### Insights for the Management Meeting:

1. **Page Number Distribution**:
   * The distribution of page flows shows a significant drop-off in frequency as the page number increases. This suggests that users interact less with pages of higher numbers.
   * Management might consider investigating the reasons behind this drop-off. Is it due to content relevance, navigation challenges, or simply user fatigue?
2. **Time Spent on Pages (Raw and Log-Transformed)**:
   * The raw time spent on pages is heavily skewed, with most users spending very little time on pages, and a few outliers spending significantly more time.
   * After applying a logarithmic transformation, the time spent on pages shows a more normalized distribution. This transformation highlights variability across pages that would otherwise be masked.
   * Actionable Insight: Focus on improving content and engagement strategies on pages where users tend to spend minimal time.
3. **User Behavior Patterns**:
   * The patterns from the histograms indicate potential areas of improvement in user flow or page structure. Higher engagement on initial pages may mean they are better optimized than subsequent ones.
   * Outliers in time spent could represent either highly engaged users or those facing technical issues. Investigating these outliers may provide additional insights.
4. **Optimization Areas**:
   * Consider optimizing high-traffic, low-time-spent pages for better engagement or conversion.
   * Investigate drop-off points in page flow to identify possible navigation or content barriers.

### ****Equation for Path Construction:****

* P = page (current page)
* N = next\_page (next page)
* E = is\_exit (exit flag, where E = 1 indicates exit)
* The equation for the path column can be expressed as:

The equation for the path column can be expressed as:

path={P+"−>"+N+"(Exit)"if E=1P+"−>"+Nif E≠1\text{path} = \begin{cases} P + " -> " + N + " (Exit)" & \text{if } E = 1 \\ P + " -> " + N & \text{if } E \neq 1 \end{cases}path={P+"−>"+N+"(Exit)"P+"−>"+N​if E=1if E=1​

For example, a journey might look like 'Home -> Product Page (Exit),' highlighting where we lose engagement. By grouping these paths by journey names, we can analyze patterns and pinpoint areas for improvement. This helps us identify bottlenecks or content gaps that cause users to leave, enabling us to optimize the user experience and improve retention.

def build\_valid\_journey(group):

journey = []

previous\_page = None

for \_, row in group.iterrows():

# Ensure strings for 'page' and 'next\_page'

current\_page = str(row['page'])

next\_page = str(row['next\_page'])

# Validate transitions and handle exit

if previous\_page is None or str(row['page\_referrer']) == previous\_page:

if int(row.get('is\_exit', 0)) == 1: # Check if it's an exit

journey.append(current\_page + " -> " + next\_page + " (Exit)")

else:

journey.append(current\_page + " -> " + next\_page)

previous\_page = current\_page

return journey

### ****Updated Equation:****

Let:

* JJJ: Journey list.
* PcurrentP\_{\text{current}}Pcurrent​: Current page (string/object).
* PnextP\_{\text{next}}Pnext​: Next page (string/object).
* RcurrentR\_{\text{current}}Rcurrent​: page\_referrer of the current page (string/object).
* PprevP\_{\text{prev}}Pprev​: Previous page (initialized as None).
* EEE: is\_exit (exit flag, where E=1E = 1E=1 indicates an exit page).

The logic for appending valid transitions to the journey list JJJ is expressed as:

J={J∪{(Pcurrent+"−>"+Pnext+"(Exit)")}if Pprev=None or Rcurrent=Pprev, and E=1J∪{(Pcurrent+"−>"+Pnext)}if Pprev=None or Rcurrent=Pprev, and E≠1JotherwiseJ = \begin{cases} J \cup \{(P\_{\text{current}} + " -> " + P\_{\text{next}} + " (Exit)")\} & \text{if } P\_{\text{prev}} = \text{None or } R\_{\text{current}} = P\_{\text{prev}}, \text{ and } E = 1 \\ J \cup \{(P\_{\text{current}} + " -> " + P\_{\text{next}})\} & \text{if } P\_{\text{prev}} = \text{None or } R\_{\text{current}} = P\_{\text{prev}}, \text{ and } E \neq 1 \\ J & \text{otherwise} \end{cases}J=⎩⎨⎧​J∪{(Pcurrent​+"−>"+Pnext​+"(Exit)")}J∪{(Pcurrent​+"−>"+Pnext​)}J​if Pprev​=None or Rcurrent​=Pprev​, and E=1if Pprev​=None or Rcurrent​=Pprev​, and E=1otherwise​

# Function to build valid journeys with exit page handling

def build\_valid\_journey(group):

journey = []

previous\_page = None

for \_, row in group.iterrows():

current\_page = str(row['page']) # Ensure page is a string

next\_page = str(row['next\_page']) # Ensure next\_page is a string

# Validate transitions using page\_referrer

if previous\_page is None or str(row['page\_referrer']) == previous\_page:

if int(row.get('is\_exit', 0)) == 1: # Check if it's an exit

journey.append((current\_page, next\_page + " (Exit)"))

else:

journey.append((current\_page, next\_page))

previous\_page = current\_page

return journey

# Calculate exit time

df['exit\_time'] = df['hit\_date\_time'] + pd.to\_timedelta(df['time\_spent\_seconds'], unit='s')

# Sort data by user, channel\_visit\_id, and exit\_time

df\_sorted = df.sort\_values(['account\_num', 'channel\_visit\_id', 'exit\_time'])

# Apply the build\_valid\_journey function for each user

journeys = df\_sorted.groupby('account\_num').apply(build\_valid\_journey).reset\_index(name='journey')

# Display journeys

print("Customer Journeys with Time and Referrer Validation:")

print(journeys)

# Visualize a journey for a specific user

user\_to\_visualize = 210704245

journey\_edges = journeys[journeys['account\_num'] == user\_to\_visualize]['journey'].iloc[0]

# Ensure journey\_edges is valid

if isinstance(journey\_edges, list) and len(journey\_edges) > 0:

journey\_edges = [(edge[0], edge[1]) for edge in journey\_edges if isinstance(edge, tuple) and len(edge) >= 2]

else:

print(f"No valid journey edges found for User {user\_to\_visualize}.")

journey\_edges = []

# Check if journey\_edges is not empty

if len(journey\_edges) == 0:

print(f"No journey data to plot for User {user\_to\_visualize}.")

else:

# Create the directed graph

G = nx.DiGraph()

G.add\_edges\_from(journey\_edges)

# Debugging graph data

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

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

# Customize node colors and sizes

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

node\_sizes = [1000 + 500 \* G.degree(node) for node in G.nodes()]

# Generate and debug layout

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

# Plot the graph

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

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

plt.title(f"Journey Visualization for User ID: {user\_to\_visualize}", fontsize=16, pad=20)

plt.show()

# Print the ordered journey

if len(journey\_edges) > 0:

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

print(f"Ordered Journey for User {user\_to\_visualize}:")

print(ordered\_journey)

else:

print(f"No ordered journey available for User {user\_to\_visualize}.")