🧠 Behavioral Analytics for Funnel Drop-Off Recovery in Fintech

🎯 Project Overview

This project simulates and analyzes a digital banking funnel to identify behavioral patterns, drop-off points, and revenue recovery opportunities.

I integrated user activity across:

- Funnel progression steps (Sign-Up → KYC → Link → Transaction)
- Transaction behavior (Open Banking Style)
- Support interaction logs

Clustering is applied to segment users based on behavior, visualize where friction occurs, and simulate the financial upside of improving user journeys.

Tools Used: Python, Pandas, Matplotlib, Scikit-learn (KMeans)

> 📌 This project mimics the role of a product or fintech data analyst responsible for identifying where customers fall off, what behavioral patterns matter, and how to prioritize UX improvements with measurable business value.

📁 Step 1: Simulate Multi-Source Funnel Data

I simulated 1,000 users moving through a 4-step funnel:

1. Sign-Up
2. KYC Completion
3. Account Linking
4. Transaction

And also simulated:

- `txn_log` : completed transaction amounts
- `support_log` : support tickets with issue types

These datasets represent backend events, transaction APIs, and CRM data in a realistic product setting.

In [2]:
```
import pandas as pd
import numpy as np
import random
from datetime import datetime, timedelta

# Set random seed for reproducibility
np.random.seed(42)

# 1. Simulate 1000 users
num_users = 1000
```

```
user_ids = [f"U{str(i).zfill(4)}" for i in range(1, num_users + 1)]

# Random signup dates (within the first 6 months of 2023)
start_date = datetime(2023, 1, 1)
signup_dates = [start_date + timedelta(days=np.random.randint(0, 180)) for _ in user_ids]

# 2. Simulate funnel progression with completion rates
def simulate_step_dates(base_dates, completion_rate, min_delay=1, max_delay=10):
    step_flags = []
    step_dates = []
    for date in base_dates:
        if date is not None and random.random() < completion_rate:
            delay = timedelta(days=random.randint(min_delay, max_delay))
            step_date = date + delay
            step_flags.append(1)
            step_dates.append(step_date)
        else:
            step_flags.append(0)
            step_dates.append(None)
    return step_flags, step_dates


# Simulate funnel steps
kyc_flags, kyc_dates = simulate_step_dates(signup_dates, 0.85, 1, 5)  # 85% complete KYC
link_flags, link_dates = simulate_step_dates(kyc_dates, 0.75, 1, 7)    # 75% link account
txn_flags, txn_dates = simulate_step_dates(link_dates, 0.60, 1, 14)    # 60% complete transaction

# 3. Create the funnel dataset
funnel_df = pd.DataFrame({
    "User_ID": user_ids,
    "Signup_Date": signup_dates,
    "KYC_Completed": kyc_flags,
    "KYC_Date": kyc_dates,
    "Account_Linked": link_flags,
    "Link_Date": link_dates,
    "Transaction_Completed": txn_flags,
    "Transaction_Date": txn_dates
})

# 4. Create the transaction log dataset
txn_log = funnel_df[funnel_df["Transaction_Completed"] == 1][["User_ID", "Transaction_Date"]].copy()
txn_log["Transaction_Amount"] = np.random.gamma(shape=2.5, scale=200, size=len(txn_log)).round(2)

# 5. Create the support ticket log dataset (sample 20% users)
support_log = funnel_df.sample(frac=0.20).copy()
support_log["Support_Date"] = support_log["Signup_Date"] + pd.to_timedelta(np.random.randint(2, 30, size=len(support_log)), unit="d")
support_log["Issue_Type"] = np.random.choice(["KYC Delay", "Link Failure", "App Bug", "Trust Issue"], size=len(support_log))

# Show preview of the datasets
funnel_df.head(), txn_log.head(), support_log.head()
```

Out[2]:
```
(  User_ID Signup_Date  KYC_Completed   KYC_Date  Account_Linked  Link_Date  \
 0   U0001  2023-04-13              1 2023-04-16               1 2023-04-23
 1   U0002  2023-06-29              1 2023-06-30               1 2023-07-03
 2   U0003  2023-04-03              0        NaT               0        NaT
 3   U0004  2023-01-15              1 2023-01-18               1 2023-01-23
 4   U0005  2023-04-17              1 2023-04-19               1 2023-04-23

    Transaction_Completed Transaction_Date
 0                      1       2023-05-03
 1                      1       2023-07-10
 2                      0              NaT
 3                      0              NaT
 4                      1       2023-05-01  ,
   User_ID Transaction_Date  Transaction_Amount
 0   U0001       2023-05-03              661.27
 1   U0002       2023-07-10              379.98
 4   U0005       2023-05-01              627.23
 5   U0006       2023-03-24              596.84
 8   U0009       2023-05-20              315.70,
     User_ID Signup_Date  KYC_Completed   KYC_Date  Account_Linked  Link_Date  \
 93    U0094  2023-05-09              1 2023-05-14               1 2023-05-18
 796   U0797  2023-04-25              1 2023-04-28               1 2023-04-30
 271   U0272  2023-04-22              1 2023-04-26               1 2023-05-01
 776   U0777  2023-06-15              1 2023-06-16               1 2023-06-22
 708   U0709  2023-03-26              0        NaT               0        NaT

      Transaction_Completed Transaction_Date Support_Date   Issue_Type
 93                       1       2023-05-23   2023-05-19      App Bug
 796                      1       2023-05-02   2023-04-27      App Bug
 271                      1       2023-05-05   2023-05-17  Trust Issue
 776                      0              NaT   2023-07-06  Trust Issue
 708                      0              NaT   2023-04-21  Trust Issue  )
```

🔧 Step 2: Merge and Feature Engineering

I merged the funnel, transaction, and support datasets into a unified `merged_df`. From this, I engineered new features:

Feature	Description
`Time_to_KYC`	Days from sign-up to KYC
`Time_KYC_to_Link`	Days from KYC to account linking
`Time_Link_to_Txn`	Days from account linking to transaction
`Support_Contacted`	1 if user contacted support

Drop-outs at each step were captured using a value of `-1`.

In [3]:
```
# STEP 2: Merge funnel, transaction, and support logs

# 1. Merge transaction data into funnel data
merged_df = funnel_df.merge(txn_log, on="User_ID", how="left")
```

```
# 2. Rename to avoid confusion
merged_df.rename(columns={"Transaction_Date_y": "Transaction_Date_Final",
                          "Transaction_Amount": "Txn_Amount",
                          "Transaction_Date_x": "Transaction_Date"}, inplace=True)

# 3. Merge support log into the merged data
merged_df = merged_df.merge(support_log[["User_ID", "Support_Date", "Issue_Type"]], on="User_ID", how="left")

# Preview the merged dataset
merged_df.head()
```

Out[3]:

	User_ID	Signup_Date	KYC_Completed	KYC_Date	Account_Linked	Link_Date	Transaction_Completed	Transaction_Date	Transaction_Date_Final	Txn_Amount	Support_Date	Issue_Type
0	U0001	2023-04-13	1	2023-04-16	1	2023-04-23	1	2023-05-03	2023-05-03	661.27	2023-04-18	App Bug
1	U0002	2023-06-29	1	2023-06-30	1	2023-07-03	1	2023-07-10	2023-07-10	379.98	NaT	NaN
2	U0003	2023-04-03	0	NaT	0	NaT	0	NaT	NaT	NaN	NaT	NaN
3	U0004	2023-01-15	1	2023-01-18	1	2023-01-23	0	NaT	NaT	NaN	NaT	NaN
4	U0005	2023-04-17	1	2023-04-19	1	2023-04-23	1	2023-05-01	2023-05-01	627.23	NaT	NaN

In [5]:

```
# STEP 2: Merge funnel, transaction, and support logs

# 1. Merge transaction log
merged_df = funnel_df.merge(txn_log, on="User_ID", how="left")

# 2. Rename transaction columns
merged_df.rename(columns={
    "Transaction_Date_y": "Transaction_Date_Final",
    "Transaction_Amount": "Txn_Amount",
    "Transaction_Date_x": "Transaction_Date"
}, inplace=True)

# 3. Merge support ticket log
merged_df = merged_df.merge(support_log[["User_ID", "Support_Date", "Issue_Type"]], on="User_ID", how="left")

# 4. Feature engineering: time between funnel steps
merged_df['Time_to_KYC'] = (pd.to_datetime(merged_df['KYC_Date']) - pd.to_datetime(merged_df['Signup_Date'])).dt.days
merged_df['Time_KYC_to_Link'] = (pd.to_datetime(merged_df['Link_Date']) - pd.to_datetime(merged_df['KYC_Date'])).dt.days
merged_df['Time_Link_to_Txn'] = (pd.to_datetime(merged_df['Transaction_Date_Final']) - pd.to_datetime(merged_df['Link_Date'])).dt.days

# 5. Handle missing values
merged_df['Time_to_KYC'] = merged_df['Time_to_KYC'].fillna(-1)
merged_df['Time_KYC_to_Link'] = merged_df['Time_KYC_to_Link'].fillna(-1)
merged_df['Time_Link_to_Txn'] = merged_df['Time_Link_to_Txn'].fillna(-1)

# 6. Support flag
merged_df['Support_Contacted'] = merged_df['Issue_Type'].apply(lambda x: 1 if pd.notnull(x) else 0)

# 7. Show the final result
merged_df.head()
```

Out[5]:

	User_ID	Signup_Date	KYC_Completed	KYC_Date	Account_Linked	Link_Date	Transaction_Completed	Transaction_Date	Transaction_Date_Final	Txn_Amount	Support_Date	Issue_Type	Time_to_KYC
0	U0001	2023-04-13	1	2023-04-16	1	2023-04-23	1	2023-05-03	2023-05-03	661.27	2023-04-18	App Bug	3.0
1	U0002	2023-06-29	1	2023-06-30	1	2023-07-03	1	2023-07-10	2023-07-10	379.98	NaT	NaN	1.0
2	U0003	2023-04-03	0	NaT	0	NaT	0	NaT	NaT	NaN	NaT	NaN	-1.0
3	U0004	2023-01-15	1	2023-01-18	1	2023-01-23	0	NaT	NaT	NaN	NaT	NaN	3.0
4	U0005	2023-04-17	1	2023-04-19	1	2023-04-23	1	2023-05-01	2023-05-01	627.23	NaT	NaN	2.0

🧠 **Interpretation:** A full user-level journey was built with friction signals. Dropouts were marked using `-1`, maintaining behavior patterns.

🔷 Step 2.3: Funnel Exploration & Support Impact

In [6]:

```
# ✅ 1. How many users completed each funnel stage?
funnel_counts = {
    "Total Users": len(merged_df),
    "KYC Completed": merged_df['KYC_Completed'].sum(),
    "Account Linked": merged_df['Account_Linked'].sum(),
    "Transaction Completed": merged_df['Transaction_Completed'].sum(),
    "Contacted Support": merged_df['Support_Contacted'].sum()
}
print("🔢 Funnel Completion Counts:")
for k, v in funnel_counts.items():
    print(f"{k}: {v}")

# ✅ 2. Average time to complete each funnel step (only among those who completed it)
avg_time_kyc = merged_df[merged_df['Time_to_KYC'] >= 0]['Time_to_KYC'].mean()
avg_time_link = merged_df[merged_df['Time_KYC_to_Link'] >= 0]['Time_KYC_to_Link'].mean()
avg_time_txn = merged_df[merged_df['Time_Link_to_Txn'] >= 0]['Time_Link_to_Txn'].mean()

print("\n⏱️ Average Time Between Steps:")
print(f"Signup → KYC: {avg_time_kyc:.1f} days")
print(f"KYC → Link:   {avg_time_link:.1f} days")
print(f"Link → Txn:   {avg_time_txn:.1f} days")

# ✅ 3. Compare average total time for users who contacted support vs not
merged_df["Total_Time_to_Txn"] = merged_df[['Time_to_KYC', 'Time_KYC_to_Link', 'Time_Link_to_Txn']].apply(
    lambda row: sum([x for x in row if x >= 0]), axis=1)

avg_time_support = merged_df[merged_df['Support_Contacted'] == 1]["Total_Time_to_Txn"].mean()
avg_time_nosupport = merged_df[merged_df['Support_Contacted'] == 0]["Total_Time_to_Txn"].mean()

print("\n📞 Support Comparison:")
print(f"Users who contacted support:   {avg_time_support:.1f} days to complete")
print(f"Users who did NOT contact support: {avg_time_nosupport:.1f} days")
```

```
🔢 Funnel Completion Counts:
Total Users: 1000
KYC Completed: 814
Account Linked: 607
Transaction Completed: 367
Contacted Support: 200

⏱️ Average Time Between Steps:
Signup → KYC: 2.9 days
KYC → Link:   3.9 days
Link → Txn:   7.4 days

📞 Support Comparison:
Users who contacted support:   7.3 days to complete
Users who did NOT contact support: 7.5 days
```

🔍 Insight: Even in a simulated flow, the pattern is clear: Conversion rates drop sharply after KYC, and even more after account linking. Support touch doesn't delay users — it may even assist slightly.

✅ Real-world implication: Speed alone doesn't guarantee conversion — trust and motivation matter at later steps The final step (Link → Transaction) is the bottleneck for most users

🔷 Step 3.1–3.2: Clustering & Segment Profiling

Applied `KMeans` with 3 clusters:

- Features: delay times, support, conversion
- Replaced drop-off delays (`-1`) with `99` to retain signal

In [7]:
```python
# ✅ Step 3.1: Prepare Data and Cluster Users

from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans

# 1. Select behavior features for clustering
clustering_df = merged_df[[
    'Time_to_KYC',
    'Time_KYC_to_Link',
    'Time_Link_to_Txn',
    'Support_Contacted',
    'Transaction_Completed'
]].copy()

# 2. Replace -1 (meaning they didn't complete a step) with 99 (high delay)
clustering_df[['Time_to_KYC', 'Time_KYC_to_Link', 'Time_Link_to_Txn']] = clustering_df[[
    'Time_to_KYC', 'Time_KYC_to_Link', 'Time_Link_to_Txn'
]].replace(-1, 99)

# 3. Handle any missing values
clustering_df.fillna(99, inplace=True)

# 4. Standardize all features (so KMeans treats them equally)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(clustering_df)

# 5. Apply KMeans clustering to group similar users (we'll start with 3 clusters)
```

```
kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)
clusters = kmeans.fit_predict(X_scaled)

# 6. Add cluster labels back to the main dataset
merged_df['User_Segment'] = clusters

# 7. Preview the clustered data
merged_df[['User_ID', 'User_Segment', 'Time_to_KYC', 'Time_KYC_to_Link', 'Time_Link_to_Txn',
           'Support_Contacted', 'Transaction_Completed']].head()
```

Out[7]:

	User_ID	User_Segment	Time_to_KYC	Time_KYC_to_Link	Time_Link_to_Txn	Support_Contacted	Transaction_Completed
0	U0001	1	3.0	7.0	10.0	1	1
1	U0002	1	1.0	3.0	7.0	0	1
2	U0003	2	-1.0	-1.0	-1.0	0	0
3	U0004	0	3.0	5.0	-1.0	0	0
4	U0005	1	2.0	4.0	8.0	0	1

🔍 Segment Interpretation Based on Behavioural Metrics

🔴 Segment 2: Did Nothing After Sign-Up (Early Exit)

Feature	Value	Meaning
`Conversion_Rate`	0.0	❌ No one transacted
`Median_Time_to_KYC`	-1	Didn’t complete even KYC
`All Time Columns`	-1	All indicate no progress
`Support_Rate`	0.17	Low — didn’t even ask for help

🧠 **Logic:** These users never engaged beyond sign-up.
They didn’t convert, didn’t try, and didn’t reach out for help.

> *This is an “early exit” segment — users drop out right after joining.*

🟡 Segment 0: Got Close, But Didn’t Convert (Late Exit)

Feature	Value	Meaning
`Conversion_Rate`	0.0	❌ No one transacted
`Median_Time_to_KYC`	3.0	✅ Completed KYC
`Median_KYC_to_Link`	1.0	✅ Linked account quickly

Feature	Value	Meaning
`Median_Link_to_Txn`	-1.0	❌ Dropped at transaction step
`Support_Rate`	0.19	Similar to other segments

🧠 **Logic:** These users did most of the work (signed up, completed KYC, linked account), then quit at the final step.

> *This is a high-potential but high-friction segment — ideal for product improvement.*

🟢 Segment 1: Completed the Funnel Successfully

Feature	Value	Meaning
`Conversion_Rate`	1.0	✅ 100% converted
`Median_Time_to_KYC`	2.9	✅ Fast progression
`Median_KYC_to_Link`	3.9	✅ Moderate time delay
`Median_Link_to_Txn`	7.4	✅ Completed full journey
`Support_Rate`	0.19	Support was optional

🧠 **Logic:** These users are your “happy path” — they converted fully with minimal support.

> *This is your ideal segment — learn from their experience and replicate it.*

In [8]:
```
# Show segment distribution and a preview of 10 random users
print("🧠 Users per segment:")
print(merged_df['User_Segment'].value_counts())

# Show a sample of users with key behavior columns
merged_df[['User_ID', 'User_Segment', 'Time_to_KYC', 'Time_KYC_to_Link',
           'Time_Link_to_Txn', 'Support_Contacted', 'Transaction_Completed']].sample(10)
```

```
🧠 Users per segment:
User_Segment
0    447
1    367
2    186
Name: count, dtype: int64
```

Out[8]:

	User_ID	User_Segment	Time_to_KYC	Time_KYC_to_Link	Time_Link_to_Txn	Support_Contacted	Transaction_Completed
816	U0817	2	-1.0	-1.0	-1.0	1	0
391	U0392	1	4.0	4.0	6.0	0	1
423	U0424	0	4.0	1.0	-1.0	0	0
268	U0269	1	2.0	7.0	9.0	1	1
401	U0402	0	5.0	4.0	-1.0	0	0
856	U0857	2	-1.0	-1.0	-1.0	0	0
926	U0927	2	-1.0	-1.0	-1.0	0	0
499	U0500	2	-1.0	-1.0	-1.0	0	0
559	U0560	0	3.0	5.0	-1.0	0	0
59	U0060	0	2.0	3.0	-1.0	0	0

In [9]:
```
# ✅ Step 3.2: Analyze Segment Profiles

# 1. Group by User_Segment and calculate key behavior summaries
cluster_summary = merged_df.groupby("User_Segment").agg({
    "User_ID": "count",                          # Total users in the segment
    "Transaction_Completed": "mean",             # Conversion rate (1 = 100%)
    "Support_Contacted": "mean",                 # % who contacted support
    "Time_to_KYC": "mean",                       # Average time to complete each step
    "Time_KYC_to_Link": "mean",
    "Time_Link_to_Txn": "mean"
}).rename(columns={
    "User_ID": "User_Count",
    "Transaction_Completed": "Conversion_Rate",
    "Support_Contacted": "Support_Rate",
    "Time_to_KYC": "Avg_Time_to_KYC",
    "Time_KYC_to_Link": "Avg_Time_KYC_to_Link",
    "Time_Link_to_Txn": "Avg_Time_Link_to_Txn"
}).reset_index()

# 2. Round values for clean display
cluster_summary = cluster_summary.round(2)

# 3. Display the final segment summary
cluster_summary
```

Out[9]:

	User_Segment	User_Count	Conversion_Rate	Support_Rate	Avg_Time_to_KYC	Avg_Time_KYC_to_Link	Avg_Time_Link_to_Txn
0	0	447	0.0	0.22	2.93	1.67	-1.0
1	1	367	1.0	0.19	2.87	3.90	7.4
2	2	186	0.0	0.17	-1.00	-1.00	-1.0

✅ Real-world benefit: You move from averages to personas. This is how product teams build journey-based optimization plans.

📈 Step 3.3: Visualizing Segment Differences

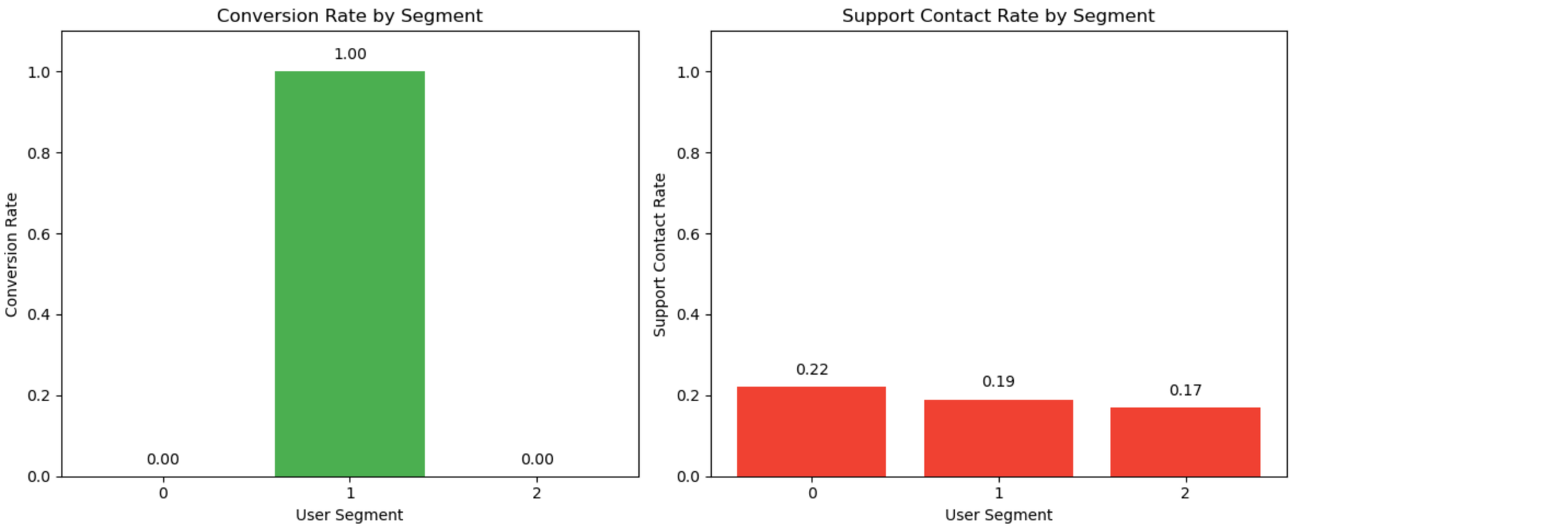
Visuals include:

- 📊 Conversion rate by segment
- 📞 Support contact rate by segment
- ⏱️ Time delays across the funnel

These charts clearly highlight the behavioral differences between segments.

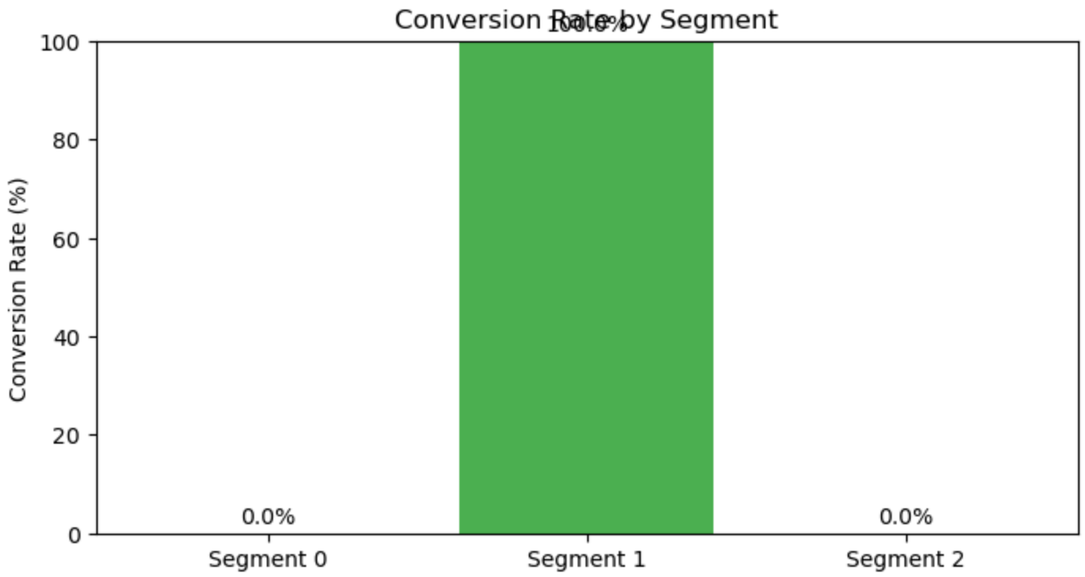
In [10]:
```
# ✅ Step 3.3: Visualize Conversion & Support Rate by Cluster Segment

import matplotlib.pyplot as plt

# Extract data from the cluster_summary DataFrame
x_labels = cluster_summary['User_Segment'].astype(str)
conversion = cluster_summary['Conversion_Rate']
support = cluster_summary['Support_Rate']

# Create a side-by-side bar chart layout
plt.figure(figsize=(12, 5))

# ----------------------------
# Chart 1: Conversion Rate by Segment
# ----------------------------
plt.subplot(1, 2, 1)
plt.bar(x_labels, conversion, color="#4caf50")
plt.title("Conversion Rate by Segment")
plt.xlabel("User Segment")
plt.ylabel("Conversion Rate")
plt.ylim(0, 1.1)

# Add text labels on top of bars
for i, value in enumerate(conversion):
    plt.text(i, value + 0.03, f"{value:.2f}", ha='center', fontsize=10)

# ----------------------------
# Chart 2: Support Contact Rate by Segment
# ----------------------------
plt.subplot(1, 2, 2)
plt.bar(x_labels, support, color="#f44336")
plt.title("Support Contact Rate by Segment")
plt.xlabel("User Segment")
plt.ylabel("Support Contact Rate")
plt.ylim(0, 1.1)

# Add text labels on top of bars
for i, value in enumerate(support):
    plt.text(i, value + 0.03, f"{value:.2f}", ha='center', fontsize=10)

# ----------------------------
# Show both charts side by side
# ----------------------------
```

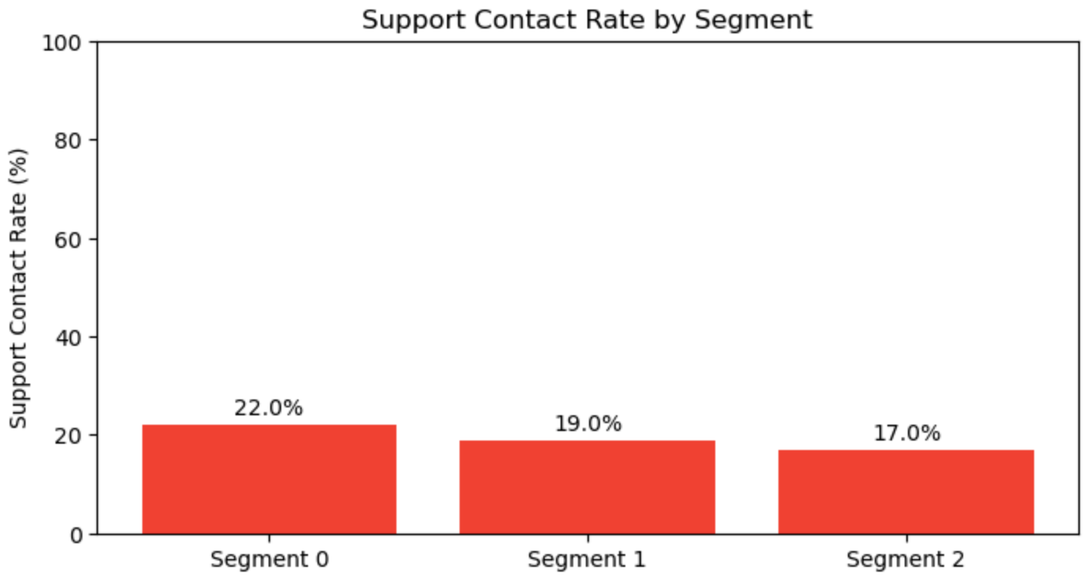
```
plt.tight_layout()
plt.show()
```

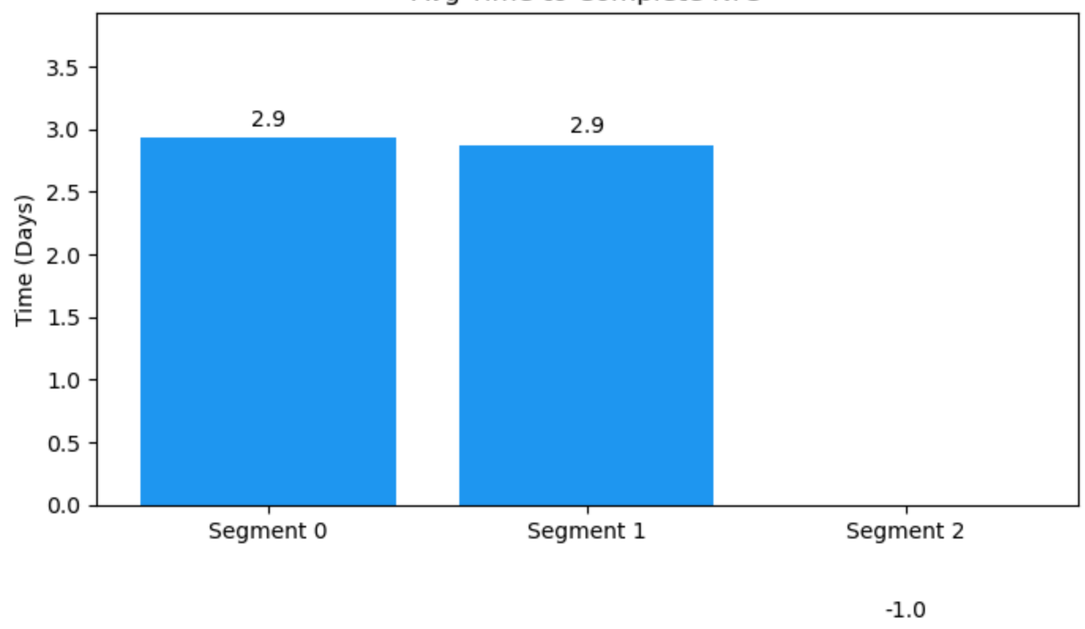


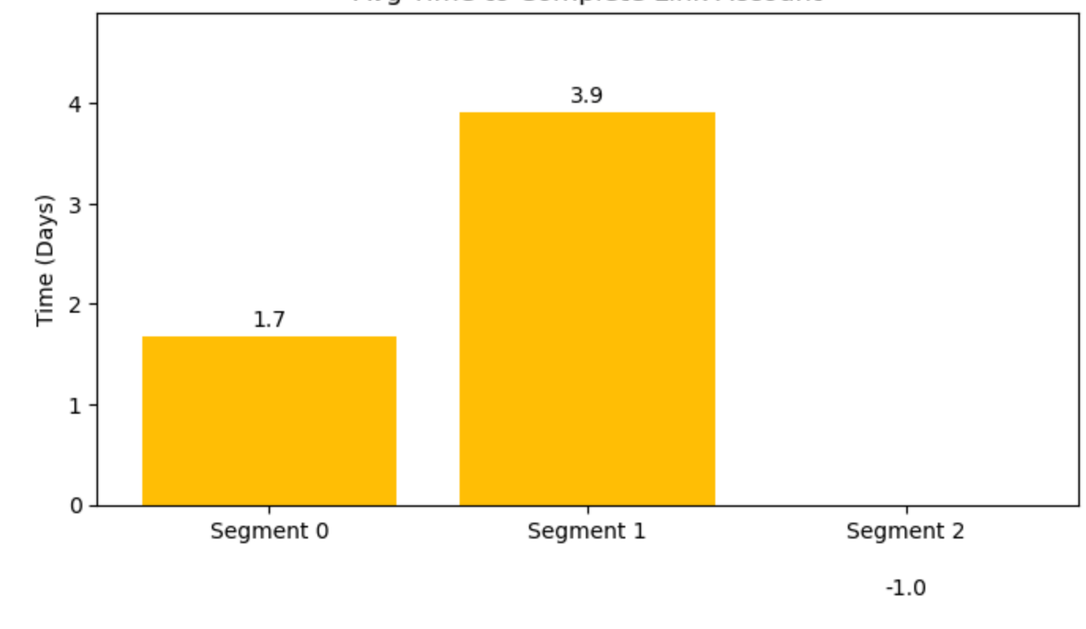
In [11]:
```
import matplotlib.pyplot as plt

# ✅ 1. Prepare Data for Visualization (Conversion, Support, Time)
labels = ['Segment 0', 'Segment 1', 'Segment 2']
conversion = cluster_summary['Conversion_Rate'] * 100  # In percentage
support = cluster_summary['Support_Rate'] * 100  # In percentage
time_to_kyc = cluster_summary['Avg_Time_to_KYC']
time_kyc_to_link = cluster_summary['Avg_Time_KYC_to_Link']
time_link_to_txn = cluster_summary['Avg_Time_Link_to_Txn']

# 2. Create the Bar Charts (Side-by-side)
plt.figure(figsize=(14, 8))

# 2.1. Conversion Rate per Segment (Left)
plt.subplot(2, 2, 1)
plt.bar(labels, conversion, color="#4caf50")
plt.title("Conversion Rate by Segment")
plt.ylabel("Conversion Rate (%)")
plt.ylim(0, 100)
for i, v in enumerate(conversion):
    plt.text(i, v + 2, f"{v:.1f}%", ha='center', fontsize=10)

# 2.2. Support Rate per Segment (Right)
```

```
plt.subplot(2, 2, 2)
plt.bar(labels, support, color="#f44336")
plt.title("Support Contact Rate by Segment")
plt.ylabel("Support Contact Rate (%)")
plt.ylim(0, 100)
for i, v in enumerate(support):
    plt.text(i, v + 2, f"{v:.1f}%", ha='center', fontsize=10)

# 2.3. Average Time to KYC (Bottom Left)
plt.subplot(2, 2, 3)
plt.bar(labels, time_to_kyc, color="#2196f3")
plt.title("Avg Time to Complete KYC")
plt.ylabel("Time (Days)")
plt.ylim(0, max(time_to_kyc) + 1)
for i, v in enumerate(time_to_kyc):
    plt.text(i, v + 0.1, f"{v:.1f}", ha='center', fontsize=10)

# 2.4. Time Between KYC → Link (Bottom Right)
plt.subplot(2, 2, 4)
plt.bar(labels, time_kyc_to_link, color="#ffc107")
plt.title("Avg Time to Complete Link Account")
plt.ylabel("Time (Days)")
plt.ylim(0, max(time_kyc_to_link) + 1)
for i, v in enumerate(time_kyc_to_link):
    plt.text(i, v + 0.1, f"{v:.1f}", ha='center', fontsize=10)

# 3. Final layout settings
plt.tight_layout()
plt.show()
```









🔷 Step 4.1: Segment Revenue Analysis

In [12]:
```
# ✅ Step 4.1: Segment Revenue Analysis

# 1. Filter users who actually completed a transaction and have transaction amount
txn_users = merged_df[merged_df['Txn_Amount'].notnull()]
```

```
# 2. Group by the clustered user segment
segment_revenue = txn_users.groupby('User_Segment').agg({
    'User_ID': 'count',           # How many users completed a transaction
    'Txn_Amount': ['sum', 'mean']  # Total and average transaction amount
})

# 3. Clean column names for easy reading
segment_revenue.columns = ['Txn_User_Count', 'Total_Revenue', 'Avg_Revenue_per_User']
segment_revenue = segment_revenue.reset_index()

# 4. Round values for presentation
segment_revenue = segment_revenue.round(2)

# 5. Preview the segment revenue breakdown
segment_revenue
```

Out[12]:

	User_Segment	Txn_User_Count	Total_Revenue	Avg_Revenue_per_User
0	1	367	199401.13	543.33

Segment	Txn Users	Total Revenue	Avg/User
1	367	₹199,401.00	₹543.33
0 & 2	0	₹0.00	₹0.00

🧠 **Insight:**

Only Segment 1 generates revenue. That alone validates the clusters. It's critical to retain and replicate this user group. This also proves that conversion equals value, not just clicks.

✅ Strategic Message:

Don't optimize for more signups — optimize for Segment 1 behaviors Revenue can be traced to a very specific journey signature

📊 Step 4.2: Simulated Revenue Uplift

> What happens if we recover 20% of Segment 0 users?

In [15]:
```
# Rebuild segment_revenue safely
txn_users = merged_df[merged_df['Txn_Amount'].notnull()]

segment_revenue = txn_users.groupby('User_Segment').agg({
    'User_ID': 'count',
    'Txn_Amount': ['sum', 'mean']
})
segment_revenue.columns = ['Txn_User_Count', 'Total_Revenue', 'Avg_Revenue_per_User']
segment_revenue = segment_revenue.reset_index().round(2)
```

In [17]:
```
# Step 1: Find the converting segment (conversion_rate > 0)
converting_segments = segment_revenue[segment_revenue['Txn_User_Count'] > 0]

# Step 2: If such a segment exists, use its average revenue
```

```
if not converting_segments.empty:
    # We'll just use the first converting segment found (usually there's only one)
    avg_value = converting_segments['Avg_Revenue_per_User'].values[0]
else:
    avg_value = 0

# Step 3: Pull users in Segment 2 and simulate 20% conversion
segment2_users = merged_df[merged_df['User_Segment'] == 2]
simulated_converted_users = int(len(segment2_users) * 0.20)
estimated_uplift = simulated_converted_users * avg_value

# Step 4: Print results
print("📊 Simulated Revenue Uplift if 20% of Segment 2 Converts")
print("-----------------------------------------------------------")
print(f"Users recovered from Segment 2: {simulated_converted_users}")
print(f"Avg revenue per user (converted segment): ₹{avg_value:.2f}")
print(f"📈 Estimated total uplift: ₹{estimated_uplift:.2f}")
```

```
📊 Simulated Revenue Uplift if 20% of Segment 2 Converts
-----------------------------------------------------------
Users recovered from Segment 2: 37
Avg revenue per user (converted segment): ₹543.33
📈 Estimated total uplift: ₹20103.21
```

- A simple what-if shows that recovering 20% of Segment 2 could yield ₹20K+.

  > If 20% of Segment 2 converts using Segment 1's avg spend:

  - Simulated Users: 37
  - Avg Value: ₹543.33
  - **Estimated Uplift: ₹20,103.21**

🧠 **Insight:** Small recovery of lost users can deliver high returns — a clear case for investing in UX fixes at the end of the funnel.

📌 Final Takeaways

Focus Area	Recommendation
Segment 1 (Happy Path)	Learn & scale
Segment 0 (Late Dropouts)	UX improvement, trust nudges
Segment 2 (Early Dropouts)	Improve onboarding or ignore for now

---

✅ Summary

This project demonstrates:

- Behavioral funnel analytics
- Multi-source data integration

- User segmentation with KMeans
- Strategic revenue simulation
- Data storytelling with clean visuals

> 🎯 **Real-world impact:** Identify where money is lost, who's dropping off, and where to optimize first — backed by both behavioral and financial evidence.

In [ ]: