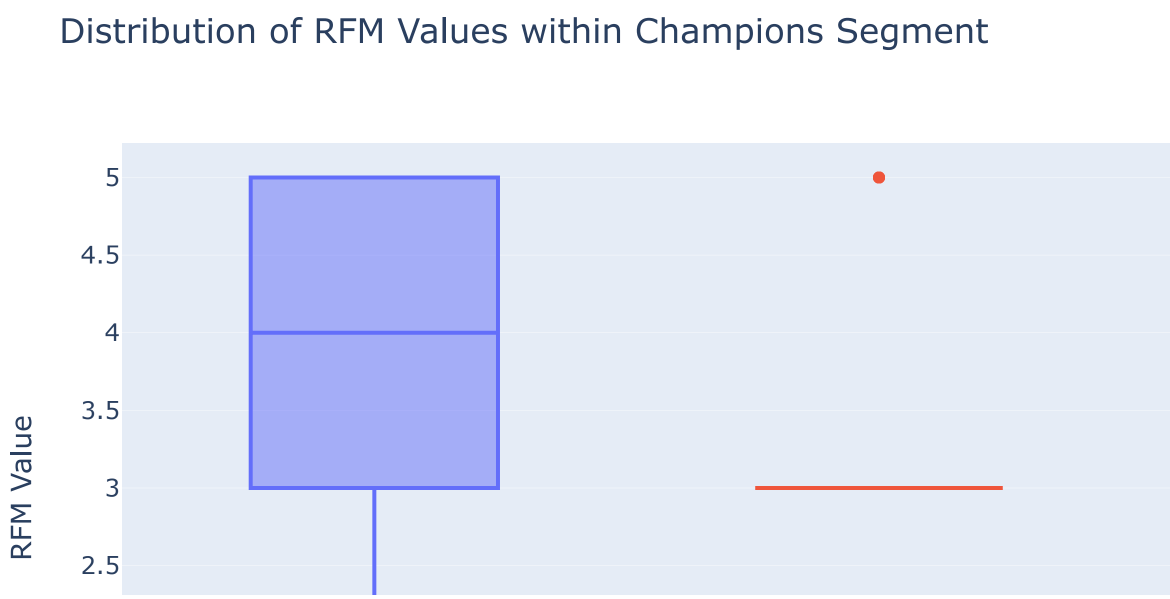
```
     CustomerID RFM Customer Segments
0          8814            Can't Lose
1          2188                  Lost
2          4608                  Lost
3          2559                  Lost
4          9482            Can't Lose
..          ...                   ...
995        2970   Potential Loyalists
996        6669   Potential Loyalists
997        8836   Potential Loyalists
998        1440   Potential Loyalists
999        4759   Potential Loyalists

[1000 rows x 2 columns]
```

RFM Analysis

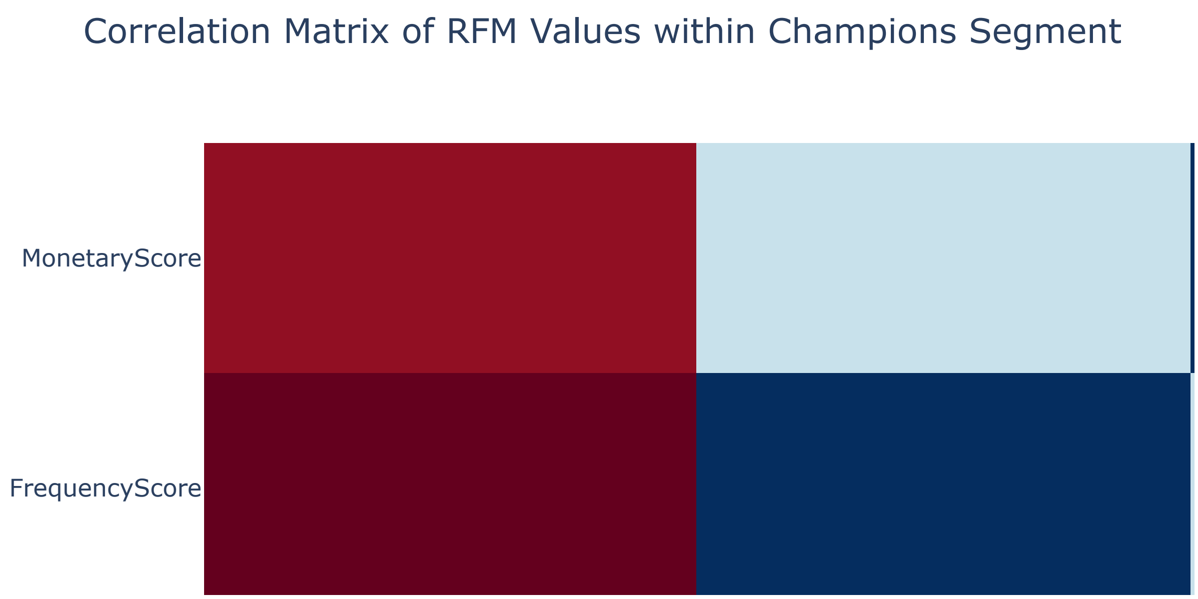
In [21]:
```
segment_product_counts = data.groupby(['Value Segment', 'RFM Customer Seg

segment_product_counts = segment_product_counts.sort_values('Count', asc

fig_treemap_segment_product = px.treemap(segment_product_counts,
                                          path=['Value Segment', 'RFM Cus
                                          values='Count',
                                          color='Value Segment', color_di
                                          title='RFM Customer Segments by
fig_treemap_segment_product.show()
```



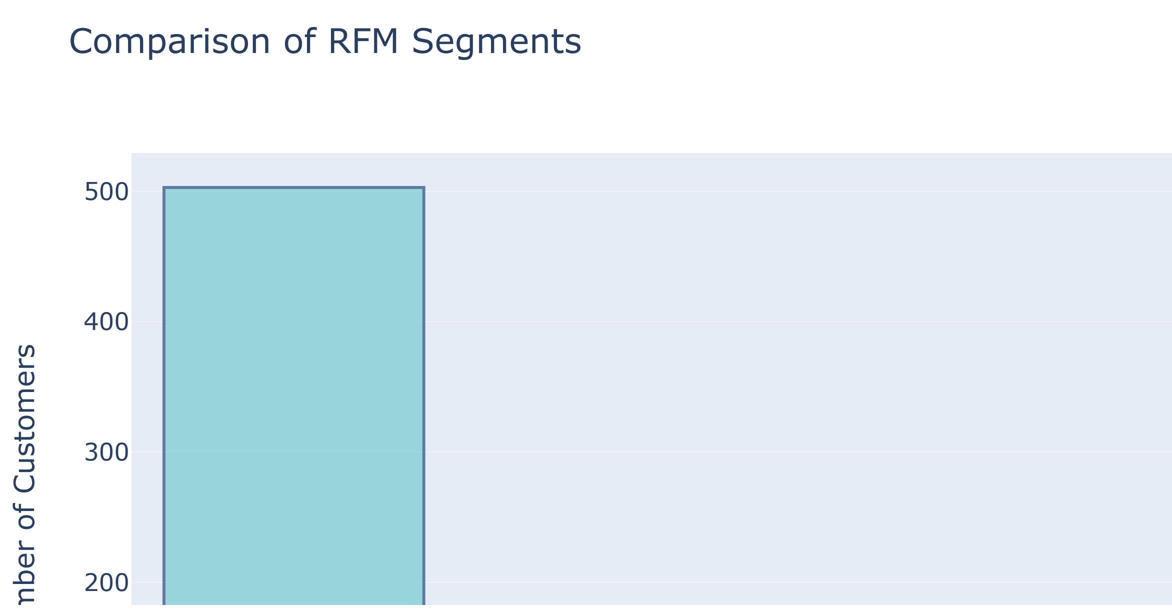
In [22]:
```
# Filter the data to include only the customers in the Champions segment
champions_segment = data[data['RFM Customer Segments'] == 'Champions']

fig = go.Figure()
fig.add_trace(go.Box(y=champions_segment['RecencyScore'], name='Recency'
fig.add_trace(go.Box(y=champions_segment['FrequencyScore'], name='Frequen
fig.add_trace(go.Box(y=champions_segment['MonetaryScore'], name='Monetary

fig.update_layout(title='Distribution of RFM Values within Champions Segm
                  yaxis_title='RFM Value',
                  showlegend=True)

fig.show()
```



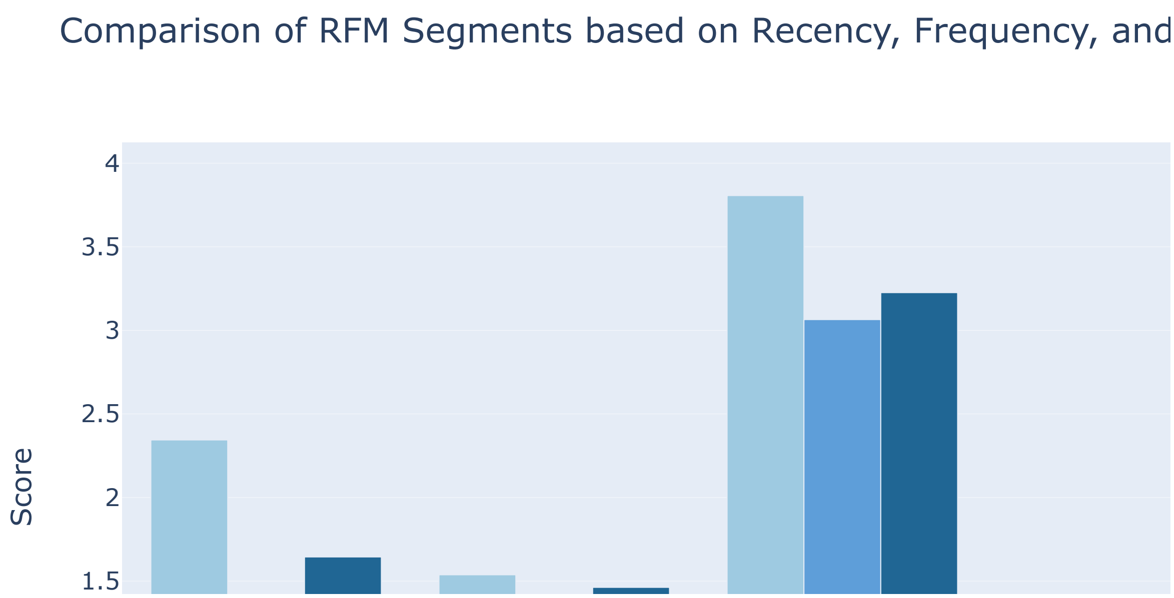
In [24]:
```
correlation_matrix = champions_segment[['RecencyScore', 'FrequencyScore'

# Visualize the correlation matrix using a heatmap
fig_heatmap = go.Figure(data=go.Heatmap(
                   z=correlation_matrix.values,
                   x=correlation_matrix.columns,
                   y=correlation_matrix.columns,
                   colorscale='RdBu',
                   colorbar=dict(title='Correlation')))

fig_heatmap.update_layout(title='Correlation Matrix of RFM Values within

fig_heatmap.show()
```



In [25]:
```
import plotly.colors

pastel_colors = plotly.colors.qualitative.Pastel

segment_counts = data['RFM Customer Segments'].value_counts()

# Create a bar chart to compare segment counts
fig = go.Figure(data=[go.Bar(x=segment_counts.index, y=segment_counts.va
                            marker=dict(color=pastel_colors))])

# Set the color of the Champions segment as a different color
champions_color = 'rgb(158, 202, 225)'
fig.update_traces(marker_color=[champions_color if segment == 'Champions
                                for i, segment in enumerate(segment_coun
                  marker_line_color='rgb(8, 48, 107)',
                  marker_line_width=1.5, opacity=0.6)

# Update the layout
fig.update_layout(title='Comparison of RFM Segments',
                  xaxis_title='RFM Segments',
                  yaxis_title='Number of Customers',
                  showlegend=False)

fig.show()
```



In [26]:
```
# Calculate the average Recency, Frequency, and Monetary scores for each
segment_scores = data.groupby('RFM Customer Segments')[['RecencyScore',

# Create a grouped bar chart to compare segment scores
fig = go.Figure()

# Add bars for Recency score
fig.add_trace(go.Bar(
    x=segment_scores['RFM Customer Segments'],
    y=segment_scores['RecencyScore'],
    name='Recency Score',
    marker_color='rgb(158,202,225)'
))

# Add bars for Frequency score
fig.add_trace(go.Bar(
    x=segment_scores['RFM Customer Segments'],
    y=segment_scores['FrequencyScore'],
    name='Frequency Score',
    marker_color='rgb(94,158,217)'
))

# Add bars for Monetary score
fig.add_trace(go.Bar(
    x=segment_scores['RFM Customer Segments'],
    y=segment_scores['MonetaryScore'],
    name='Monetary Score',
    marker_color='rgb(32,102,148)'
))

# Update the layout
fig.update_layout(
    title='Comparison of RFM Segments based on Recency, Frequency, and Mo
    xaxis_title='RFM Segments',
    yaxis_title='Score',
    barmode='group',
    showlegend=True
)

fig.show()
```



Summary

RFM Analysis is a powerful method for understanding and segmenting customers based on their buying behavior. The acronym RFM stands for recency, frequency, and monetary value, which are crucial indicators of customer engagement, loyalty, and value to a business. By leveraging Python, we can perform RFM Analysis and gain valuable insights into customer segments.

END