In [1]:
```
# Importing necessary Libraries

import pandas as pd
from datetime import datetime
import plotly.express as px
import plotly.io as pio
import plotly.colors
import plotly.graph_objects as go
pio.templates.default = "simple_white"
```

In [2]:
```
# Read CSV input file

df= pd.read_csv(".//Dataset//rfm_data.csv")

# Convert Datetime

df["PurchaseDate"] = pd.to_datetime(df["PurchaseDate"],format = "%Y-%m-%d")
```

In [3]:
```
# Rearranging columns

# shift column 'OrderID' to first position
Order_column = df.pop('OrderID')
Transaction_column = df.pop("TransactionAmount")

df.insert(1, 'OrderID', Order_column)
df.insert(5, "TransactionAmount",Transaction_column)
```

In [4]:
```
df
```

Out[4]:

	CustomerID	OrderID	PurchaseDate	ProductInformation	Location	TransactionAmount
0	8814	890075	2023-04-11	Product C	Tokyo	943.31
1	2188	176819	2023-04-11	Product A	London	463.70
2	4608	340062	2023-04-11	Product A	New York	80.28
3	2559	239145	2023-04-11	Product A	London	221.29
4	9482	194545	2023-04-11	Product A	Paris	739.56
...	...	...	...	...	...	...
995	2970	275284	2023-06-10	Product B	London	759.62
996	6669	987025	2023-06-10	Product C	New York	941.50
997	8836	512842	2023-06-10	Product C	London	545.36
998	1440	559753	2023-06-10	Product B	Paris	729.94
999	4759	467544	2023-06-10	Product D	New York	804.28

1000 rows × 6 columns

For Recency calculation , the purchase date is substracted from the current date and determined the number of days using the datetime.now().date() function. This process yields the number of days elapsed since the customer's most recent purchase, serving as their recency value.

In [5]:
```
# Calculate Recency

df['Recency'] = (datetime.now().date() - df['PurchaseDate'].dt.date).dt.days
```

Following that, the frequency for each customer is calculated. By grouping the data according to 'CustomerID' and counted the distinct 'OrderID' values, gives the quantity of purchases made by each customer. This procedure yields the frequency value, denoting the overall number of purchases completed by individual customers.

In [6]:
```
# Calculate Frequency

frequency_data = df.groupby('CustomerID')['OrderID'].count().reset_index()
frequency_data.rename(columns={'OrderID': 'Frequency'}, inplace=True)
df= df.merge(frequency_data, on='CustomerID', how='left')
```

Ultimately, the monetary value for each customer is calculated. By grouping the data based on 'CustomerID' and aggregating the 'TransactionAmount' values, the cumulative amount spent by each customer i determined. This yields the monetary value, indicating the overall financial contribution of each customer.

In [7]:
```
# Calculate Monetary Value

monetory_data= df.groupby("CustomerID")["TransactionAmount"].sum().reset_index()
monetory_data.rename(columns={'TransactionAmount':'MonetoryValue'}, inplace=True)
df = df.merge(monetory_data, on= "CustomerID", how = "left" )
```

Through these computations, we have acquired the essential RFM metrics (recency, frequency, monetary value) for every customer. These metrics serve as pivotal indicators for comprehending customer behavior and facilitating segmentation within RFM analysis.

In [8]:
```
df
```

Out[8]:

	CustomerID	OrderID	PurchaseDate	ProductInformation	Location	TransactionAmount	Recency	Frequency	MonetoryValue
0	8814	890075	2023-04-11	Product C	Tokyo	943.31	342	1	943.31
1	2188	176819	2023-04-11	Product A	London	463.70	342	1	463.70
2	4608	340062	2023-04-11	Product A	New York	80.28	342	1	80.28
3	2559	239145	2023-04-11	Product A	London	221.29	342	1	221.29
4	9482	194545	2023-04-11	Product A	Paris	739.56	342	1	739.56
...	...	...	...	...	...	...	...	...	...
995	2970	275284	2023-06-10	Product B	London	759.62	282	1	759.62
996	6669	987025	2023-06-10	Product C	New York	941.50	282	1	941.50
997	8836	512842	2023-06-10	Product C	London	545.36	282	1	545.36
998	1440	559753	2023-06-10	Product B	Paris	729.94	282	1	729.94
999	4759	467544	2023-06-10	Product D	New York	804.28	282	1	804.28

1000 rows × 9 columns

In [9]:
```
# Defining score for each RFM value
recency_scores = [5, 4, 3, 2, 1]  # Higher score for lower recency (more recent)
frequency_scores = [1, 2, 3, 4, 5]  # Higher score for higher frequency
monetary_scores = [1, 2, 3, 4, 5]  # Higher score for higher monetary value

# Calculate RFM scores
df['RecencyScore'] = pd.cut(df['Recency'], bins=5, labels=recency_scores)
df['FrequencyScore'] = pd.cut(df['Frequency'], bins=5, labels=frequency_scores)
df['MonetaryScore'] = pd.cut(df['MonetoryValue'], bins=5, labels=monetary_scores)
```

In [10]:
```
# Convert RFM scores to numeric type
df['RecencyScore'] = df['RecencyScore'].astype(int)
df['FrequencyScore'] = df['FrequencyScore'].astype(int)
df['MonetaryScore'] = df['MonetaryScore'].astype(int)
```

In [11]:
```
# Calculate RFM score by combining the individual scores
df['RFM_Score'] = df['RecencyScore'] + df['FrequencyScore'] + df['MonetaryScore']
```

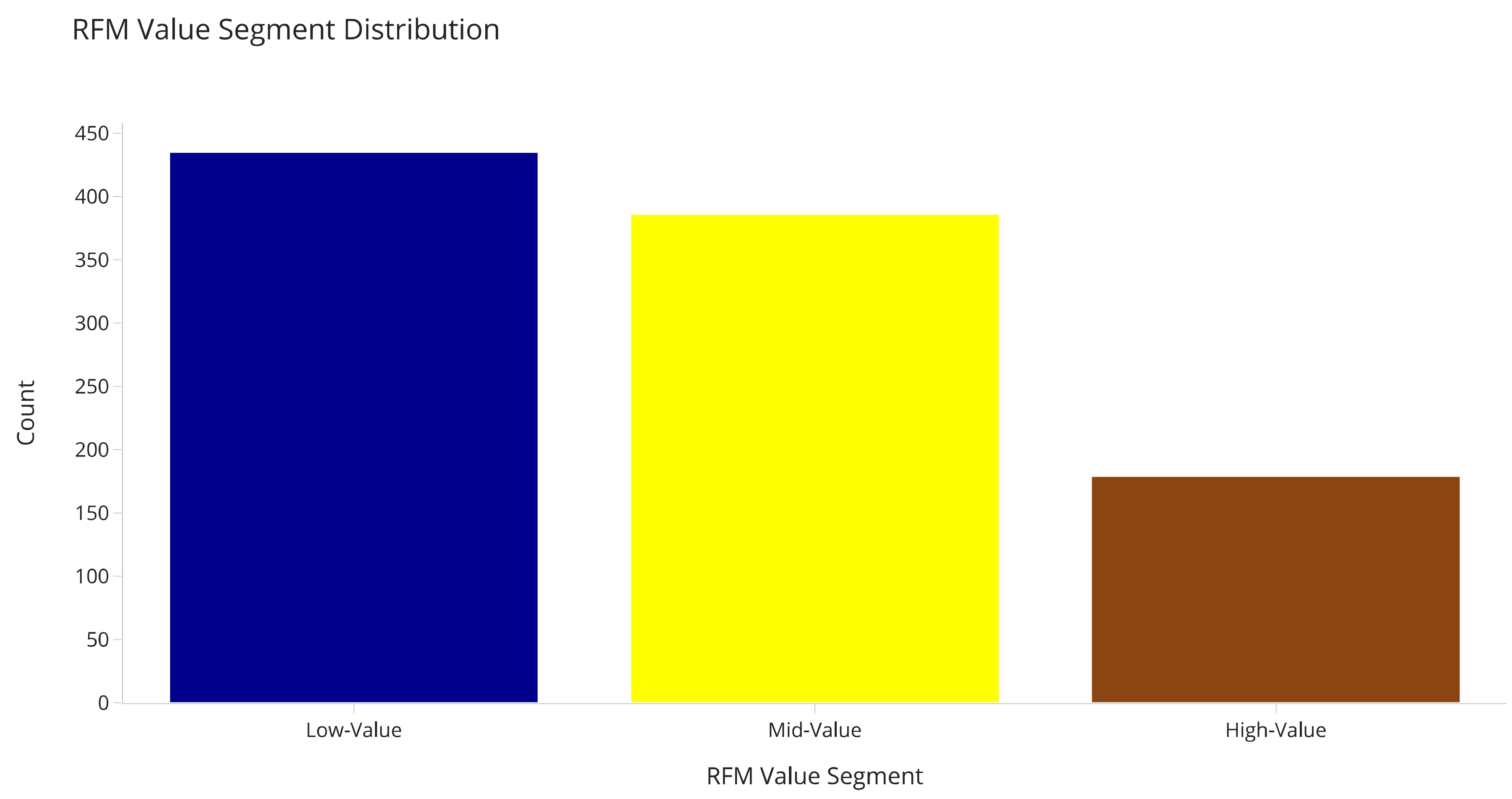
In [12]:
```
# Create RFM segments based on the RFM score
segment_labels = ['Low-Value', 'Mid-Value', 'High-Value']
df['Value Segment'] = pd.qcut(df['RFM_Score'], q=3, labels=segment_labels)
```

In [13]:
```
df.head()
```

Out[13]:

	CustomerID	OrderID	PurchaseDate	ProductInformation	Location	TransactionAmount	Recency	Frequency	MonetoryValue	RecencyScore	FrequencyScore	MonetaryScore	RFM_Score	Value Segment
0	8814	890075	2023-04-11	Product C	Tokyo	943.31	342	1	943.31	1	1	2	4	Low-Value
1	2188	176819	2023-04-11	Product A	London	463.70	342	1	463.70	1	1	1	3	Low-Value
2	4608	340062	2023-04-11	Product A	New York	80.28	342	1	80.28	1	1	1	3	Low-Value
3	2559	239145	2023-04-11	Product A	London	221.29	342	1	221.29	1	1	1	3	Low-Value
4	9482	194545	2023-04-11	Product A	Paris	739.56	342	1	739.56	1	1	2	4	Low-Value

In [14]:
```
# RFM Segment Distribution
segment_counts = df['Value Segment'].value_counts().reset_index()
segment_counts.columns = ['Value Segment', 'Count']

# Create the bar chart
custom_color ={
    'Low-Value': 'darkblue',
    'Mid-Value': 'yellow',
    'High-Value': 'saddlebrown'
}

fig_segment_dist = px.bar(segment_counts, x='Value Segment', y='Count',
                          color='Value Segment', color_discrete_map=custom_color,
                          title='RFM Value Segment Distribution')
fig_segment_dist.update_layout(xaxis_title='RFM Value Segment',
                              yaxis_title='Count',
                              showlegend=False)
fig_segment_dist.show()
```

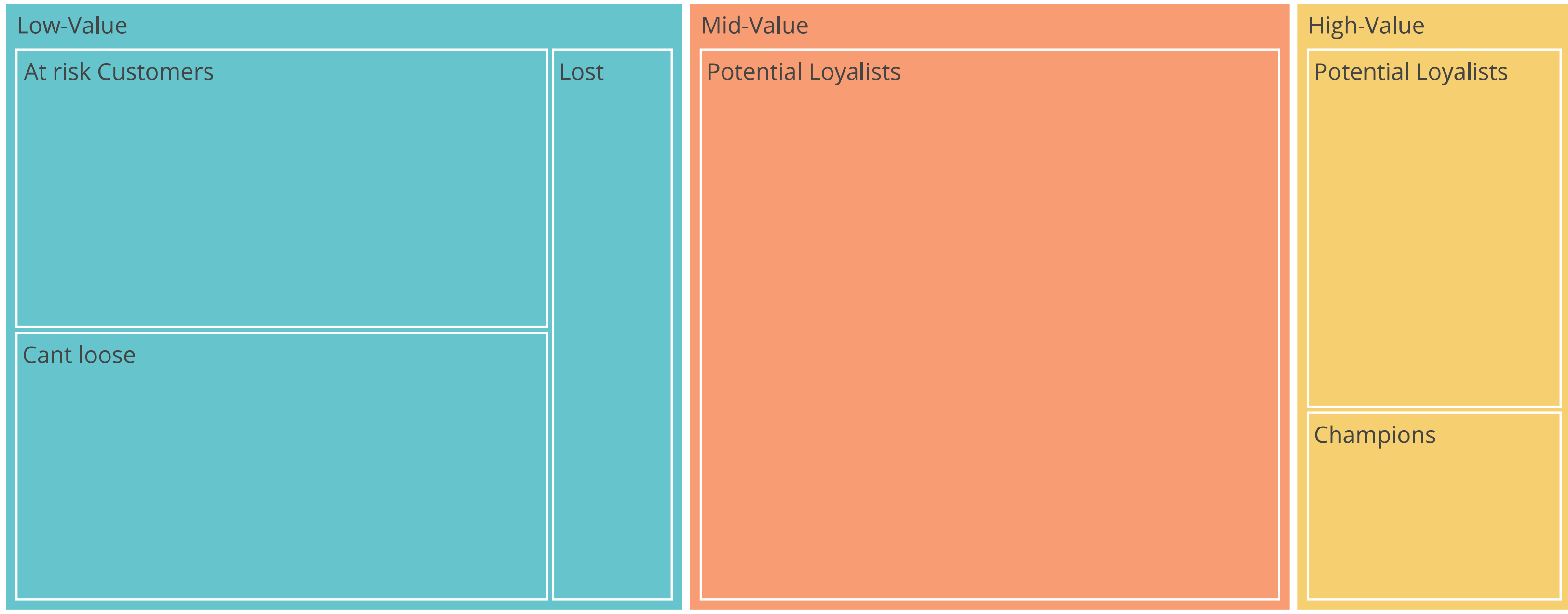


In [15]:
```
# Create a new column for RFM Customer Segments
df['RFM Customer Segments'] = ''

# Assign RFM segments based on the RFM score
df.loc[df['RFM_Score'] >= 9, 'RFM Customer Segments'] = 'Champions'
df.loc[(df['RFM_Score'] >= 6) & (df['RFM_Score'] < 9), 'RFM Customer Segments'] = 'Potential Loyalists'
df.loc[(df['RFM_Score'] >= 5) & (df['RFM_Score'] < 6), 'RFM Customer Segments'] = 'At risk Customers'
df.loc[(df['RFM_Score'] >= 4) & (df['RFM_Score'] < 5), 'RFM Customer Segments'] = "Cant loose"
df.loc[(df['RFM_Score'] >= 3) & (df['RFM_Score'] < 4), 'RFM Customer Segments'] = "Lost"

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

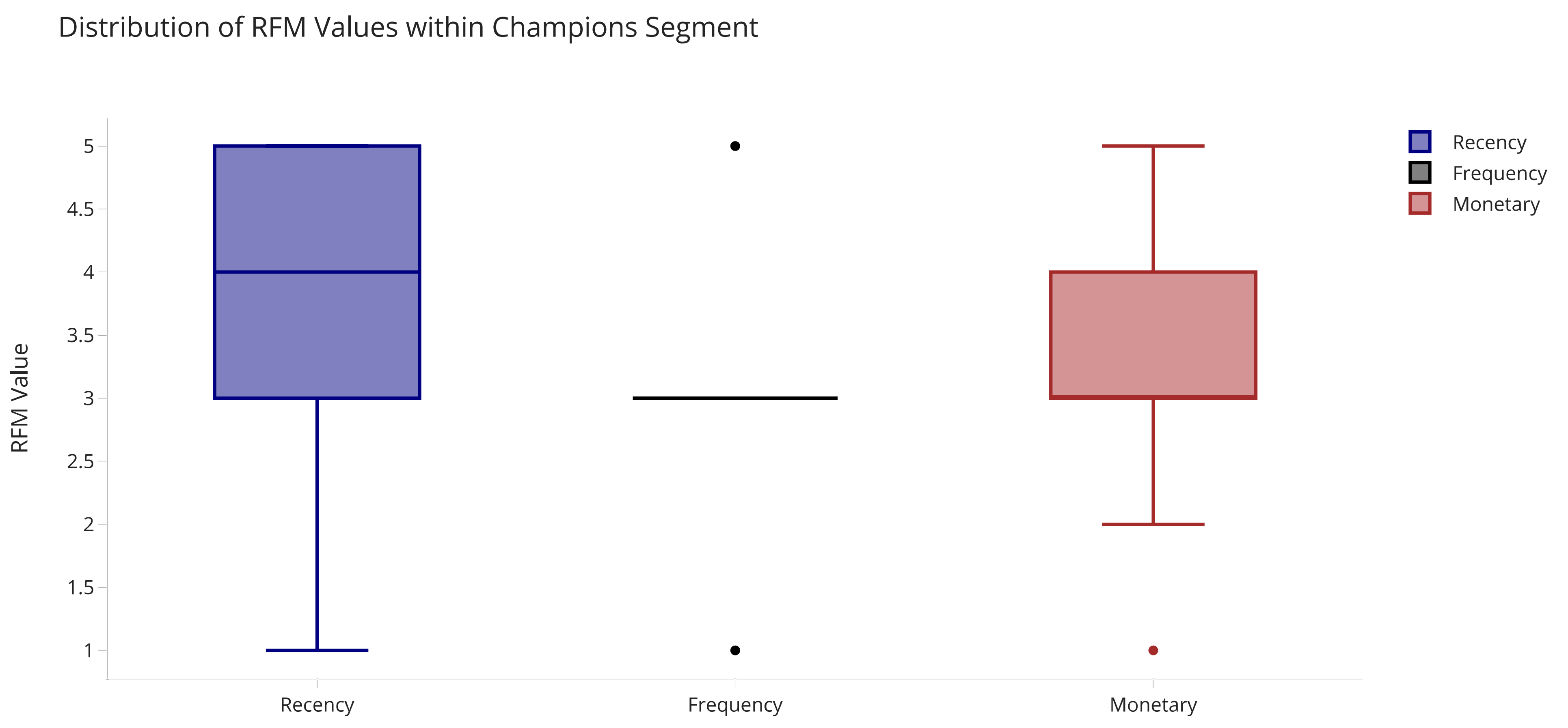
```
     CustomerID RFM Customer Segments
0          8814            Cant loose
1          2188                  Lost
2          4608                  Lost
3          2559                  Lost
4          9482            Cant loose
..          ...                   ...
995        2970   Potential Loyalists
996        6669   Potential Loyalists
997        8836   Potential Loyalists
998        1440   Potential Loyalists
999        4759   Potential Loyalists

[1000 rows x 2 columns]
```

In [16]:
```
segment_product_counts = df.groupby(['Value Segment', 'RFM Customer Segments']).size().reset_index(name='Count')

segment_product_counts = segment_product_counts.sort_values('Count', ascending=False)

fig_treemap_segment_product = px.treemap(segment_product_counts,
                                         path=['Value Segment', 'RFM Customer Segments'],
                                         values='Count',
                                         color='Value Segment',  color_discrete_sequence=px.colors.qualitative.Pastel,
                                         title='RFM Customer Segments by Value')
fig_treemap_segment_product.show()
```

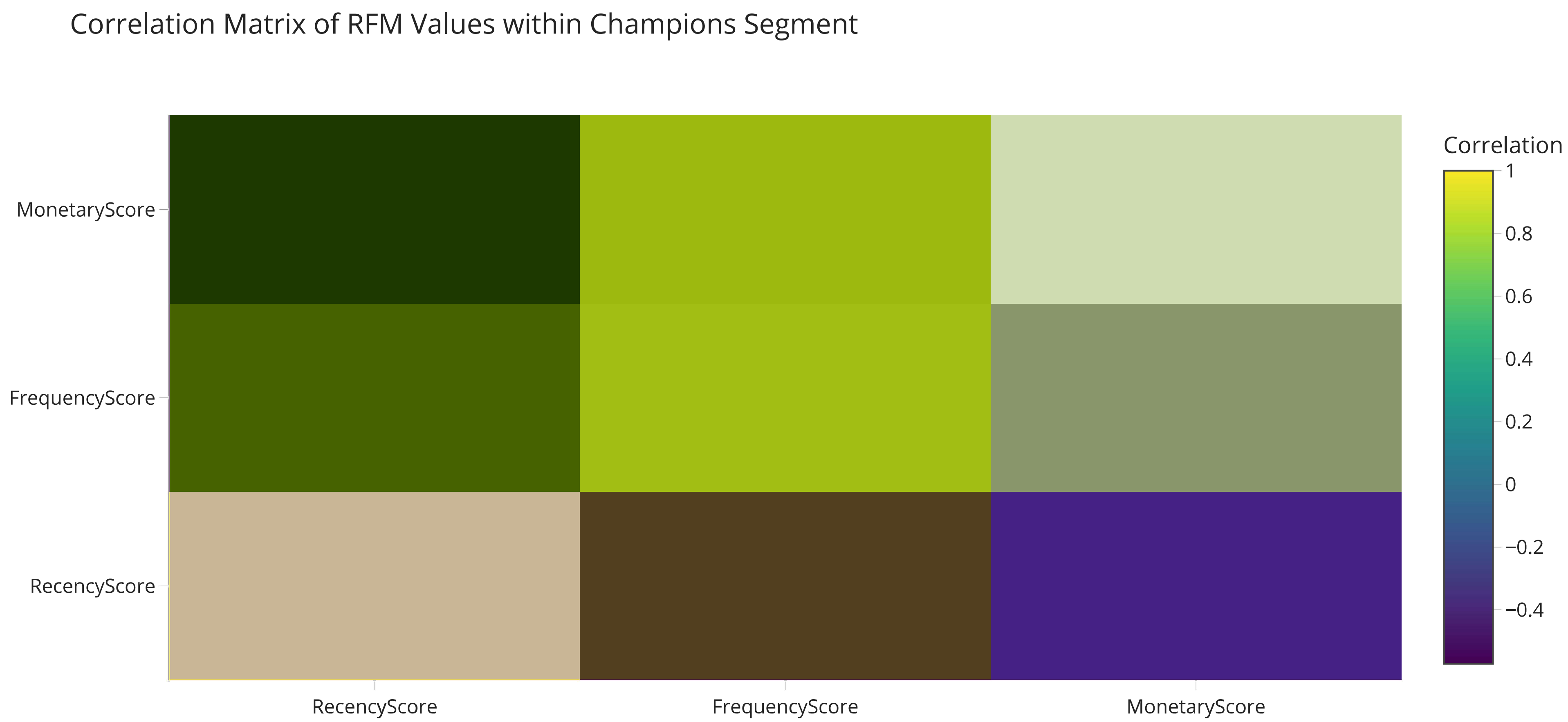
RFM Customer Segments by Value



In [17]:
```
# Filter the data to include only the customers in the Champions segment
champions_segment = df[df['RFM Customer Segments'] == 'Champions']

custom_color = {
    'Recency': 'navy',
    'Frequency': 'black',
    'Monetary': 'brown'
}

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

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

fig.show()
```

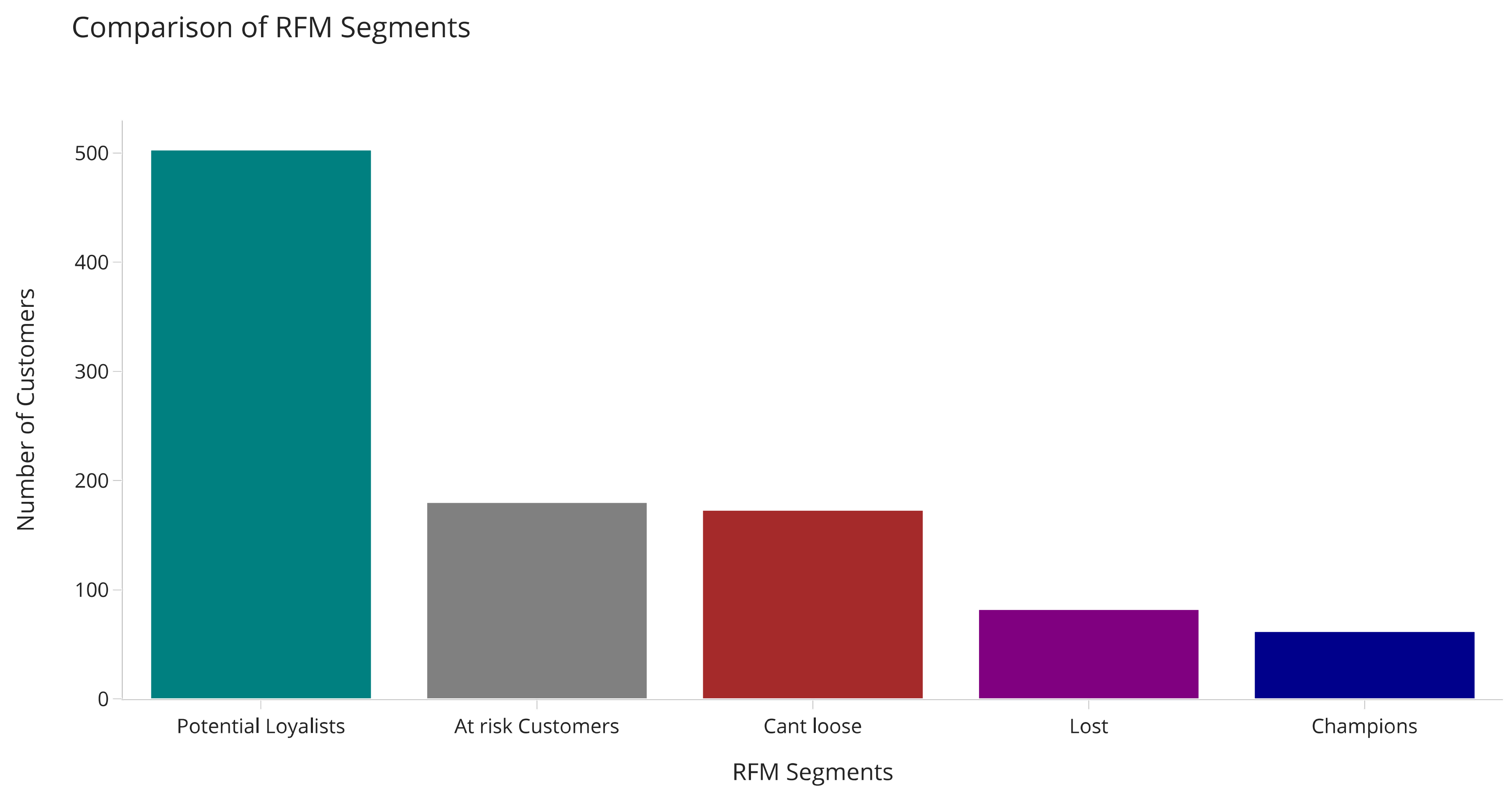


In [18]:
```
correlation_matrix = champions_segment[['RecencyScore', 'FrequencyScore', 'MonetaryScore']].corr()

# 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='Viridis',  # Change colorscale to Viridis
                   colorbar=dict(title='Correlation')))

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

fig_heatmap.show()
```



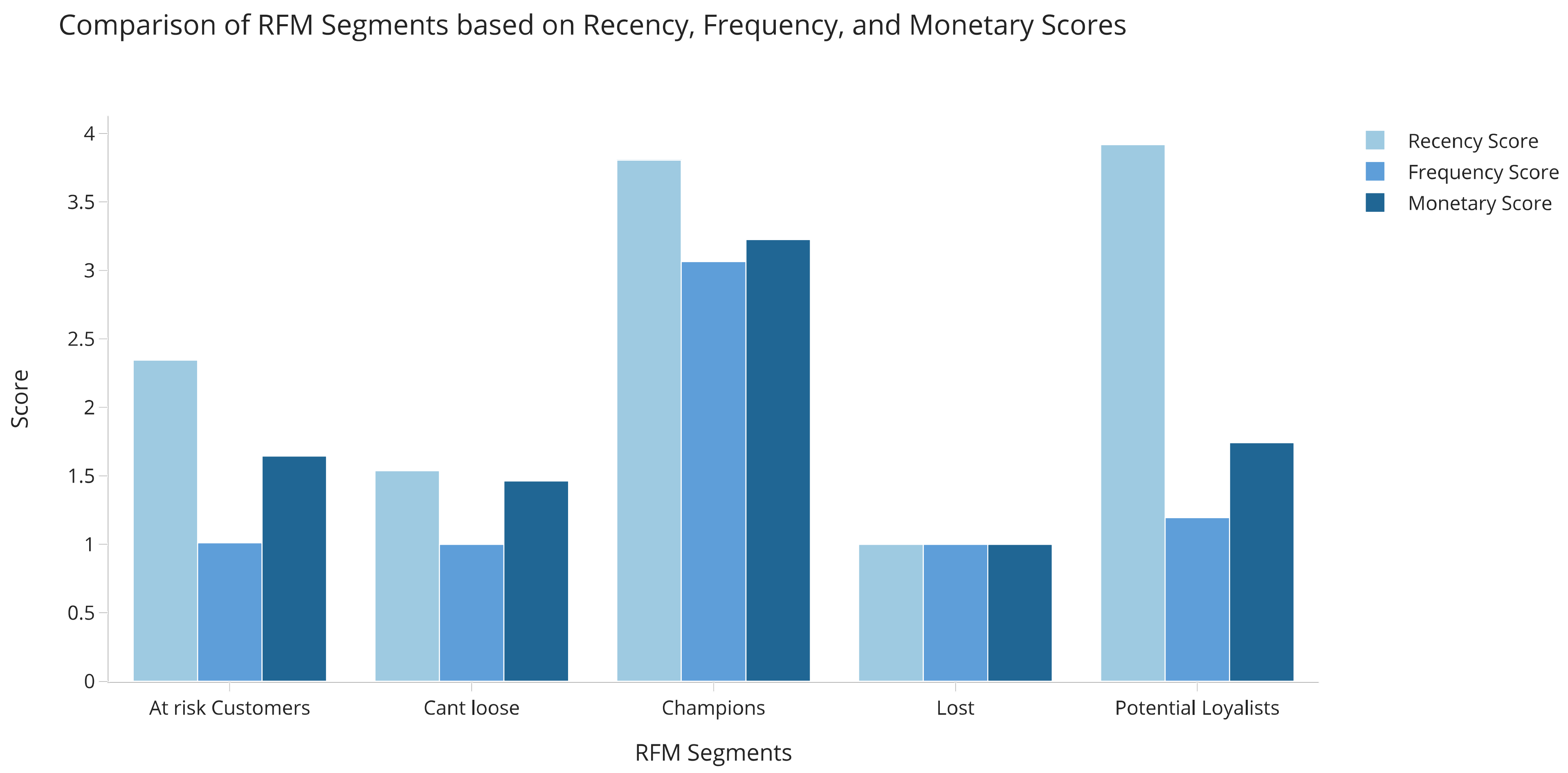
In [19]:

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

# Define custom colors for each segment
custom_color = {
    'Potential Loyalists': 'teal',
    'At risk': 'orange',
    'Champions': 'darkblue',
    'Cant loose': 'brown',
    'Lost': 'purple'
}

fig = go.Figure(data=[go.Bar(x=segment_counts.index, y=segment_counts.values,
                             marker=dict(color=[custom_color.get(segment, 'gray') for segment in segment_counts.index]))])

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

fig.show()
```



In [20]:
```
# Calculate the average Recency, Frequency, and Monetary scores for each segment
segment_scores = df.groupby('RFM Customer Segments')['RecencyScore', 'FrequencyScore', 'MonetaryScore'].mean().reset_index()

# 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 Monetary Scores',
    xaxis_title='RFM Segments',
    yaxis_title='Score',
    barmode='group',
    showlegend=True
)

fig.show()
```

```
<ipython-input-20-6cd24555ec08>:2: FutureWarning:

Indexing with multiple keys (implicitly converted to a tuple of keys) will be deprecated, use a list instead.
```



RFM Analysis is a methodology employed to dissect and categorize customers according to their purchasing patterns. RFM denotes recency, frequency, and monetary value, three pivotal metrics that furnish insights into customer engagement, loyalty, and significance to a business.

In [ ]:

```

```