**Group and Calculate Transitions**:****

* + Group the data by page and next\_page to calculate the frequency of transitions.
  + Sort transitions by count in descending order to prioritize the most common paths.

1. **Start with the Most Common First Page**:
   * Initialize the current\_page with the page that has the highest transition frequency.
2. **Construct the Path Iteratively**:
   * Add the current\_page to the happy\_path.
   * Check the is\_exit column to determine if the current\_page is an exit page. If is\_exit == 1, append "Exit" and terminate.
3. **Validate Transitions Using** page\_referrer:
   * Ensure that the page\_referrer matches the previous\_page to validate the transition.
   * Use this condition to filter valid next pages:

python

CopyEdit

next\_pages = transitions[(transitions['page'] == current\_page) &

(group['page\_referrer'] == previous\_page)]

1. **Handle Empty Transitions**:
   * If no valid next pages exist, append "Exit" to the happy\_path and break the loop.
2. **Return the Happy Path**:
   * Convert the happy\_path list into a string with arrows (->) indicating the sequence of pages.

### # Function to calculate the happy path

def calculate\_happy\_path(group):

# Check if the group is empty

if group.empty:

return "No Data"

# Calculate transition frequencies for the journey

transitions = group.groupby(['page', 'next\_page']).size().reset\_index(name='count')

transitions = transitions.sort\_values('count', ascending=False)

# Ensure transitions are valid

if transitions.empty:

return "No Valid Transitions"

# Start constructing the happy path

happy\_path = []

current\_page = transitions.iloc[0]['page'] # Start with the most common first page

previous\_page = None

while True:

# Add the current page to the happy path

happy\_path.append(current\_page)

# Check if the current page is an exit page

is\_exit = group[group['page'] == current\_page]['is\_exit'].iloc[0]

if is\_exit == 1: # Terminate if the current page is marked as exit

happy\_path.append('Exit')

break

# Get the most common valid transition from the current page

next\_pages = transitions[(transitions['page'] == current\_page) &

(group['page\_referrer'] == previous\_page)] # Validate page\_referrer

# If no valid next pages exist, terminate the loop

if next\_pages.empty:

happy\_path.append('Exit')

break

# Update the current and previous pages

previous\_page = current\_page

current\_page = next\_pages.iloc[0]['next\_page']

return " -> ".join(happy\_path)

# Sort data by user journey

df\_sorted = df.sort\_values(['journey\_name', 'account\_num', 'exit\_time'])

# Apply the logic journey-wise

happy\_paths = df\_sorted.groupby('journey\_name').apply(calculate\_happy\_path).reset\_index(name='happy\_path')

# Display the happy paths

print("Happy Paths for Each Journey:")

print(happy\_paths)

# Create the Plotly figure with arrows representing transition weights

fig = go.Figure()

# Add edge traces with arrows proportional to the weight

for edge in G.edges(data=True):

x0, y0 = pos[edge[0]]

x1, y1 = pos[edge[1]]

weight = edge[2]['weight']

# Add an arrow annotation for the transition with width proportional to weight

fig.add\_annotation(

x=x1, y=y1, ax=x0, ay=y0,

xref='x', yref='y', axref='x', ayref='y',

showarrow=True,

arrowhead=2, # Arrowhead style

arrowsize=2, # Arrow size

arrowwidth=0.5 + 0.5 \* weight, # Arrow width proportional to weight

arrowcolor='blue',

opacity=0.8

)

# Add hover text for the edge

fig.add\_trace(go.Scatter(

x=[(x0 + x1) / 2], # Position the hover text at the midpoint

y=[(y0 + y1) / 2],

text=f"{edge[0]} → {edge[1]}: {weight} visits",

mode='text',

hoverinfo='text',

textfont=dict(size=10, color='darkblue')

))

# Add node traces

node\_trace = go.Scatter(

x=node\_x,

y=node\_y,

mode='markers+text',

text=node\_text, # Only node names

hoverinfo='text',

textposition="top center",

marker=dict(

color=['red' if 'Exit' in node else 'lightblue' for node in G.nodes()],

size=20,

line\_width=2

)

)

# Add the node trace to the figure

fig.add\_trace(node\_trace)

# Update the layout

fig.update\_layout(

title=f"Interactive Journey Visualization with Weighted Arrows for User {user\_to\_visualize}",

titlefont\_size=16,

showlegend=False,

hovermode='closest',

margin=dict(b=0, l=0, r=0, t=40),

xaxis=dict(showgrid=False, zeroline=False),

yaxis=dict(showgrid=False, zeroline=False)

)

# Save the updated visualization

output\_file\_with\_weighted\_arrows = "/mnt/data/journey\_visualization\_weighted\_arrows.html"

fig.write\_html(output\_file\_with\_weighted\_arrows)

output\_file\_with\_weighted\_arrows

# Ensure transition weights are represented in the graph

if len(journeys) == 0:

print(f"No journey data to plot.")

else:

# Iterate over each journey\_name to create graphs for each journey

for journey\_name, journey\_edges in journeys.itertuples(index=False):

if len(journey\_edges) == 0:

print(f"No journey data to plot for Journey: {journey\_name}.")

continue

# Create a directed graph

G = nx.DiGraph()

# Add edges with weights (transition counts)

for edge in journey\_edges:

source, target = edge[0], edge[1]

if len(edge) > 2: # Check if weight is included in edge

weight = edge[2]

else:

weight = 1 # Default weight if not specified

G.add\_edge(source, target, weight=weight)

# Debugging graph data

print(f"Journey: {journey\_name}")

print(f"Nodes: {G.nodes()}")

print(f"Edges: {G.edges(data=True)}")

# Customize node colors (highlight Exit nodes in red)

node\_colors = ['red' if 'Exit' in node else 'lightblue' for node in G.nodes()]

# Generate a layout for better clarity

pos = nx.spring\_layout(G, seed=42)

# Plot the graph

plt.figure(figsize=(14, 10))

edge\_labels = nx.get\_edge\_attributes(G, 'weight') # Retrieve weights for edges

nx.draw\_networkx\_nodes(G, pos, node\_color=node\_colors, node\_size=1000, edgecolors='black')

nx.draw\_networkx\_edges(G, pos, width=2, edge\_color='blue', arrowsize=20)

nx.draw\_networkx\_labels(G, pos, font\_size=10, font\_color='black', verticalalignment='center')

nx.draw\_networkx\_edge\_labels(G, pos, edge\_labels=edge\_labels, font\_size=8, font\_color='darkgreen')

plt.title(f"Journey Visualization with Transition Weights for Journey: {journey\_name}", fontsize=16, pad=20)

plt.tight\_layout()

plt.show()

# Print the ordered journey

ordered\_journey = " -> ".join([edge[0] for edge in journey\_edges] + [journey\_edges[-1][1]])

print(f"Ordered Journey for {journey\_name}:")

print(ordered\_journey)