RFM Analysis

RFM Analysis is a **strategic tool** utilized by **Data Science professionals**, particularly in the realm of **marketing**, to gain insights into **customer behavior** and effectively **segment them**.

This analytical approach allows businesses to evaluate customers based on three fundamental metrics:

1. **Recency**: This metric indicates how recently a customer made a purchase. It provides insight into the **freshness of customer engagement**.
2. **Frequency**: Frequency denotes how often a customer makes purchases within a given timeframe. It highlights the **consistency and regularity** of customer transactions.
3. **Monetary value**: This metric reflects the **monetary worth** of a customer to the business, typically measured by the **total amount spent** on purchases. It indicates the **level of financial contribution** made by each customer.

By assessing these **RFM values**, businesses can gain a comprehensive understanding of **customer engagement**, **loyalty**, and overall **value**. This insight enables **targeted marketing strategies** and **tailored customer experiences**, ultimately driving **profitability** and **growth**.

To conduct RFM analysis using **Python**, a dataset containing **customer IDs**, **purchase dates**, and **transaction amounts** is required. By leveraging this dataset, RFM values can be computed for each customer, facilitating **detailed analysis** of their **purchasing patterns** and **behaviors**.

```
import pandas as pd
import plotly.express as px
import plotly.io as pio
import plotly.graph_objects as go
pio.templates.default = "plotly_white"
```

```
# Mounting to you own Google Colab drive
from google.colab import drive
try:
  drive.mount('/gdrive')
except:
  drive.mount('/content/gdrive', force_remount=True)
```

```
Drive already mounted at /gdrive; to attempt to forcibly remount, call drive.mount("/gdrive", force_remount=Tru
```

```
%cd '/gdrive/MyDrive/projects'
```

```
/gdrive/MyDrive/projects
```

```
import pandas as pd
data = pd.read_csv('/gdrive/My Drive/projects/rfm_data.csv')
print(data.head())
```

```
   CustomerID PurchaseDate  TransactionAmount ProductInformation  OrderID  \
0        8814   2023-04-11             943.31          Product C   890075
1        2188   2023-04-11             463.70          Product A   176819
2        4608   2023-04-11              80.28          Product A   340062
3        2559   2023-04-11             221.29          Product A   239145
4        9482   2023-04-11             739.56          Product A   194545

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

I will now determine the Recency, Frequency, and Monetary values of the customers to proceed with the analysis.

```
from datetime import datetime

# Convert 'PurchaseDate' to datetime
data['PurchaseDate'] = pd.to_datetime(data['PurchaseDate'])

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

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

To calculate **recency**, we determined the number of days since the customer's last purchase by subtracting the purchase date from the current date using the `datetime.now().date()` function. This provides us with the recency value, reflecting how recently the customer made a purchase.

Next, for **frequency**, we grouped the data by 'CustomerID' and counted the number of unique 'OrderID' values to ascertain the total number of purchases made by each customer. This yields the frequency value, representing the overall purchase frequency of each customer.

Lastly, to compute **monetary value**, we grouped the data by 'CustomerID' and summed the 'TransactionAmount' values to calculate the total amount spent by each customer. This results in the monetary value, indicating the total monetary contribution made by each customer.

These calculations yield the essential **RFM values**—recency, frequency, and monetary value—for each customer. These values serve as crucial indicators for understanding **customer behavior** and facilitating segmentation in RFM analysis.

```
print(data.head())
```

```
   CustomerID PurchaseDate  TransactionAmount ProductInformation  OrderID  \
0        8814   2023-04-11             943.31          Product C   890075
1        2188   2023-04-11             463.70          Product A   176819
2        4608   2023-04-11              80.28          Product A   340062
3        2559   2023-04-11             221.29          Product A   239145
4        9482   2023-04-11             739.56          Product A   194545

   Location  Recency  Frequency  MonetaryValue
0     Tokyo      344          1         943.31
1    London      344          1         463.70
2  New York      344          1          80.28
3    London      344          1         221.29
4     Paris      344          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 (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
data['RecencyScore'] = pd.cut(data['Recency'], bins=5, labels=recency_scores)
data['FrequencyScore'] = pd.cut(data['Frequency'], bins=5, labels=frequency_scores)
data['MonetaryScore'] = pd.cut(data['MonetaryValue'], bins=5, labels=monetary_scores)
```

We assigned scores ranging from 5 to 1 to calculate the **recency score**, where a higher score signifies a more recent purchase. This implies that customers who have made purchases more recently will receive higher recency scores.

Similarly, for the **frequency score**, we assigned scores from 1 to 5, with a higher score indicating a higher purchase frequency. Customers who made more frequent purchases will consequently receive higher frequency scores.

To determine the **monetary score**, we allocated scores from 1 to 5, where a higher score signifies a higher amount spent by the customer.

To calculate the **RFM scores**, we utilized the `pd.cut()` function to segment the recency, frequency, and monetary values into bins. We divided each value into 5 bins and assigned corresponding scores to each bin based on the assigned scoring criteria.

Once the scores are incorporated into the data, they will be categorized as **categorical variables**. To utilize these scores further, their datatype needs to be converted into integers. You can confirm this by using the `data.info()` method.

By converting these scores into integers, we can effectively leverage them for in-depth analysis and segmentation in RFM analysis.

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

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
segment_labels = ['Low-Value', 'Mid-Value', 'High-Value']
data['Value Segment'] = pd.qcut(data['RFM_Score'], q=3, labels=segment_labels)
```

To determine the **RFM score**, we summed up the scores obtained for recency, frequency, and monetary value. For instance, if a customer has a recency score of 3, a frequency score of 4, and a monetary score of 5, their RFM score would be 12.

Following the calculation of RFM scores, we proceeded to create **RFM segments** based on these scores. The RFM scores were divided into three distinct segments: **"Low-Value," "Mid-Value," and "High-Value."** This segmentation was achieved using the `pd.qcut()` function, which evenly distributes scores between segments.

By segmenting customers based on their RFM scores, we can effectively categorize them into different value groups, enabling targeted marketing strategies and tailored approaches to customer engagement and retention.

```
print(data.head())
```

```
   CustomerID PurchaseDate  TransactionAmount ProductInformation  OrderID  \
0        8814   2023-04-11             943.31          Product C   890075
1        2188   2023-04-11             463.70          Product A   176819
2        4608   2023-04-11              80.28          Product A   340062
3        2559   2023-04-11             221.29          Product A   239145
4        9482   2023-04-11             739.56          Product A   194545

   Location  Recency  Frequency  MonetaryValue  RecencyScore  FrequencyScore  \
0     Tokyo      344          1         943.31             1               1
1    London      344          1         463.70             1               1
2  New York      344          1          80.28             1               1
3    London      344          1         221.29             1               1
4     Paris      344          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
```

To analyze the segment distribution, we'll examine the distribution of customers across the "Low-Value," "Mid-Value," and "High-Value" segments based on their RFM scores. This distribution provides valuable insights into the composition of

customer segments and their relative proportions within the dataset. By understanding the distribution, businesses can tailor their strategies to cater to each segment effectively.

We can visualize the segment distribution using various techniques such as histograms, bar plots, or pie charts. These visualizations will depict the proportion of customers in each segment, allowing for easy interpretation and decision-making.

```
# RFM Segment Distribution
segment_counts = data['Value Segment'].value_counts().reset_index()
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=pastel_colors,
                          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()
```

To create RFM Customer Segments, we'll define broader classifications based on the RFM scores, which offer a strategic perspective on customer behavior and characteristics in terms of recency, frequency, and monetary aspects. These segments, such as **"Champions," "Potential Loyalists," and "Can't Lose,"** provide insights into the overall value and potential of customers within the dataset.

Here's how we can create the RFM Customer Segments:

1. **Champions**: These are customers with high RFM scores across all three dimensions – recency, frequency, and monetary value. They are the most valuable customers who recently made frequent purchases and spent significant amounts. They are key drivers of revenue and should be targeted for loyalty programs and personalized offers to maintain their engagement.
2. **Potential Loyalists**: This segment consists of customers with high recency and frequency scores but lower monetary scores. While they may have made frequent purchases recently, their overall spending may not be as high as

Champions. However, they show potential for becoming high-value customers with targeted marketing efforts aimed at increasing their average transaction value.

3. **Can't Lose**: These customers have high monetary scores but lower recency and frequency scores. They may not have made recent purchases or bought frequently, but when they do, they spend significantly. It's essential to keep them engaged to prevent churn and encourage repeat purchases by offering incentives or exclusive deals.

By categorizing customers into these segments, businesses can tailor their marketing strategies and engagement tactics to address the unique needs and behaviors of each segment effectively. This targeted approach can help maximize customer lifetime value, drive retention, and boost overall profitability.

```
# 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 Customer Segments'] = 'Potential Loyalists'
data.loc[(data['RFM_Score'] >= 5) & (data['RFM_Score'] < 6), 'RFM Customer Segments'] = 'At Risk Customers'
data.loc[(data['RFM_Score'] >= 4) & (data['RFM_Score'] < 5), 'RFM Customer Segments'] = "Can't Lose"
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']])
```

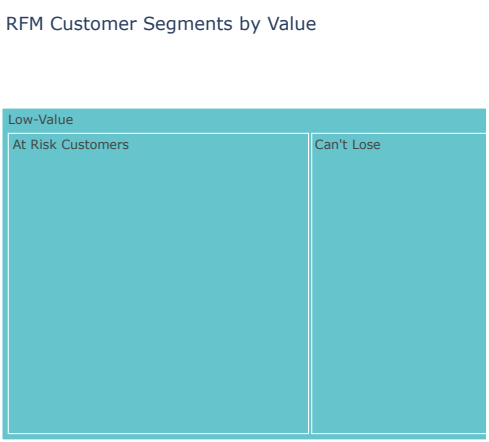
```
     CustomerID RFM Customer Segments
0          8814            Can't Lose
1          2188                  Lost
2          4608                  Lost
3          2559                  Lost
4          9482            Can't Lose
..          ...                   ...
995        2970   Potential Loyalists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

[1000 rows x 2 columns]
```

RFM Analysis

To analyze the distribution of customers across different RFM customer segments within each value segment, we will examine how customers are distributed among the broader classifications such as "Champions," "Potential Loyalists," and "Can't Lose" within each RFM value segment. This analysis provides insights into the composition of customer segments and their relative proportions across different levels of customer value.

We can visualize this distribution using stacked bar plots or pie charts, where each segment (Champions, Potential Loyalists, Can't Lose) is represented as a portion of the whole within each RFM value segment. This visualization helps in understanding the distribution patterns and identifying areas of opportunity for targeted marketing efforts and engagement strategies.

```
segment_product_counts = data.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.Paste
                                         title='RFM Customer Segments by Value')
fig_treemap_segment_product.show()
```

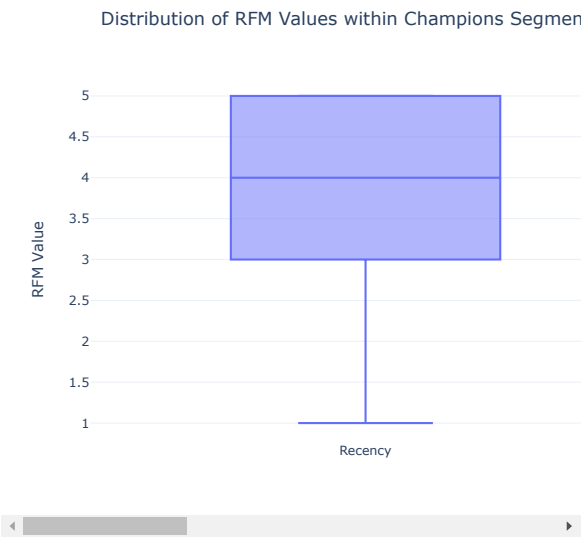


To analyze the distribution of RFM values within the "Champions" segment, we'll focus on examining how recency, frequency, and monetary values are distributed among customers categorized as "Champions." This analysis provides insights into the specific characteristics and behaviors of customers who are considered highly valuable and active.

We can visualize the distribution of RFM values within the "Champions" segment using histograms, density plots, or box plots. These visualizations help in understanding the range and distribution of recency, frequency, and monetary values among the "Champions" segment, enabling deeper insights into their purchasing patterns and engagement levels.

Additionally, we can calculate summary statistics such as mean, median, and quartiles for each RFM value within the "Champions" segment to provide a quantitative understanding of their distribution.

```
# 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='Frequency'))
fig.add_trace(go.Box(y=champions_segment['MonetaryScore'], name='Monetary'))

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

fig.show()
```

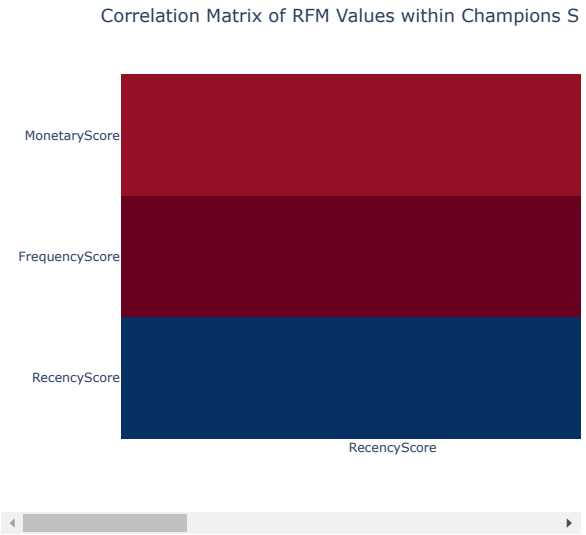


To analyze the correlation of the recency, frequency, and monetary scores within the "Champions" segment, we'll examine how these three RFM metrics are related to each other among customers classified as "Champions." This analysis provides insights into whether there are any patterns or relationships among recency, frequency, and monetary values within this highly valuable customer segment.

We can calculate the correlation coefficient between recency, frequency, and monetary scores within the "Champions" segment to measure the strength and direction of their linear relationship. The correlation coefficient ranges from -1 to 1, where:

A correlation coefficient close to 1 indicates a strong positive correlation (i.e., as one variable increases, the other also tends to increase). A correlation coefficient close to -1 indicates a strong negative correlation (i.e., as one variable increases, the other tends to decrease). A correlation coefficient close to 0 indicates little to no linear relationship between the variables. Additionally, we can visualize the correlation matrix using a heatmap to provide a graphical representation of the correlation between recency, frequency, and monetary scores within the "Champions" segment. This visualization helps in identifying any significant correlations and understanding the overall relationship between these RFM metrics.

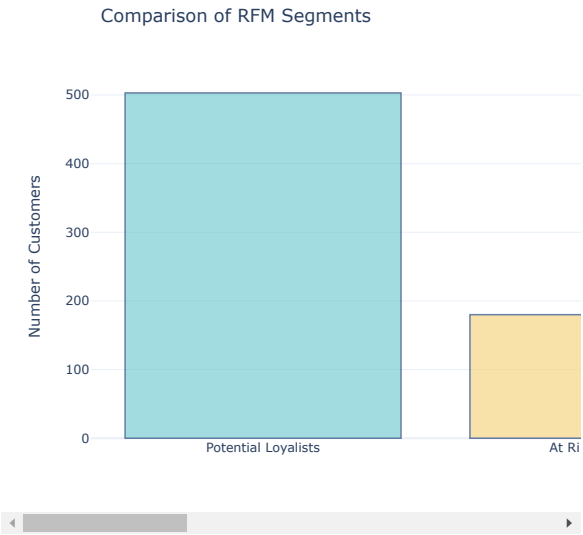
```
correlation_matrix = champions_segment[['RecencyScore', 'FrequencyScore', 'MonetaryScore']].corr()

# 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='RdBu',
                   colorbar=dict(title='Correlation')))

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

fig_heatmap.show()
```



To examine the number of customers in all segments, we'll gather data on the distribution of customers across each RFM segment, including "Low-Value," "Mid-Value," and "High-Value," as well as the RFM Customer Segments such as "Champions," "Potential Loyalists," and "Can't Lose."

We can present this information using various visualization techniques such as bar charts or pie charts. These visualizations will provide a clear overview of the customer distribution across different segments, allowing for easy comparison and identification of dominant segments.

```
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.values,
                             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' else pastel_colors[i]
                                for i, segment in enumerate(segment_counts.index)],
                  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()
```

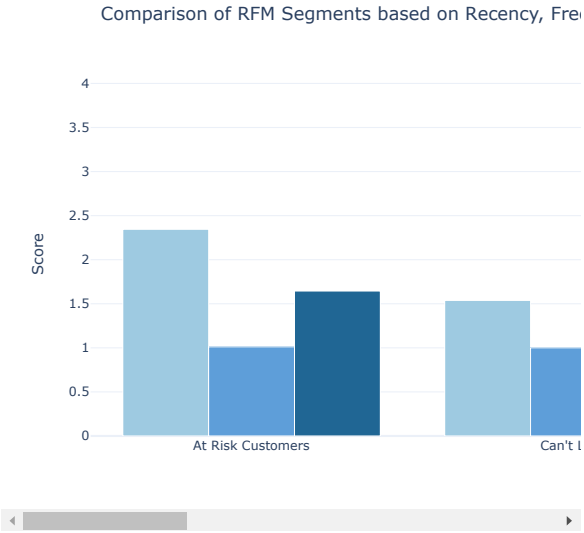


To analyze the recency, frequency, and monetary scores of all segments, we'll gather data on these RFM metrics for each segment, including "Low-Value," "Mid-Value," and "High-Value," as well as the RFM Customer Segments such as "Champions," "Potential Loyalists," and "Can't Lose."

We can present this information using summary statistics such as mean, median, standard deviation, and quartiles for each RFM metric within each segment. Additionally, we can visualize the distribution of these scores using box plots or histograms to provide a visual comparison across segments.

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

# 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-55-cb2923398eb1>:2: FutureWarning:

Indexing with multiple keys (implicitly converted to a tuple of keys) w
```




```
!pip install dash
```

```
import dash
from dash import dcc, html
from dash.dependencies import Input, Output
import plotly.io as pio
import plotly.colors as pc
```

```
Requirement already satisfied: dash in /usr/local/lib/python3.10/dist-packages (2.16.1)
Requirement already satisfied: Flask<3.1,>=1.0.4 in /usr/local/lib/python3.10/dist-packages (from dash) (2.2.5)
Requirement already satisfied: Werkzeug<3.1 in /usr/local/lib/python3.10/dist-packages (from dash) (3.0.1)
Requirement already satisfied: plotly>=5.0.0 in /usr/local/lib/python3.10/dist-packages (from dash) (5.15.0)
Requirement already satisfied: dash-html-components==2.0.0 in /usr/local/lib/python3.10/dist-packages (from das
Requirement already satisfied: dash-core-components==2.0.0 in /usr/local/lib/python3.10/dist-packages (from das
Requirement already satisfied: dash-table==5.0.0 in /usr/local/lib/python3.10/dist-packages (from dash) (5.0.0)
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.10/dist-packages (from dash) (7.0.2
Requirement already satisfied: typing-extensions>=4.1.1 in /usr/local/lib/python3.10/dist-packages (from dash)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from dash) (2.31.0)
Requirement already satisfied: retrying in /usr/local/lib/python3.10/dist-packages (from dash) (1.3.4)
Requirement already satisfied: nest-asyncio in /usr/local/lib/python3.10/dist-packages (from dash) (1.6.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from dash) (67.7.2)
Requirement already satisfied: Jinja2>=3.0 in /usr/local/lib/python3.10/dist-packages (from Flask<3.1,>=1.0.4->
Requirement already satisfied: itsdangerous>=2.0 in /usr/local/lib/python3.10/dist-packages (from Flask<3.1,>=1
Requirement already satisfied: click>=8.0 in /usr/local/lib/python3.10/dist-packages (from Flask<3.1,>=1.0.4->d
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from plotly>=5.0.0->
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from plotly>=5.0.0->dash)
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dist-packages (from Werkzeug<3.1-
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.10/dist-packages (from importlib-metadata->d
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from reques
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->dash) (3
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->da
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->da
Requirement already satisfied: six>=1.7.0 in /usr/local/lib/python3.10/dist-packages (from retrying->dash) (
```

This line initializes a new Dash application. The `__name__` variable is used to help determine the root path for the application. By using `__name__`, Dash can identify the main module where the application is defined, allowing it to appropriately locate

and organize the application's resources and routes.

```
# Initialize the Dash app
app = dash.Dash(__name__)
```

This description outlines the structure of a Dash dashboard using HTML and Dash components:

- **html.Div**: This is a container that holds all other Dash and HTML components, representing a division or section in an HTML document. It defines the structure of the dashboard.

Inside the `html.Div`, the following components are included:

- **html.H1**: This creates a header (H1 tag in HTML) for the title of the dashboard. The `className` parameter is used to apply CSS classes for styling.
- **html.Div**: This component contains a text description providing context about the dashboard. It is styled using the `className` parameter.
- **dcc.Dropdown**: This creates a dropdown menu that allows users to select the type of chart they want to display. The `options` parameter is a list of dictionaries, each representing a different chart option the user can select. The `value` parameter sets the default value displayed in the dropdown.
- **dcc.Graph**: This component serves as a container for the graph that will display the selected chart. The `id` parameter is used to identify this component in callbacks, while the `className` parameter applies CSS classes for styling.

This structure defines the layout and interactive elements of the RFM Analysis Dashboard, providing users with options to explore different charts and visualize customer segments based on RFM scores.

```
# Define the app layout using Bootstrap components
app.layout = html.Div([
    html.H1("RFM Analysis Dashboard", className="text-center mb-4"),
    html.Div("Analyze customer segments based on RFM scores.", className="text-center mb-4"),

    # Dropdown for selecting the chart
    dcc.Dropdown(
        id='chart-type-dropdown',
        options=[
            {'label': 'RFM Value Segment Distribution', 'value': 'segment_distribution'},
            {'label': 'Distribution of RFM Values within Customer Segment', 'value': 'RFM_distribution'},
            {'label': 'Correlation Matrix of RFM Values within Champions Segment', 'value': 'correlation_matrix'},
            {'label': 'Comparison of RFM Segments', 'value': 'segment_comparison'},
            {'label': 'Comparison of RFM Segments based on Scores', 'value': 'segment_scores'},
        ],
        value='segment_distribution',  # Default selection
        className="mb-4",
    ),

    # Graph container
    dcc.Graph(id='rfm-chart', className="mb-4"),
])
```

The `@app.callback` decorator is a key feature in Dash, converting a Python function into a callback function. This function is automatically invoked whenever there's a change in the specified input component's property.

The `Output('rfm-chart', 'figure')` part of the decorator indicates that the output of the callback will modify the 'figure' property of the component with the id 'rfm-chart'. This means that the callback function will update the chart displayed in the 'rfm-chart' component.

The `Input('chart-type-dropdown', 'value')` portion specifies the input for the callback. It monitors changes in the 'value' property of the component with the id 'chart-type-dropdown'. Whenever a new option is selected from the dropdown, triggering a change in value, the callback function is executed.

The `def update_chart(selected_chart_type)` function is the callback function itself. It's called when the input (selected_chart_type) changes. This function receives the new value of the dropdown as an argument and returns the figure

corresponding to the selected chart type. In essence, it dynamically updates the chart displayed based on the user's selection from the dropdown menu.

```
# Define callback to update the selected chart
@app.callback(
    Output('rfm-chart', 'figure'),
    [Input('chart-type-dropdown', 'value')]
)
def update_chart(selected_chart_type):
    if selected_chart_type == 'segment_distribution':
        return fig_segment_dist
    elif selected_chart_type == 'RFM_distribution':
        return fig_treemap_segment_product
    elif selected_chart_type == 'correlation_matrix':
```