import pandas as pd

# Example DataFrame (Replace with your actual data)

data = {

'journey\_name': ['payment', 'payment', 'payment', 'browse', 'browse', 'payment'],

'user\_id': [101, 101, 101, 102, 102, 101],

'channel\_visit\_id': [1, 1, 2, 3, 3, 2],

'hit\_date\_time': [

'2024-10-08 10:00:00', '2024-10-08 10:05:00', '2024-10-08 10:10:00',

'2024-10-08 11:00:00', '2024-10-08 11:05:00', '2024-10-08 10:15:00'

],

'page': ['Page A', 'Page B', 'Page C', 'Page D', 'Page E', 'Page F'],

'next\_page': ['Page B', 'Page C', 'Exit', 'Page E', 'Exit', 'Exit'],

'time\_spent\_seconds': [300, 600, 900, 1200, 600, 300],

'page\_referrer': [None, 'Page A', 'Page B', None, 'Page D', 'Page B']

}

df = pd.DataFrame(data)

# Convert `hit\_date\_time` to datetime

df['hit\_date\_time'] = pd.to\_datetime(df['hit\_date\_time'])

# Sort by `journey\_name`, `user\_id`, and `hit\_date\_time`

df\_sorted = df.sort\_values(['journey\_name', 'user\_id', 'hit\_date\_time'])

# Group by journey and calculate transitions

def calculate\_happy\_path(group):

# Calculate transition frequencies for the journey

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

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

# Start constructing the happy path

happy\_path = []

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

while True:

happy\_path.append(current\_page)

# Get the most common transition from the current page

next\_pages = transitions[transitions['page'] == current\_page]

if next\_pages.empty or next\_pages.iloc[0]['next\_page'] == 'Exit':

happy\_path.append('Exit')

break

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

return " → ".join(happy\_path)

# Apply the logic journey-wise

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

# Display Happy Paths

print("Happy Paths for Each Journey:")

print(happy\_paths)

import pandas as pd

import networkx as nx

import matplotlib.pyplot as plt

# Example DataFrame (Replace with actual data)

data = {

'user\_id': [101, 101, 101, 102, 102, 101],

'channel\_visit\_id': [1, 1, 2, 3, 3, 2],

'hit\_date\_time': [

'2024-10-08 10:00:00', '2024-10-08 10:05:00', '2024-10-08 10:10:00',

'2024-10-08 11:00:00', '2024-10-08 11:05:00', '2024-10-08 10:15:00'

],

'page': ['Page A', 'Page B', 'Page C', 'Page D', 'Page E', 'Page F'],

'next\_page': ['Page B', 'Page C', 'Exit', 'Page E', 'Exit', 'Exit'],

'time\_spent\_seconds': [300, 600, 900, 1200, 600, 300],

'page\_referrer': [None, 'Page A', 'Page B', None, 'Page D', 'Page B']

}

df = pd.DataFrame(data)

# Convert `hit\_date\_time` to datetime

df['hit\_date\_time'] = pd.to\_datetime(df['hit\_date\_time'])

# 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(['user\_id', 'channel\_visit\_id', 'exit\_time'])

# Build the journey path by validating `page\_referrer`

def build\_valid\_journey(group):

journey = []

previous\_page = None

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

# Validate the transition using `page\_referrer`

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

journey.append((row['page'], row['next\_page']))

previous\_page = row['page']

return journey

# Apply the logic for each user

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

# Visualize a journey for a specific `user\_id`

user\_to\_visualize = 101

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

# Create a directed graph

G = nx.DiGraph()

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

# Customize node colors (highlight Exit nodes in red)

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

# Calculate node sizes based on their degree

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

# Generate a structured layout

pos = nx.shell\_layout(G) # Try nx.planar\_layout or nx.kamada\_kawai\_layout as alternatives

# Plot the graph

plt.figure(figsize=(14, 8))

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')

# Add edge labels (optional)

edge\_labels = {(u, v): f"→ {v}" for u, v in G.edges()}

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

# Add title and improve spacing

plt.title(f"Journey Visualization for User ID: {user\_to\_visualize}", fontsize=14)

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 User {user\_to\_visualize}:")

print(ordered\_journey)