```
from google.colab import files

uploaded = files.upload()  # This will open a file selection box
```

Choose Files 2 files

- **merged_mooc_data.csv**(text/csv) - 49858514 bytes, last modified: 3/20/2025 - 100% done
- **Social Network Analysis.ipynb**(n/a) - 1689386 bytes, last modified: 3/21/2025 - 100% done

```
Saving merged_mooc_data.csv to merged_mooc_data (1).csv
Saving Social Network Analysis.ipynb to Social Network Analysis (1).ipynb
```

```
import pandas as pd  # For data manipulation
import numpy as np  # For numerical computations
import networkx as nx  # For network analysis
import matplotlib.pyplot as plt  # For visualization
import seaborn as sns  # For advanced visualizations
```

```
#Current Working Directory (CWD)

import os
print(os.getcwd())
```

```
/content
```

```
import pandas as pd

# Check if file exists
import os
print("Files in Directory:", os.listdir("/content/"))  # List files

# Load the dataset
file_path = "/content/merged_mooc_data.csv"
if os.path.exists(file_path):
    df = pd.read_csv(file_path)
    print("File Loaded Successfully!")
else:
    print("File Not Found! Upload it again.")
```

```
Files in Directory: ['.config', 'Social Network Analysis.ipynb', 'merged_mooc_data.csv', 'sample_data']
File Loaded Successfully!
```

```
df.duplicated().sum()
```

```
np.int64(15031)
```

```
df_actions = df[["ACTIONID", "USERID", "TARGETID", "TIMESTAMP"]]  # Adjust columns if needed
df_actions.duplicated().sum()
```

```
np.int64(15110)
```

```
import pandas as pd

# Load the dataset
df = pd.read_csv("/content/merged_mooc_data.csv")

# Confirm it's loaded
print("Dataset Loaded Successfully!")
df.head()  # Show first few rows
```

```
Dataset Loaded Successfully!
```

	ACTIONID	USERID	TARGETID	TIMESTAMP	FEATURE0	FEATURE1	FEATURE2	FEATURE3	LABEL
0	1	0	1	1970-01-01 00:00:06	-0.319991	-0.435701	0.106784	-0.067309	0.0
1	2	0	2	1970-01-01 00:00:41	-0.319991	-0.435701	0.106784	-0.067309	0.0
2	3	0	1	1970-01-01 00:00:49	-0.319991	-0.435701	0.106784	-0.067309	0.0
3	4	0	2	1970-01-01 00:00:51	-0.319991	-0.435701	0.106784	-0.067309	0.0
4	5	0	3	1970-01-01 00:00:55	-0.319991	-0.435701	0.106784	-0.067309	0.0

```
# Check the first few rows
df.head()

# Get basic information about the dataset
df.info()

# Check for missing values
df.isnull().sum()

# Get summary statistics
df.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 426771 entries, 0 to 426770
Data columns (total 9 columns):
 #   Column     Non-Null Count   Dtype
---  ------     --------------   -----
 0   ACTIONID   426771 non-null  int64
 1   USERID     426771 non-null  int64
 2   TARGETID   426771 non-null  int64
 3   TIMESTAMP  426771 non-null  object
 4   FEATURE0   426771 non-null  float64
 5   FEATURE1   426771 non-null  float64
 6   FEATURE2   426771 non-null  float64
 7   FEATURE3   426771 non-null  float64
 8   LABEL      426771 non-null  float64
dtypes: float64(5), int64(3), object(1)
memory usage: 29.3+ MB
```

	ACTIONID	USERID	TARGETID	FEATURE0	FEATURE1	FEATURE2	FEATURE3	LABEL
count	426771.000000	426771.000000	426771.000000	426771.000000	426771.000000	426771.000000	426771.000000	426771.000000
mean	205263.889688	3045.055622	26.733086	-0.011115	0.066487	-0.011873	-0.002069	-0.025890
std	118860.572847	1978.397084	21.095592	0.984349	1.050852	0.986305	0.983732	0.210403
min	1.000000	0.000000	0.000000	-0.319991	-0.435701	-0.394237	-0.067309	-1.000000
25%	102392.500000	1278.000000	9.000000	-0.319991	-0.435701	-0.394237	-0.067309	0.000000
50%	204916.000000	2846.000000	22.000000	-0.319991	-0.435701	0.106784	-0.067309	0.000000
75%	308091.500000	4716.000000	39.000000	-0.319991	-0.435701	0.106784	-0.067309	0.000000
max	411748.000000	7046.000000	96.000000	57.647547	98.796794	276.169444	183.970924	1.000000

```
#Duplicate rows

df_actions.duplicated().sum()
```

```
np.int64(15110)
```

```
df["TIMESTAMP"] = pd.to_numeric(df["TIMESTAMP"], errors="coerce")
```

```
#Checking for Invalid User Interactions

df[df["USERID"] == df["TARGETID"]]
```

ACTIONID	USERID	TARGETID	TIMESTAMP	FEATURE0	FEATURE1	FEATURE2	FEATURE3	LABEL

```
#Check if ACTIONID repeats across different users

df.groupby("ACTIONID")["USERID"].nunique().sort_values(ascending=False).head(10)
```

	USERID
ACTIONID	
411748	1
1	1
2	1
3	1
4	1
5	1
6	1
7	1
8	1
9	1

dtype: int64

```
#Determine If Self Interaction are Errors

df_self_interactions = df[df["USERID"] == df["TARGETID"]]
print("Total self-interactions:", df_self_interactions.shape[0])
print(df_self_interactions["TIMESTAMP"].describe())
```

```
Total self-interactions: 0
count    0.0
mean     NaN
std      NaN
min      NaN
25%      NaN
50%      NaN
75%      NaN
max      NaN
Name: TIMESTAMP, dtype: float64
```

```
#Remove Self Interaction (Likely Error)

df = df[df["USERID"] != df["TARGETID"]]
```

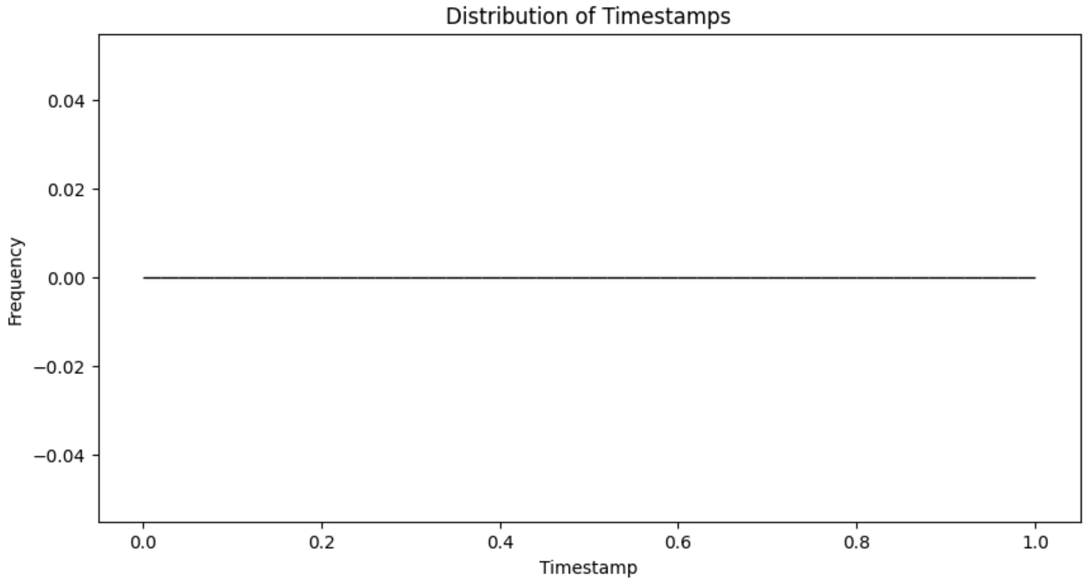
```
#Check Remaining Self Interactions

df[df["USERID"] == df["TARGETID"]]
```

ACTIONID	USERID	TARGETID	TIMESTAMP	FEATURE0	FEATURE1	FEATURE2	FEATURE3	LABEL

```
import pandas as pd

# Load your dataset (update with the correct file path)
df_actions = pd.read_csv("/content/merged_mooc_data.csv")

# Convert TIMESTAMP to numeric (in case of type issues)
df_actions["TIMESTAMP"] = pd.to_numeric(df_actions["TIMESTAMP"], errors="coerce")
```

```
#Checking for Timestamp Outliers

import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(10, 5))
sns.histplot(df_actions["TIMESTAMP"], bins=50, kde=True)
plt.xlabel("Timestamp")
plt.ylabel("Frequency")
plt.title("Distribution of Timestamps")
plt.show()
```

```
#No Extreme Outliers
```




```
#Convert timestamp to datetime format

# Convert TIMESTAMP assuming it's in seconds since Unix epoch
df_actions["TIMESTAMP"] = pd.to_datetime(df_actions["TIMESTAMP"], unit="s")

# Display first few rows to confirm the conversion
df_actions.head()
```

	ACTIONID	USERID	TARGETID	TIMESTAMP	FEATURE0	FEATURE1	FEATURE2	FEATURE3	LABEL
0	1	0	1	NaT	-0.319991	-0.435701	0.106784	-0.067309	0.0
1	2	0	2	NaT	-0.319991	-0.435701	0.106784	-0.067309	0.0
2	3	0	1	NaT	-0.319991	-0.435701	0.106784	-0.067309	0.0
3	4	0	2	NaT	-0.319991	-0.435701	0.106784	-0.067309	0.0
4	5	0	3	NaT	-0.319991	-0.435701	0.106784	-0.067309	0.0

```
#Checking the Timestamp Format

df_actions["TIMESTAMP"].dtype
```

```
dtype('O')
```

```
#Save as csv

df_actions.to_csv("processed_data.csv", index=False)
```

```
#Check for Missing Values

df_actions.isnull().sum()
```

	0
ACTIONID	0
USERID	0
TARGETID	0
TIMESTAMP	426771
FEATURE0	0
FEATURE1	0
FEATURE2	0
FEATURE3	0
LABEL	0

dtype: int64

```
#Check Datatypes

df_actions.dtypes
```

	0
ACTIONID	int64
USERID	int64
TARGETID	int64
TIMESTAMP	datetime64[ns]
FEATURE0	float64
FEATURE1	float64
FEATURE2	float64
FEATURE3	float64
LABEL	float64

dtype: object

```
#Check for Negative Values (UserID & TargetID)

(df_actions[['USERID', 'TARGETID']] < 0).sum()
```

	0
USERID	0
TARGETID	0

dtype: int64

```
df_actions["TIMESTAMP"].describe()
```

	TIMESTAMP
count	0
mean	NaT
min	NaT
25%	NaT
50%	NaT
75%	NaT
max	NaT

dtype: object

```
df_actions["TIMESTAMP"] = pd.to_datetime(df_actions["TIMESTAMP"], unit="ms")
```

```
# Example: Extracting feature columns from an existing DataFrame

df_features = df_actions[["FEATURE0", "FEATURE1", "FEATURE2", "FEATURE3"]]
```

```
#Create Merged File

df_merged = pd.read_csv("/content/merged_mooc_data.csv")
```

```
#Distribution of Missing "Labels"

df_merged[df_merged["LABEL"].isna()].describe()
```

	ACTIONID	USERID	TARGETID	FEATURE0	FEATURE1	FEATURE2	FEATURE3	LABEL
count	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
mean	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
std	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
min	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
25%	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
50%	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
75%	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
max	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

```
df_merged.describe()
```

	ACTIONID	USERID	TARGETID	FEATURE0	FEATURE1	FEATURE2	FEATURE3	LABEL
count	426771.000000	426771.000000	426771.000000	426771.000000	426771.000000	426771.000000	426771.000000	426771.000000
mean	205263.889688	3045.055622	26.733086	-0.011115	0.066487	-0.011873	-0.002069	-0.025890
std	118860.572847	1978.397084	21.095592	0.984349	1.050852	0.986305	0.983732	0.210403
min	1.000000	0.000000	0.000000	-0.319991	-0.435701	-0.394237	-0.067309	-1.000000
25%	102392.500000	1278.000000	9.000000	-0.319991	-0.435701	-0.394237	-0.067309	0.000000
50%	204916.000000	2846.000000	22.000000	-0.319991	-0.435701	0.106784	-0.067309	0.000000
75%	308091.500000	4716.000000	39.000000	-0.319991	-0.435701	0.106784	-0.067309	0.000000
max	411748.000000	7046.000000	96.000000	57.647547	98.796794	276.169444	183.970924	1.000000

```
#Save Merged Cell as csv

df_merged.to_csv("merged_mooc_data.csv", index=False)
```

```
#TIME STAMP EXPLORATORY ANALYSIS (Merged)

#Understanding The Time Range
df_merged["TIMESTAMP"].min(), df_merged["TIMESTAMP"].max()
```

```
('1970-01-01 00:00:06', '1970-01-30 18:28:06')
```

```
df_merged["TIMESTAMP"] = pd.to_datetime(df_merged["TIMESTAMP"])
```

```
#Check Current Working Directory

import os
print(os.getcwd())
```

```
/content
```

```
 # Listing all Files in The Current Directory
import os
print(os.listdir())
```

```
['.config', 'Social Network Analysis.ipynb', 'merged_mooc_data.csv', 'processed_data.csv', 'sample_data']
```

```
#Reloading Merged Data

df_merged = pd.read_csv("merged_mooc_data.csv")
df_merged.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 426771 entries, 0 to 426770
Data columns (total 9 columns):
 #   Column     Non-Null Count   Dtype
---  ------     --------------   -----
 0   ACTIONID   426771 non-null  int64
 1   USERID     426771 non-null  int64
 2   TARGETID   426771 non-null  int64
 3   TIMESTAMP  426771 non-null  object
 4   FEATURE0   426771 non-null  float64
 5   FEATURE1   426771 non-null  float64
 6   FEATURE2   426771 non-null  float64
 7   FEATURE3   426771 non-null  float64
 8   LABEL      426771 non-null  float64
dtypes: float64(5), int64(3), object(1)
memory usage: 29.3+ MB
```

```
#Check Data Type of Timestamp

print(df_merged["TIMESTAMP"].head())  # View first few values
print(df_merged["TIMESTAMP"].dtype)   # Check the data type
```

```
0    1970-01-01 00:00:06
1    1970-01-01 00:00:41
2    1970-01-01 00:00:49
3    1970-01-01 00:00:51
4    1970-01-01 00:00:55
Name: TIMESTAMP, dtype: object
object
```

```
#Convert Strings to Datetime

df_merged["TIMESTAMP"] = pd.to_datetime(df_merged["TIMESTAMP"])
```

```
#Verify Conversion

print(df_merged["TIMESTAMP"].dtype)  # Should output: datetime64[ns]
```

```
datetime64[ns]
```

```
# Extract year, month, day, hour, and day of the week (Time Features)

df_merged["YEAR"] = df_merged["TIMESTAMP"].dt.year
df_merged["MONTH"] = df_merged["TIMESTAMP"].dt.month
df_merged["DAY"] = df_merged["TIMESTAMP"].dt.day
df_merged["HOUR"] = df_merged["TIMESTAMP"].dt.hour
df_merged["DAYOFWEEK"] = df_merged["TIMESTAMP"].dt.day_name()  # Gives Monday, Tuesday, etc.

# Display first few rows to confirm
df_merged.head()
```

	ACTIONID	USERID	TARGETID	TIMESTAMP	FEATURE0	FEATURE1	FEATURE2	FEATURE3	LABEL	YEAR	MONTH	DAY	HOUR	DAYOFWEEK
0	1	0	1	1970-01-01 00:00:06	-0.319991	-0.435701	0.106784	-0.067309	0.0	1970	1	1	0	Thursday
1	2	0	2	1970-01-01 00:00:41	-0.319991	-0.435701	0.106784	-0.067309	0.0	1970	1	1	0	Thursday
2	3	0	1	1970-01-01 00:00:49	-0.319991	-0.435701	0.106784	-0.067309	0.0	1970	1	1	0	Thursday
3	4	0	2	1970-01-01 00:00:51	-0.319991	-0.435701	0.106784	-0.067309	0.0	1970	1	1	0	Thursday
4	5	0	3	1970-01-01 00:00:55	-0.319991	-0.435701	0.106784	-0.067309	0.0	1970	1	1	0	Thursday

```
df_merged.head()  # View first few rows after sorting
df_merged.tail()  # View last few rows to confirm sorting
```

	ACTIONID	USERID	TARGETID	TIMESTAMP	FEATURE0	FEATURE1	FEATURE2	FEATURE3	LABEL	YEAR	MONTH	DAY	HOUR	DAYOFWEEK
426766	411744	7026	8	1970-01-30 18:27:21	-0.319991	-0.435701	0.106784	-0.067309	0.0	1970	1	30	18	Friday
426767	411745	6842	8	1970-01-30 18:27:23	-0.319991	-0.435701	0.106784	-0.067309	0.0	1970	1	30	18	Friday
426768	411746	7026	9	1970-01-30 18:27:28	-0.319991	-0.435701	0.106784	-0.067309	0.0	1970	1	30	18	Friday
426769	411747	6842	5	1970-01-30 18:27:34	-0.319991	-0.435701	0.106784	-0.067309	0.0	1970	1	30	18	Friday
426770	411748	70	23	1970-01-30 18:28:06	-0.319991	-0.435701	0.106784	-0.067309	0.0	1970	1	30	18	Friday

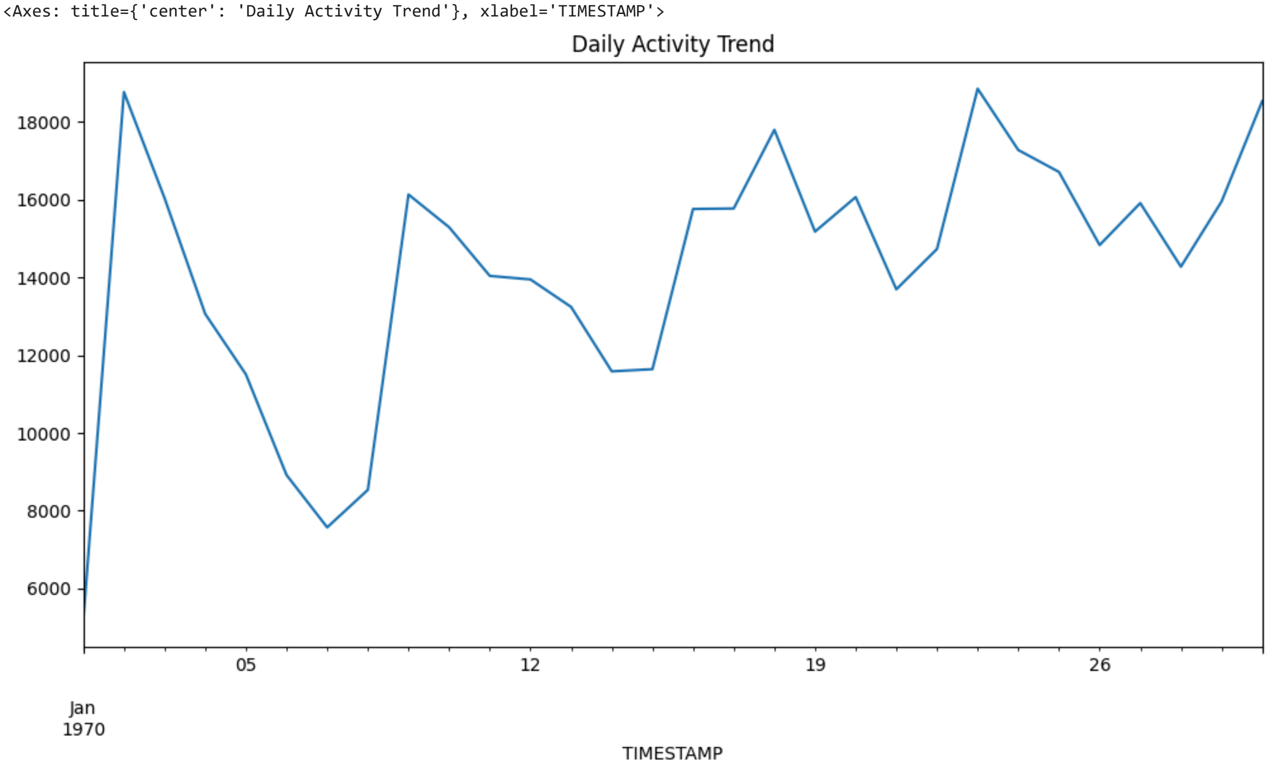
```
df_filtered = df_merged[df_merged["TIMESTAMP"] >= "2020-01-01"]
```

```
print(df_merged["TIMESTAMP"].min())  # Earliest timestamp
print(df_merged["TIMESTAMP"].max())  # Latest timestamp
```

```
1970-01-01 00:00:06
1970-01-30 18:28:06
```

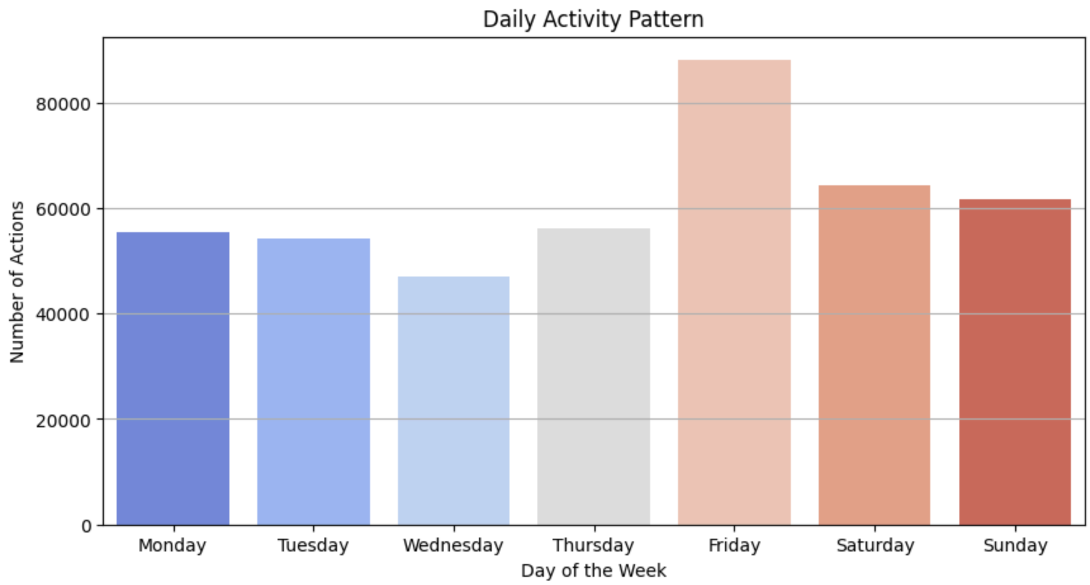
```
#Number of Actions Over Time (Overall Activity Trends) - Line Chart

df_merged.resample("D", on="TIMESTAMP").size().plot(
    kind="line", figsize=(12, 6), title="Daily Activity Trend"
)
```

```
<Axes: title={'center': 'Daily Activity Trend'}, xlabel='TIMESTAMP'>
```

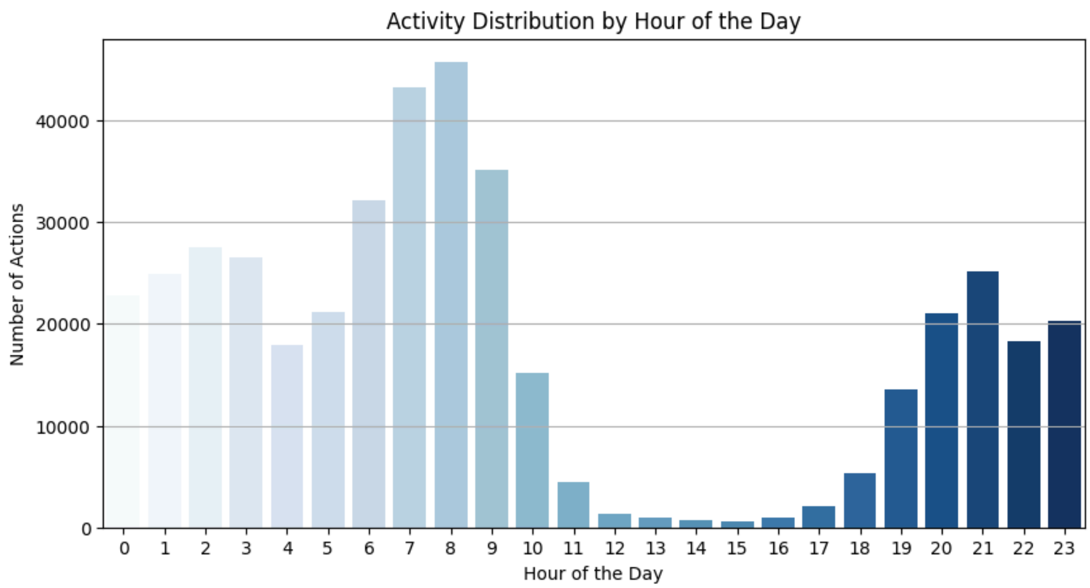



```
#Daily Activity Pattern

#import seaborn as sns
import matplotlib.pyplot as plt

# Define the correct order of days
day_order = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"]

# Count the number of actions per day
day_counts = df_merged["DAYOFWEEK"].value_counts()

# Reindex to match the correct order
```

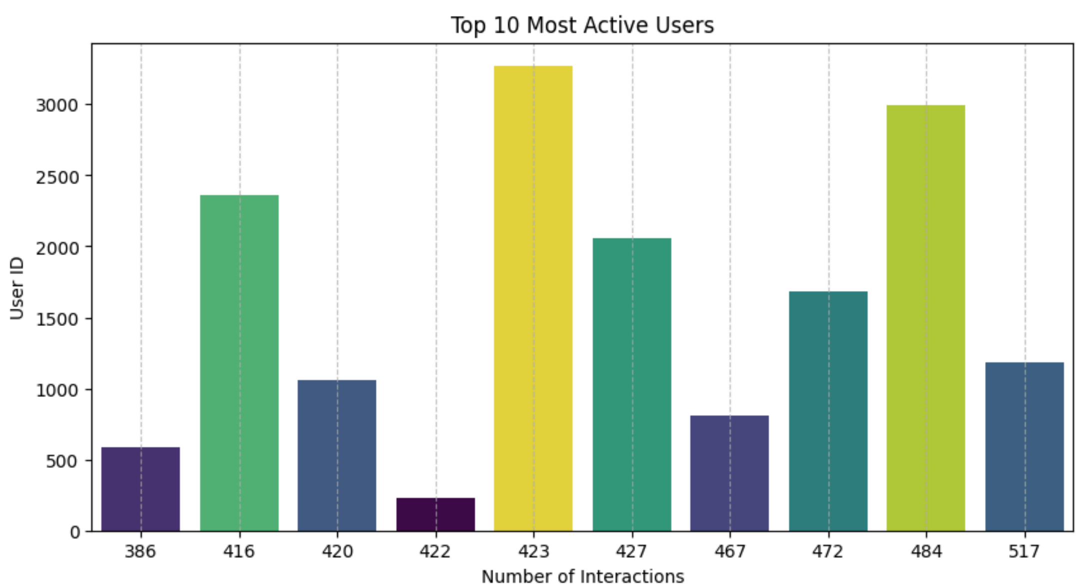
```
day_counts = day_counts.reindex(day_order)

# Plot
plt.figure(figsize=(10, 5))
sns.barplot(x=day_counts.index, y=day_counts.values, hue=day_counts.index, palette="coolwarm", legend=False)
plt.xlabel("Day of the Week")
plt.ylabel("Number of Actions")
plt.title("Daily Activity Pattern")
plt.grid(axis="y")
plt.show()
```




```
#Hourly Activity Pattern (Bar Chart)

import matplotlib.pyplot as plt
import seaborn as sns

# Group by hour and count the number of actions
df_hourly_activity = df_merged.groupby("HOUR").size()

# Plot
plt.figure(figsize=(10, 5))
sns.barplot(x=df_hourly_activity.index, y=df_hourly_activity.values, hue=df_hourly_activity.index, palette="Blues", legend=False)
plt.xlabel("Hour of the Day")
plt.ylabel("Number of Actions")
plt.title("Activity Distribution by Hour of the Day")
plt.xticks(range(0, 24))  # Ensure all hours are shown
plt.grid(axis="y")
plt.show()
```

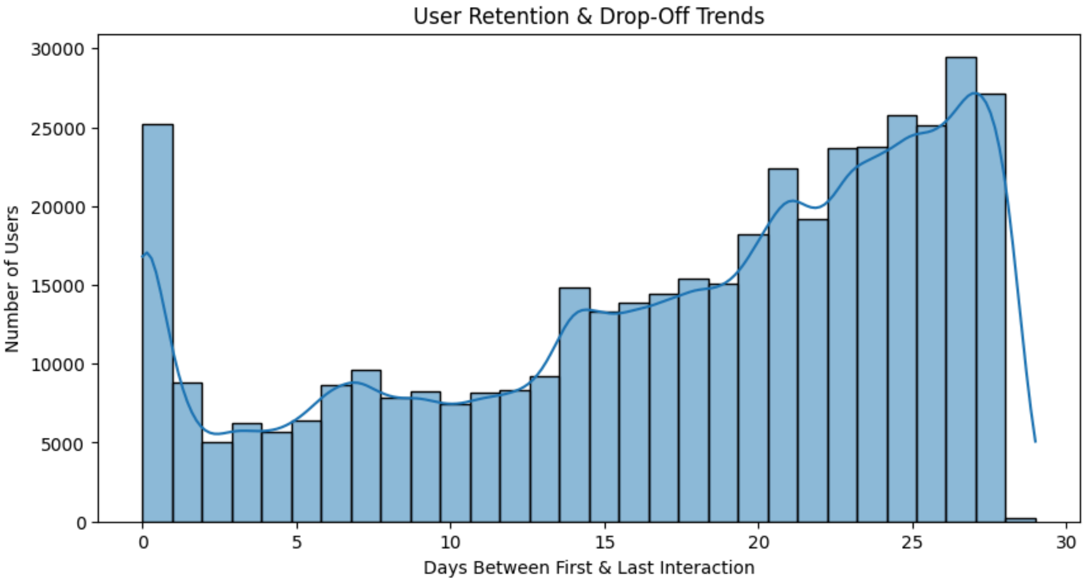



```
import matplotlib.pyplot as plt
import seaborn as sns

# Get Top 10 Most Active Users
top_users = df_merged["USERID"].value_counts().head(10)

# Plot
plt.figure(figsize=(10, 5))
sns.barplot(y=top_users.index, x=top_users.values, hue=top_users.index, palette="viridis", legend=False)
plt.ylabel("User ID")
plt.xlabel("Number of Interactions")
plt.title("Top 10 Most Active Users")
plt.grid(axis="x", linestyle="--", alpha=0.7)  # Add gridlines for better readability
plt.show()
```

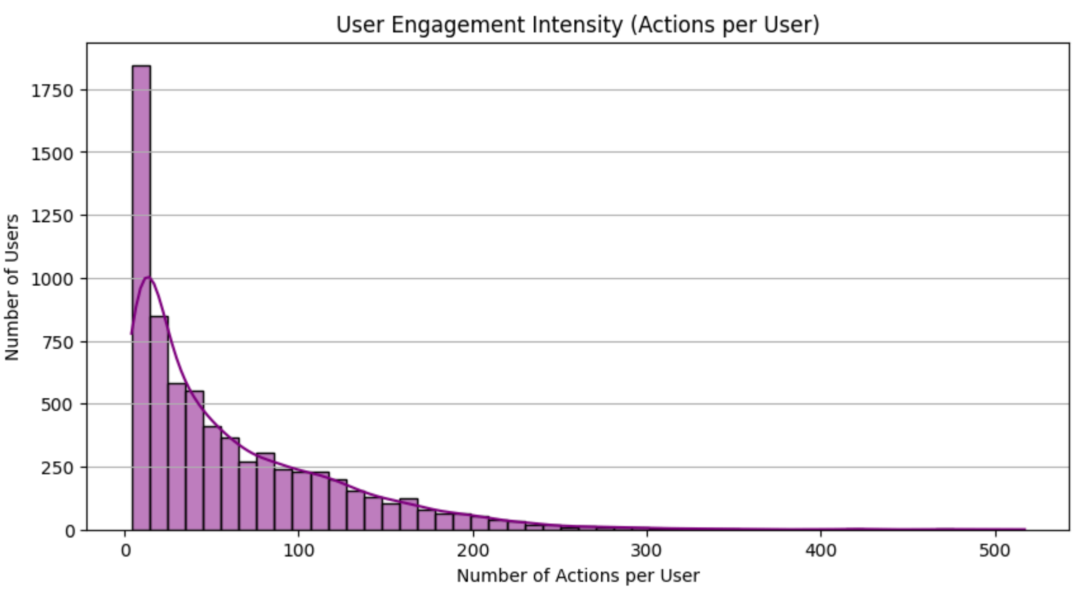



```
#User Retention and Drop Off Trend (Histogram)

df_merged["USER_FIRST_DATE"] = df_merged.groupby("USERID")["TIMESTAMP"].transform("min")
df_merged["USER_LAST_DATE"] = df_merged.groupby("USERID")["TIMESTAMP"].transform("max")

df_merged["USER_LIFESPAN_DAYS"] = (df_merged["USER_LAST_DATE"] - df_merged["USER_FIRST_DATE"]).dt.days

plt.figure(figsize=(10, 5))
sns.histplot(df_merged["USER_LIFESPAN_DAYS"], bins=30, kde=True)
plt.xlabel("Days Between First & Last Interaction")
plt.ylabel("Number of Users")
plt.title("User Retention & Drop-Off Trends")
```

```
plt.show()
```




```
#NUmber of Actions per User

import seaborn as sns

# Calculate the number of actions per user
actions_per_user = df_merged.groupby("USERID").size()

# Plot distribution
plt.figure(figsize=(10, 5))
sns.histplot(actions_per_user, bins=50, kde=True, color="purple")
plt.xlabel("Number of Actions per User")
plt.ylabel("Number of Users")
plt.title("User Engagement Intensity (Actions per User)")
plt.grid(axis="y")
plt.show()
```



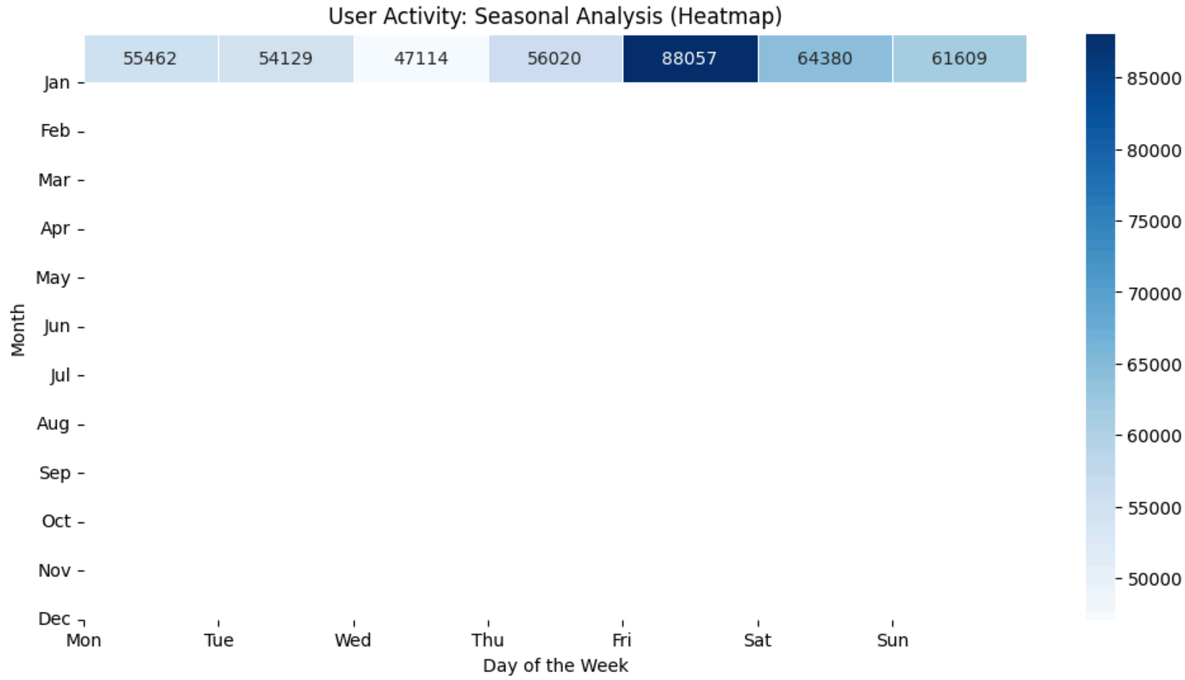

```
#User Activity Overtimw (Line Chart)

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Ensure timestamp column is in datetime format
df_merged["TIMESTAMP"] = pd.to_datetime(df_merged["TIMESTAMP"])
```

```
# Aggregate daily user activity
daily_activity = df_merged.groupby(df_merged["TIMESTAMP"].dt.date).size()

# Plot the time series
plt.figure(figsize=(12, 6))
sns.lineplot(x=daily_activity.index, y=daily_activity.values, marker="o", color="blue")

# Formatting the plot
plt.xlabel("Date")
plt.ylabel("Number of Interactions")
plt.title("User Activity Over Time")
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability
plt.grid(True, linestyle="--", alpha=0.6)

plt.show()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-21-9deae44eccef> in <cell line: 0>()
      6
      7 # Ensure timestamp column is in datetime format
----> 8 df_merged["TIMESTAMP"] = pd.to_datetime(df_merged["TIMESTAMP"])
      9
     10 # Aggregate daily user activity

NameError: name 'df_merged' is not defined
```

Next steps: Explain error

```
#Seasonal Analysis (Heatmap)

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Standardize column names to lowercase
df_merged.columns = df_merged.columns.str.lower()

# Identify the timestamp column dynamically
possible_cols = ["timestamp", "date", "created_at", "datetime"]
timestamp_col = next((col for col in possible_cols if col in df_merged.columns), None)

if timestamp_col is None:
    raise KeyError("No timestamp column found. Please check your dataframe!")

# Convert the timestamp column to datetime format
df_merged[timestamp_col] = pd.to_datetime(df_merged[timestamp_col])

# Extract month and day of the week
df_merged["month"] = df_merged[timestamp_col].dt.month
df_merged["day_of_week"] = df_merged[timestamp_col].dt.dayofweek  # 0=Monday, 6=Sunday

# Aggregate user interactions by Month & Day of the Week
seasonal_activity = df_merged.groupby(["month", "day_of_week"]).size().unstack()

# Create a heatmap
plt.figure(figsize=(12, 6))
sns.heatmap(seasonal_activity, cmap="Blues", annot=True, fmt="d", linewidths=0.5)

# Formatting the heatmap
plt.xlabel("Day of the Week")
plt.ylabel("Month")
plt.title("User Activity: Seasonal Analysis (Heatmap)")

# Replace numeric labels with actual day names
plt.xticks(ticks=range(7), labels=["Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"])
plt.yticks(ticks=range(1, 13), labels=[
    "Jan", "Feb", "Mar", "Apr", "May", "Jun",
    "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"
], rotation=0)

plt.show()
```




```
#Verify if TARGETID refers to other users or resources

import pandas as pd

# Load your dataset
df = pd.read_csv("merged_mooc_data.csv")

# Check if all TARGETID values exist in USERID
unique_users = set(df["USERID"].unique())
unique_targets = set(df["TARGETID"].unique())

# Find TARGETID values that are not in USERID
non_user_targets = unique_targets - unique_users

# Display results
if len(non_user_targets) == 0:
    print("TARGETID represents other users (User-User Interaction Network).")
else:
    print(f"TARGETID represents external resources (Bipartite Network).")
    print(f"Number of TARGETIDs that are NOT in USERID: {len(non_user_targets)}")

# Additional insights
print(f"Unique USERIDs: {len(unique_users)}")
print(f"Unique TARGETIDs: {len(unique_targets)}")
```

```
TARGETID represents other users (User-User Interaction Network).
Unique USERIDs: 7047
Unique TARGETIDs: 97
```

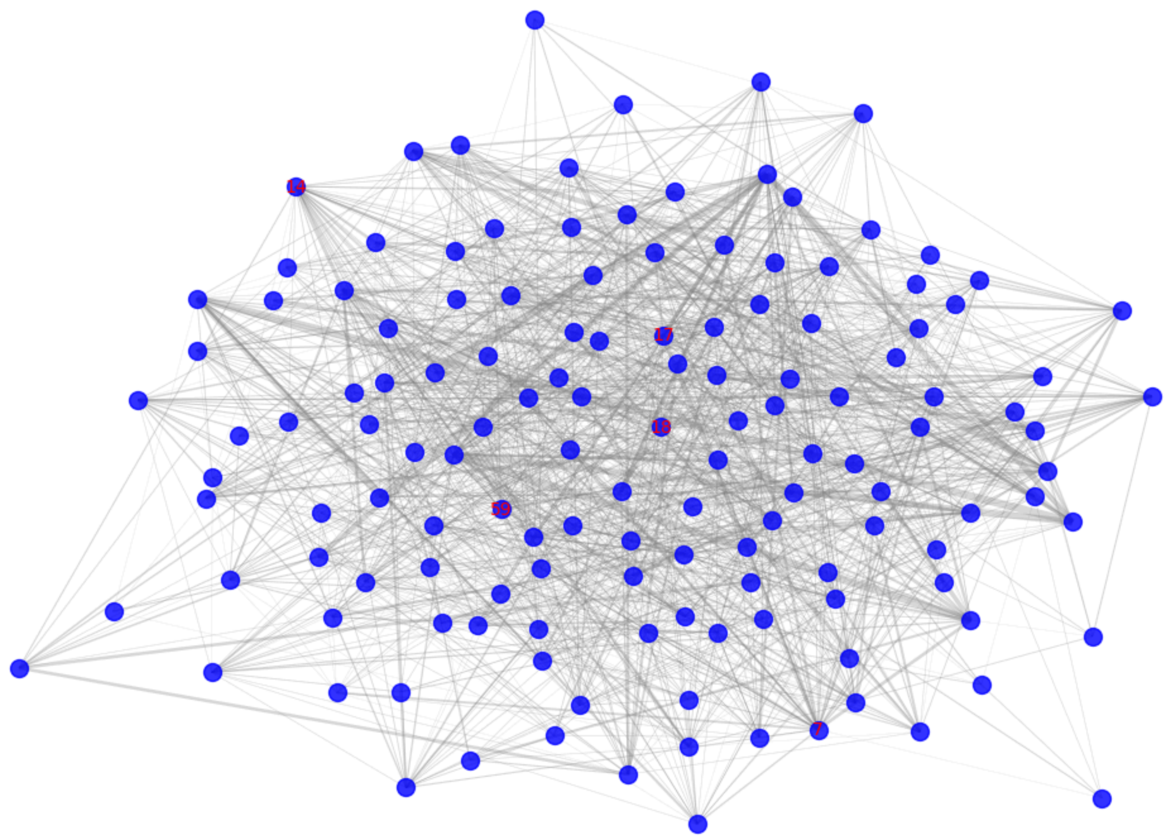
```
import pandas as pd

# Load your dataset (Make sure the file path is correct)
df = pd.read_csv("merged_mooc_data.csv")

# Verify it's loaded
print(df.head())  # Show first few rows
print(df.info())  # Check if data is properly loaded
```

```
   ACTIONID  USERID  TARGETID            TIMESTAMP  FEATURE0  FEATURE1  \
0         1       0         1  1970-01-01 00:00:06 -0.319991 -0.435701
1         2       0         2  1970-01-01 00:00:41 -0.319991 -0.435701
2         3       0         1  1970-01-01 00:00:49 -0.319991 -0.435701
3         4       0         2  1970-01-01 00:00:51 -0.319991 -0.435701
4         5       0         3  1970-01-01 00:00:55 -0.319991 -0.435701

   FEATURE2  FEATURE3  LABEL
0  0.106784 -0.067309    0.0
1  0.106784 -0.067309    0.0
2  0.106784 -0.067309    0.0
```

```
3  0.106784 -0.067309    0.0
4  0.106784 -0.067309    0.0
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 426771 entries, 0 to 426770
Data columns (total 9 columns):
 #   Column     Non-Null Count   Dtype  
---  ------     --------------   -----  
 0   ACTIONID   426771 non-null  int64  
 1   USERID     426771 non-null  int64  
 2   TARGETID   426771 non-null  int64  
 3   TIMESTAMP  426771 non-null  object 
 4   FEATURE0   426771 non-null  float64
 5   FEATURE1   426771 non-null  float64
 6   FEATURE2   426771 non-null  float64
 7   FEATURE3   426771 non-null  float64
 8   LABEL      426771 non-null  float64
dtypes: float64(5), int64(3), object(1)
memory usage: 29.3+ MB
None
```

```python
import pandas as pd
import networkx as nx
import matplotlib.pyplot as plt

# Load dataset (Assuming dataset is already loaded in df)
# df = pd.read_csv("merged_mooc_data.csv")

# Remove missing values
df = df.dropna(subset=["USERID", "TARGETID"])

# Compute edge weights (interaction frequency)
edge_weights = df.groupby(["USERID", "TARGETID"]).size().reset_index(name="weight")

# Sample a subset of edges for visualization (this does NOT affect later analysis)
edge_weights_sample = edge_weights.sample(frac=0.2, random_state=42)  # Keep 20% of edges

# Create weighted graph
G = nx.Graph()

# Add weighted edges from the sampled dataset
for _, row in edge_weights_sample.iterrows():
    G.add_edge(row["USERID"], row["TARGETID"], weight=row["weight"])

# **Filter to remove low-degree nodes**
degree_threshold = 20  # Adjust as needed
filtered_nodes = [n for n, d in dict(G.degree()).items() if d >= degree_threshold]
G_filtered = G.subgraph(filtered_nodes)

# **Extract edge weights**
weights = [G_filtered[u][v]["weight"] for u, v in G_filtered.edges()]

# **Identify top influencers (highest degree) for labeling**
top_nodes = sorted(G_filtered.degree, key=lambda x: x[1], reverse=True)[:5]  # Top 5 nodes
top_nodes_labels = {node: str(node) for node, _ in top_nodes}

# **Plot weighted network with optimized performance**
plt.figure(figsize=(14, 10))

# Use a faster layout algorithm
pos = nx.kamada_kawai_layout(G_filtered)  # Alternative: spring_layout with lower iterations

# Draw edges with varying thickness and transparency
nx.draw_networkx_edges(
    G_filtered, pos, alpha=0.3, edge_color="gray",
    width=[w / max(weights) * 5 for w in weights]
)

# Draw nodes
nx.draw_networkx_nodes(G_filtered, pos, node_size=100, node_color="blue", alpha=0.8)

# Draw labels only for top influencers
nx.draw_networkx_labels(G_filtered, pos, labels=top_nodes_labels, font_size=10, font_color="red")

# Add a title
plt.title("Optimized Weighted User Interaction Network", fontsize=14)
plt.axis("off")  # Hide axes
plt.show()
```

Optimized Weighted User Interaction Network



```
import os; print(os.getcwd())
```

```
/content
```

```
#Reloading The Dataset

import pandas as pd

# Define the correct path to your dataset file
file_path = "merged_mooc_data.csv"

# Load the dataset
df = pd.read_csv(file_path)

# Print first few rows to confirm it loaded correctly
print(df.head())
print(df.info())
```

```
   ACTIONID  USERID  TARGETID           TIMESTAMP  FEATURE0  FEATURE1  \
0         1       0         1 1970-01-01 00:00:06 -0.319991 -0.435701
1         2       0         2 1970-01-01 00:00:41 -0.319991 -0.435701
2         3       0         1 1970-01-01 00:00:49 -0.319991 -0.435701
3         4       0         2 1970-01-01 00:00:51 -0.319991 -0.435701
4         5       0         3 1970-01-01 00:00:55 -0.319991 -0.435701

   FEATURE2  FEATURE3  LABEL
0  0.106784 -0.067309    0.0
1  0.106784 -0.067309    0.0
2  0.106784 -0.067309    0.0
3  0.106784 -0.067309    0.0
4  0.106784 -0.067309    0.0
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 426771 entries, 0 to 426770
Data columns (total 9 columns):
 #   Column     Non-Null Count   Dtype
```

```
 ---  ------     --------------   -----
 0   ACTIONID   426771 non-null  int64
 1   USERID     426771 non-null  int64
 2   TARGETID   426771 non-null  int64
 3   TIMESTAMP  426771 non-null  object
 4   FEATURE0   426771 non-null  float64
 5   FEATURE1   426771 non-null  float64
 6   FEATURE2   426771 non-null  float64
 7   FEATURE3   426771 non-null  float64
 8   LABEL      426771 non-null  float64
dtypes: float64(5), int64(3), object(1)
memory usage: 29.3+ MB
None
```

```
import pandas as pd

# Load dataset (check if this part runs first)
file_path = "merged_mooc_data.csv"
df = pd.read_csv(file_path)

# Confirm data is loaded
print("Dataset Loaded Successfully!")
print(df.head())  # See if this prints
```

```
Dataset Loaded Successfully!
   ACTIONID  USERID  TARGETID            TIMESTAMP  FEATURE0  FEATURE1  \
0         1       0         1  1970-01-01 00:00:06 -0.319991 -0.435701
1         2       0         2  1970-01-01 00:00:41 -0.319991 -0.435701
2         3       0         1  1970-01-01 00:00:49 -0.319991 -0.435701
3         4       0         2  1970-01-01 00:00:51 -0.319991 -0.435701
4         5       0         3  1970-01-01 00:00:55 -0.319991 -0.435701

   FEATURE2  FEATURE3  LABEL
0  0.106784 -0.067309    0.0
1  0.106784 -0.067309    0.0
2  0.106784 -0.067309    0.0
3  0.106784 -0.067309    0.0
4  0.106784 -0.067309    0.0
```

```
import pandas as pd
print("Pandas Imported Successfully!")
```

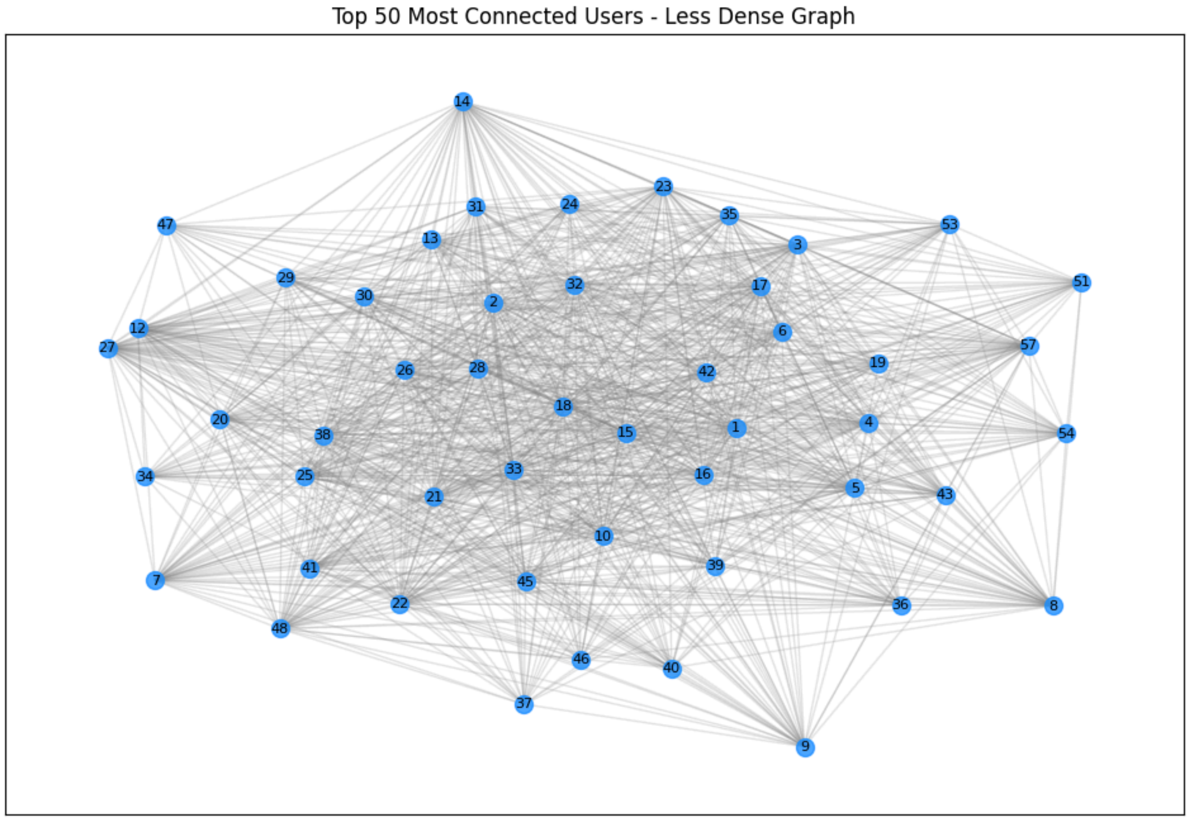
```
Pandas Imported Successfully!
```

```
import os

file_path = "merged_mooc_data.csv"

# Check if the file exists in the current directory
if os.path.exists(file_path):
    print("✅ File Found!")
else:
    print("❌ File NOT Found! Check the file path.")
```

```
✅ File Found!
```

```
#Visuals for Nodes & Edges Graph

import networkx as nx
import matplotlib.pyplot as plt

# Reduce to 50 most connected nodes (less dense)
top_nodes = sorted(G.degree, key=lambda x: x[1], reverse=True)[:50]
G_sub = G.subgraph([node for node, _ in top_nodes])

# Use Kamada-Kawai layout with increased scale for better spacing
pos = nx.kamada_kawai_layout(G_sub, scale=5)  # Increase scale to spread nodes

# Plot
plt.figure(figsize=(12, 8))
nx.draw_networkx_edges(G_sub, pos, alpha=0.2, edge_color="gray")  # Make edges lighter
nx.draw_networkx_nodes(G_sub, pos, node_size=100, node_color="dodgerblue", alpha=0.8)  # Adjust node size & color
nx.draw_networkx_labels(G_sub, pos, font_size=8, font_color="black")  # Smaller labels for readability

# Title
plt.title("Top 50 Most Connected Users - Less Dense Graph", fontsize=12)

# Show the final plot
```

```
plt.show()
```




```
print(f"Number of Nodes: {G.number_of_nodes()}")
print(f"Number of Edges: {G.number_of_edges()}")
```

```
Number of Nodes: 7047
Number of Edges: 177891
```

```
import networkx as nx
import pandas as pd

# Group by USERID and TARGETID to count frequency of interactions
edge_weights = df.groupby(["USERID", "TARGETID"]).size().reset_index(name="weight")

# Create a weighted graph
G_weighted = nx.Graph()

# Add nodes
G_weighted.add_nodes_from(df["USERID"].unique())

# Add edges with weights
for _, row in edge_weights.iterrows():
    G_weighted.add_edge(row["USERID"], row["TARGETID"], weight=row["weight"])

print(f"Number of nodes: {G_weighted.number_of_nodes()}")
print(f"Number of edges: {G_weighted.number_of_edges()}")
```

```
Number of nodes: 7047
Number of edges: 177891
```

```
# Assign Timestamps to Edges

df[['TIMESTAMP']].head()
```

	TIMESTAMP
0	1970-01-01 00:00:06
1	1970-01-01 00:00:41
2	1970-01-01 00:00:49
3	1970-01-01 00:00:51
4	1970-01-01 00:00:55

```
#Convert TIMESTAMP to Datetime Format

df['TIMESTAMP'] = pd.to_datetime(df['TIMESTAMP'])
```

```
#Extract Date, Hour & Week Number

df['DATE'] = df['TIMESTAMP'].dt.date   # Extract date (YYYY-MM-DD)
df['HOUR'] = df['TIMESTAMP'].dt.hour   # Extract hour of the day
df['WEEK'] = df['TIMESTAMP'].dt.isocalendar().week  # Extract week number
```

```
df[['TIMESTAMP', 'DATE', 'HOUR', 'WEEK']].head()
```

```
print(df.columns)
```

```
Index(['ACTIONID', 'USERID', 'TARGETID', 'TIMESTAMP', 'FEATURE0', 'FEATURE1',
       'FEATURE2', 'FEATURE3', 'LABEL', 'DATE', 'HOUR', 'WEEK'],
      dtype='object')
```

```
import pandas as pd
import networkx as nx
import matplotlib.pyplot as plt

# Load dataset (Ensure it contains USERID, TARGETID, and ACTIONID columns)
# df = pd.read_csv("merged_mooc_data.csv")

# **Check column names first**
print("Dataset Columns:", df.columns)

# **Update column selection to match actual dataset**
required_columns = ["USERID", "TARGETID", "ACTIONID"]  # Replaced ACTION with ACTIONID

# **Drop missing values only from existing columns**
df.dropna(subset=required_columns, inplace=True)

print("Data cleaned successfully! Ready for network analysis.")
```

```
Dataset Columns: Index(['ACTIONID', 'USERID', 'TARGETID', 'TIMESTAMP', 'FEATURE0', 'FEATURE1',
       'FEATURE2', 'FEATURE3', 'LABEL', 'DATE', 'HOUR', 'WEEK'],
      dtype='object')
Data cleaned successfully! Ready for network analysis.
```

```
import os

# List all files in current directory
print(os.listdir())
```

```
['.config', 'sample_data']
```

```
from google.colab import files

uploaded = files.upload()  # Click and upload the file when prompted
df = pd.read_csv(next(iter(uploaded)))  # Load the uploaded file

print(df.head())  # Check if data is loaded
print("✅ Dataset Loaded Successfully!")
```

Choose Files 2 files

- **merged_mooc_data.csv**(text/csv) - 49858514 bytes, last modified: 3/20/2025 - 100% done
- **Social Network Analysis.ipynb**(n/a) - 1689386 bytes, last modified: 3/21/2025 - 100% done

```
Saving merged_mooc_data.csv to merged_mooc_data.csv
Saving Social Network Analysis.ipynb to Social Network Analysis.ipynb
   ACTIONID  USERID  TARGETID           TIMESTAMP  FEATURE0  FEATURE1  \
0         1       0         1  1970-01-01 00:00:06 -0.319991 -0.435701
1         2       0         2  1970-01-01 00:00:41 -0.319991 -0.435701
2         3       0         1  1970-01-01 00:00:49 -0.319991 -0.435701
3         4       0         2  1970-01-01 00:00:51 -0.319991 -0.435701
4         5       0         3  1970-01-01 00:00:55 -0.319991 -0.435701

   FEATURE2  FEATURE3  LABEL
0  0.106784 -0.067309    0.0
1  0.106784 -0.067309    0.0
2  0.106784 -0.067309    0.0
3  0.106784 -0.067309    0.0
4  0.106784 -0.067309    0.0
✅ Dataset Loaded Successfully!
```

```
import pandas as pd

df = pd.read_csv("merged_mooc_data.csv")

print(df.head())  # Confirm that the data is loaded
print("✅ Dataset Loaded Successfully!")
```

```
   ACTIONID  USERID  TARGETID           TIMESTAMP  FEATURE0  FEATURE1  \
0         1       0         1  1970-01-01 00:00:06 -0.319991 -0.435701
1         2       0         2  1970-01-01 00:00:41 -0.319991 -0.435701
2         3       0         1  1970-01-01 00:00:49 -0.319991 -0.435701
3         4       0         2  1970-01-01 00:00:51 -0.319991 -0.435701
4         5       0         3  1970-01-01 00:00:55 -0.319991 -0.435701

   FEATURE2  FEATURE3  LABEL
0  0.106784 -0.067309    0.0
1  0.106784 -0.067309    0.0
2  0.106784 -0.067309    0.0
3  0.106784 -0.067309    0.0
4  0.106784 -0.067309    0.0
✅ Dataset Loaded Successfully!
```

```
import pandas as pd

# Load dataset (Uncomment and specify your actual file if needed)
# df = pd.read_csv("merged_mooc_data.csv")

# If using a Colab-uploaded file:
# from google.colab import files
# uploaded = files.upload()
# df = pd.read_csv(next(iter(uploaded)))

# Print first few rows to confirm
print(df.head())
print("Dataset Loaded Successfully!")
```

```
   ACTIONID  USERID  TARGETID           TIMESTAMP  FEATURE0  FEATURE1  \
0         1       0         1  1970-01-01 00:00:06 -0.319991 -0.435701
1         2       0         2  1970-01-01 00:00:41 -0.319991 -0.435701
2         3       0         1  1970-01-01 00:00:49 -0.319991 -0.435701
3         4       0         2  1970-01-01 00:00:51 -0.319991 -0.435701
4         5       0         3  1970-01-01 00:00:55 -0.319991 -0.435701

   FEATURE2  FEATURE3  LABEL
0  0.106784 -0.067309    0.0
1  0.106784 -0.067309    0.0
2  0.106784 -0.067309    0.0
3  0.106784 -0.067309    0.0
4  0.106784 -0.067309    0.0
✅ Dataset Loaded Successfully!
```

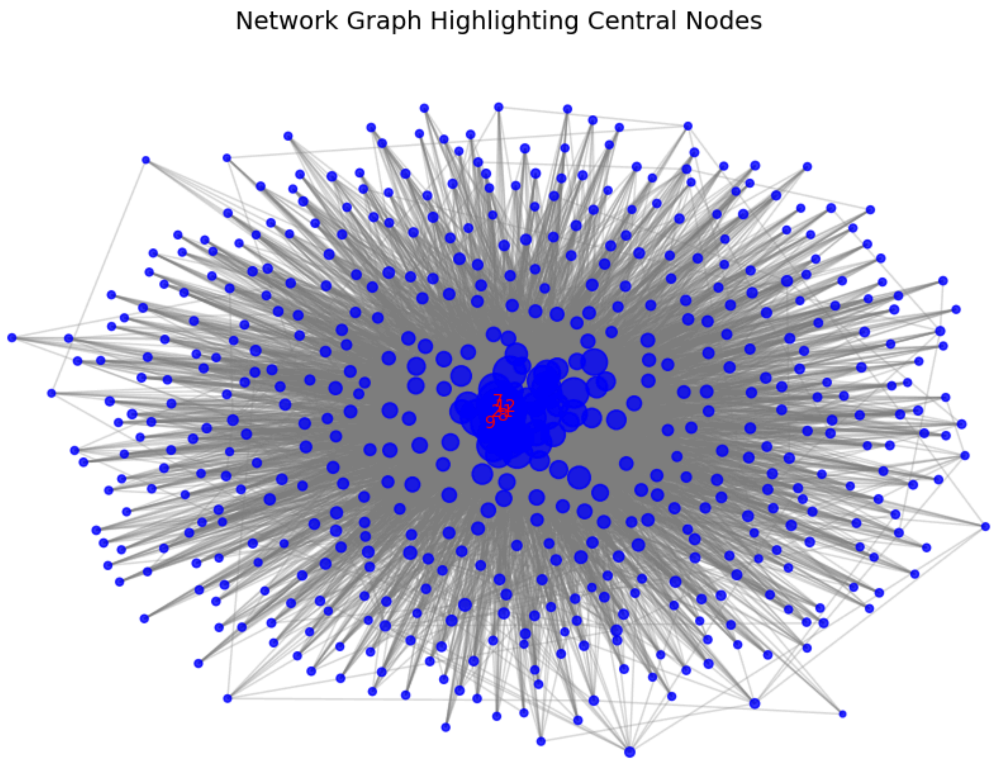
```
import pandas as pd
import networkx as nx
import matplotlib.pyplot as plt

# Load dataset
# df = pd.read_csv("merged_mooc_data.csv")

# Drop missing values
df = df.dropna(subset=["USERID", "TARGETID"])
```

```
# Compute interaction frequency (weight of edges)
edge_weights = df.groupby(["USERID", "TARGETID"]).size().reset_index(name="weight")

# **Filter out low-frequency interactions to reduce clutter**
min_interactions = 5  # Adjust threshold if needed
edge_weights = edge_weights[edge_weights["weight"] >= min_interactions]

# **Create Graph**
G = nx.Graph()

# **Add weighted edges (interactions)**
for _, row in edge_weights.iterrows():
    G.add_edge(row["USERID"], row["TARGETID"], weight=row["weight"])

# **Remove low-degree nodes**
degree_threshold = 10  # Adjust as needed
filtered_nodes = [n for n, d in dict(G.degree()).items() if d >= degree_threshold]
G_filtered = G.subgraph(filtered_nodes)

print("Graph Created Successfully!")
```

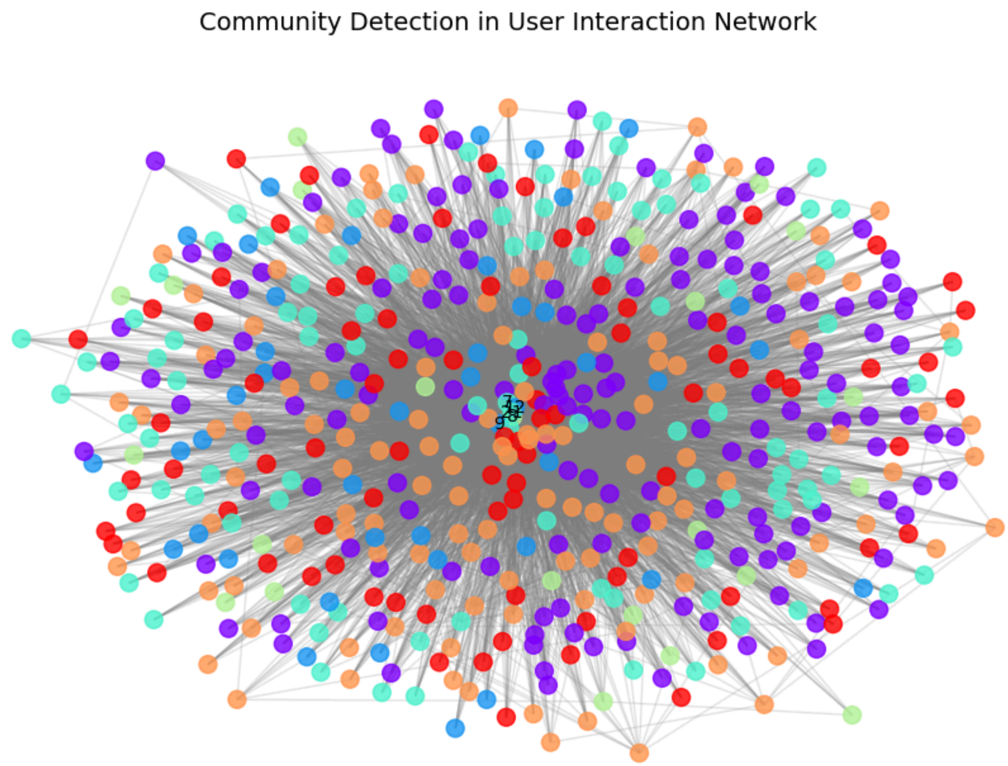
✅ Graph Created Successfully!

```
#Determining the most influential users

#Centrality Analysis

import numpy as np

# **Compute Centrality Measures**
degree_centrality = nx.degree_centrality(G_filtered)
betweenness_centrality = nx.betweenness_centrality(G_filtered)
closeness_centrality = nx.closeness_centrality(G_filtered)

# **Find Top 5 Influencers Based on Centrality**
top_degree = sorted(degree_centrality.items(), key=lambda x: x[1], reverse=True)[:5]
top_betweenness = sorted(betweenness_centrality.items(), key=lambda x: x[1], reverse=True)[:5]
top_closeness = sorted(closeness_centrality.items(), key=lambda x: x[1], reverse=True)[:5]

print("🔷 Top 5 Nodes by Degree Centrality:", top_degree)
print("🔷 Top 5 Nodes by Betweenness Centrality:", top_betweenness)
print("🔷 Top 5 Nodes by Closeness Centrality:", top_closeness)

# **Normalize Values for Visualization**
node_sizes = np.array([degree_centrality[n] for n in G_filtered.nodes()]) * 1000

# **Plot Centrality Visualization**
plt.figure(figsize=(12, 8))
pos = nx.spring_layout(G_filtered, seed=42, k=0.7)

# Draw edges
nx.draw_networkx_edges(G_filtered, pos, alpha=0.3, edge_color="gray")

# Draw nodes with sizes based on centrality
nx.draw_networkx_nodes(G_filtered, pos, node_size=node_sizes, node_color="blue", alpha=0.8)

# Label only the top influencers
top_nodes_labels = {node: str(node) for node, _ in top_degree}
nx.draw_networkx_labels(G_filtered, pos, labels=top_nodes_labels, font_size=10, font_color="red")

plt.title("Network Graph Highlighting Central Nodes", fontsize=14)
plt.axis("off")
plt.show()
```

```
🔷 Top 5 Nodes by Degree Centrality: [(np.int64(21), 0.6945996275605214), (np.int64(8), 0.633147113594041), (np.int64(7), 0.5418994413407822),
🔷 Top 5 Nodes by Betweenness Centrality: [(np.int64(21), 0.1343907193925181), (np.int64(8), 0.11015212817951883), (np.int64(7), 0.075023924340
🔷 Top 5 Nodes by Closeness Centrality: [(np.int64(21), 0.7649572649572649), (np.int64(8), 0.7316076294277929), (np.int64(7), 0.685823754789272
```




```
# Community Analysis

import community as community_louvain  # Louvain method for community detection
import networkx as nx
import matplotlib.pyplot as plt

# **Run Louvain Community Detection**
partition = community_louvain.best_partition(G_filtered)  # Dictionary {node: community_id}

# **Extract communities**
communities = set(partition.values())  # Unique community IDs

# **Assign colors to communities**
color_map = {node: partition[node] for node in G_filtered.nodes()}

# **Visualize Communities (Less Dense & Readable)**
plt.figure(figsize=(12, 8))
pos = nx.spring_layout(G_filtered, seed=42, k=0.7)  # Spaced layout

# Draw nodes with colors based on communities
nx.draw_networkx_nodes(G_filtered, pos, node_color=list(color_map.values()),
                       cmap=plt.cm.rainbow, node_size=100, alpha=0.8)

# Draw edges with transparency to reduce clutter
nx.draw_networkx_edges(G_filtered, pos, alpha=0.2, edge_color="gray")

# Highlight top influencers for readability
top_nodes = sorted(G_filtered.degree, key=lambda x: x[1], reverse=True)[:5]
top_nodes_labels = {node: str(node) for node, _ in top_nodes}
nx.draw_networkx_labels(G_filtered, pos, labels=top_nodes_labels, font_size=10, font_color="black")

# **Plot Community Graph**
plt.title("Community Detection in User Interaction Network", fontsize=14)
plt.axis("off")
plt.show()
```




```
# Clustering Coefficient

clustering_coeffs = nx.clustering(G_filtered)
avg_clustering = sum(clustering_coeffs.values()) / len(clustering_coeffs)

print(f"📊 Average Clustering Coefficient: {avg_clustering:.4f}")
```

```
📊 Average Clustering Coefficient: 0.2048
```

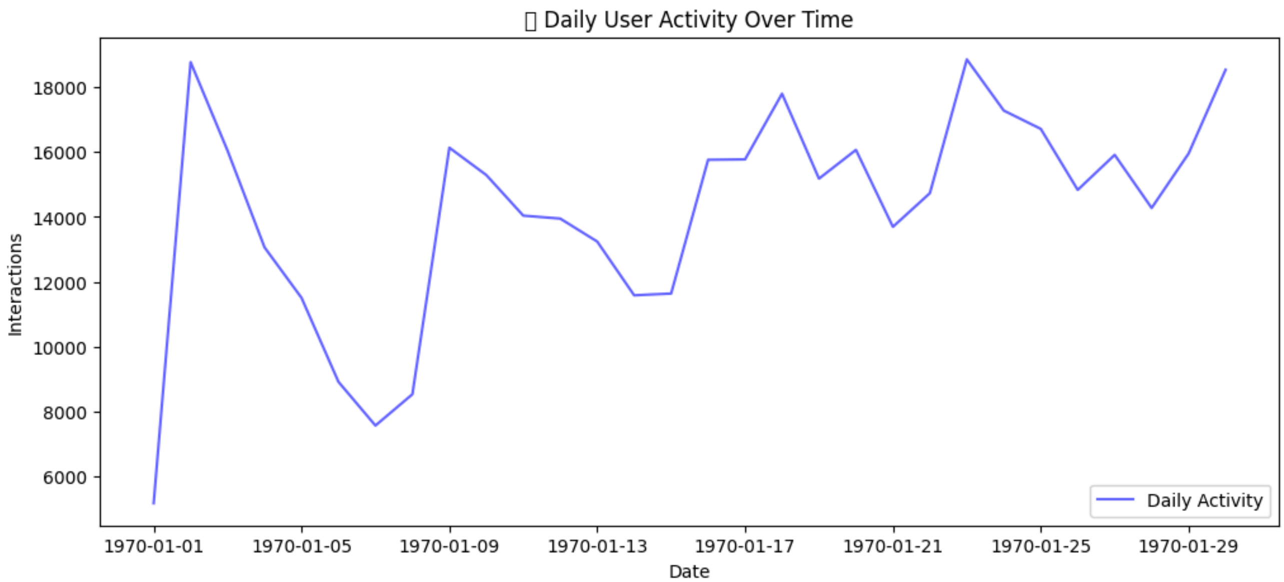
```
#The Most Influential Users (Research Question 2)

# Compute degree centrality (number of connections)
degree_centrality = nx.degree_centrality(G_filtered)

# Compute betweenness centrality (key intermediaries)
betweenness_centrality = nx.betweenness_centrality(G_filtered)

# Compute PageRank (influence score)
pagerank_scores = nx.pagerank(G_filtered)

# Identify top 5 users for each metric
top_degree = sorted(degree_centrality.items(), key=lambda x: x[1], reverse=True)[:5]
top_betweenness = sorted(betweenness_centrality.items(), key=lambda x: x[1], reverse=True)[:5]
top_pagerank = sorted(pagerank_scores.items(), key=lambda x: x[1], reverse=True)[:5]

print("🏆 Top 5 Hubs (Most Connections):", top_degree)
print("🔗 Top 5 Intermediaries (Betweenness Centrality):", top_betweenness)
print("🌟 Top 5 Influencers (PageRank):", top_pagerank)
```

```
🏆 Top 5 Hubs (Most Connections): [(np.int64(21), 0.6945996275605214), (np.int64(8), 0.633147113594041), (np.int64(7), 0.5418994413407822), (np
🔗 Top 5 Intermediaries (Betweenness Centrality): [(np.int64(21), 0.1343907193925181), (np.int64(8), 0.11015212817951883), (np.int64(7), 0.0750
🌟 Top 5 Influencers (PageRank): [(np.int64(21), 0.03509645942625998), (np.int64(8), 0.028448399524794134), (np.int64(7), 0.019866293882176 84),
```

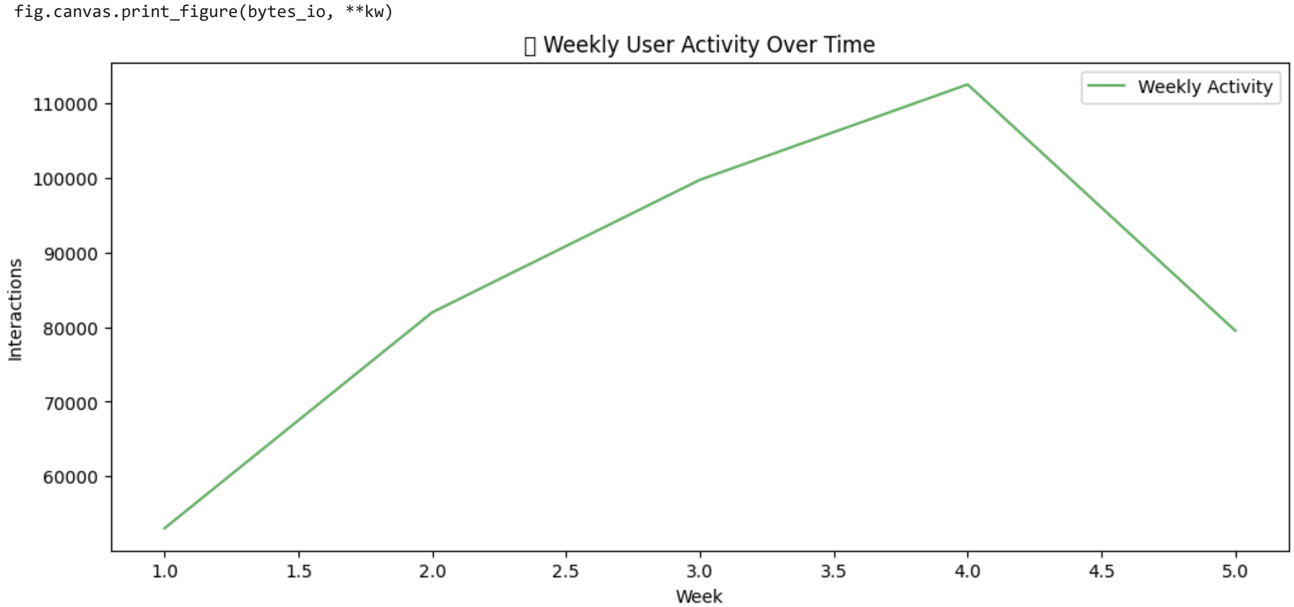
```
#Determining How User Activity Evolve Over Time? (Research Question 1)

import matplotlib.pyplot as plt

# Convert TIMESTAMP column to datetime format
df["TIMESTAMP"] = pd.to_datetime(df["TIMESTAMP"])

# Extract date and hour for analysis
```

```
df["DATE"] = df["TIMESTAMP"].dt.date
df["HOUR"] = df["TIMESTAMP"].dt.hour
df["WEEK"] = df["TIMESTAMP"].dt.isocalendar().week

# Aggregate user interactions over time
daily_activity = df.groupby("DATE").size()
weekly_activity = df.groupby("WEEK").size()

# 📊 Plot activity trends
plt.figure(figsize=(12, 5))
plt.plot(daily_activity, label="Daily Activity", color="blue", alpha=0.6)
plt.title("📈 Daily User Activity Over Time")
plt.xlabel("Date")
plt.ylabel("Interactions")
plt.legend()
plt.show()

plt.figure(figsize=(12, 5))
plt.plot(weekly_activity, label="Weekly Activity", color="green", alpha=0.6)
plt.title("📊 Weekly User Activity Over Time")
plt.xlabel("Week")
plt.ylabel("Interactions")
plt.legend()
plt.show()
```

```
/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 128200 (\N{CHART WITH UPWARDS TREND}) missing from fo
  fig.canvas.print_figure(bytes_io, **kw)
```




```
/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 128202 (\N{BAR CHART}) missing from font(s) DejaVu Sa
  fig.canvas.print_figure(bytes_io, **kw)
```



```
# Calculate network density
density = nx.density(G_filtered)

# Find number of connected components
num_components = nx.number_connected_components(G_filtered)

print(f"Network Density: {density:.6f}")
print(f"Number of Connected Components: {num_components}")
```

```
Network Density: 0.048126
Number of Connected Components: 1
```

```
# Overall Structure of The Network (Research Question 3)

import networkx.algorithms.community as nx_comm

# Detect communities using the Louvain method (optimized for large graphs)
communities = nx_comm.louvain_communities(G_filtered)

# Assign community labels
community_map = {}
for i, community in enumerate(communities):
    for node in community:
        community_map[node] = i

# Add community attribute to nodes
nx.set_node_attributes(G_filtered, community_map, "community")

# Count number of communities
num_communities = len(communities)
print(f"Total Communities Detected: {num_communities}")
```

```
Total Communities Detected: 6
```

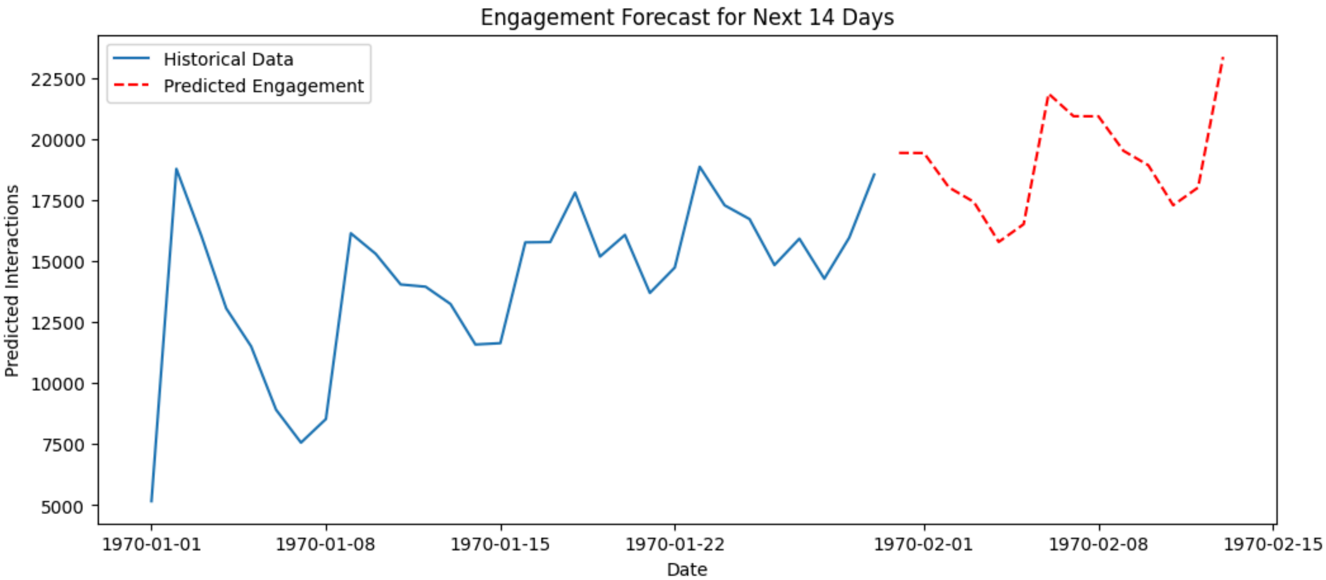
```
#How Interactions Between Users Change Over Time (Research Question 4)

import networkx.algorithms.community as nx_comm

# Detect communities using the Louvain method (optimized for large graphs)
communities = nx_comm.louvain_communities(G_filtered)

# Assign community labels
community_map = {}
for i, community in enumerate(communities):
    for node in community:
        community_map[node] = i

# Add community attribute to nodes
nx.set_node_attributes(G_filtered, community_map, "community")

# Count number of communities
num_communities = len(communities)
print(f"Total Communities Detected: {num_communities}")
```

```
Total Communities Detected: 5
```

```
#Predicting Fututre Engagement (Research Question 5)

from statsmodels.tsa.holtwinters import ExponentialSmoothing

# Train forecasting model (Exponential Smoothing)
model = ExponentialSmoothing(daily_activity, trend="add", seasonal="add", seasonal_periods=7)
fit = model.fit()

# Forecast next 14 days
forecast = fit.forecast(14)

# Plot forecast
plt.figure(figsize=(12, 5))
plt.plot(daily_activity, label="Historical Data")
plt.plot(forecast, label="Predicted Engagement", linestyle="dashed", color="red")
plt.title("Engagement Forecast for Next 14 Days")
plt.xlabel("Date")
plt.ylabel("Predicted Interactions")
plt.legend()
```

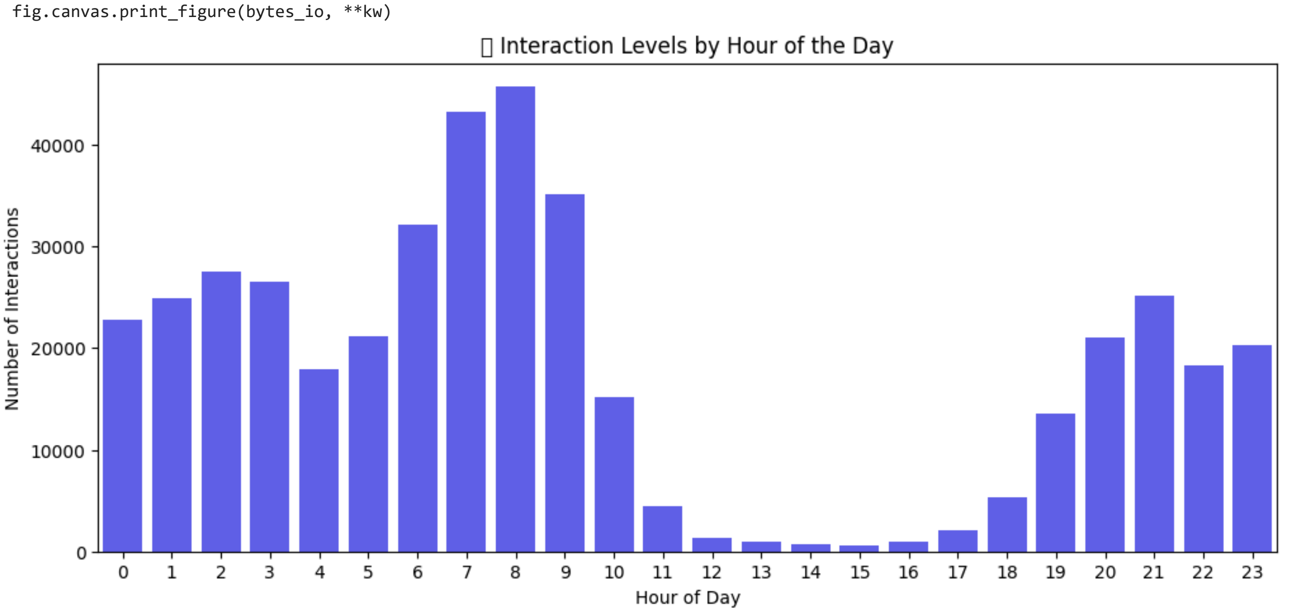
```
plt.show()
```

```
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred
  self._init_dates(dates, freq)
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/holtwinters/model.py:918: ConvergenceWarning: Optimization failed to converge. Check mle
  warnings.warn(
```




```
#Factors That Impact User Interaction Levels (research Question 6)

import seaborn as sns

# Aggregate user activity by hour
hourly_activity = df.groupby("HOUR").size()

# Plot heatmap for hour-based trends
plt.figure(figsize=(12, 5))
sns.barplot(x=hourly_activity.index, y=hourly_activity.values, color="blue", alpha=0.7)
plt.title("🕒 Interaction Levels by Hour of the Day")
plt.xlabel("Hour of Day")
plt.ylabel("Number of Interactions")
plt.show()
```

```
/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 128338 (\N{CLOCK FACE THREE OCLOCK}) missing from for
  fig.canvas.print_figure(bytes_io, **kw)
```




```
#Computing Basic Statistics

import networkx as nx
```

```
# Compute basic network statistics
num_nodes = G.number_of_nodes()
num_edges = G.number_of_edges()
density = nx.density(G)

# Average path length (only for connected graphs)
if nx.is_connected(G):
    avg_path_length = nx.average_shortest_path_length(G)
else:
    largest_cc = max(nx.connected_components(G), key=len)
    G_largest = G.subgraph(largest_cc)
    avg_path_length = nx.average_shortest_path_length(G_largest)

# Clustering coefficient (average over all nodes)
avg_clustering = nx.average_clustering(G)

# Print results
print(f"Number of Nodes: {num_nodes}")
print(f"Number of Edges: {num_edges}")
print(f"Graph Density: {density:.4f}")
print(f"Average Path Length: {avg_path_length:.4f}")
print(f"Average Clustering Coefficient: {avg_clustering:.4f}")
```

```
Number of Nodes: 4159
Number of Edges: 18863
Graph Density: 0.0022
Average Path Length: 2.7520
Average Clustering Coefficient: 0.2059
```

```
print(filtered_df.shape)  # Check number of rows after filtering
print(filtered_df.head())  # View sample data
```

```
import networkx as nx

# Compute basic network statistics
num_nodes = G.number_of_nodes()
num_edges = G.number_of_edges()
density = nx.density(G)

# Average path length (only for connected graphs)
if nx.is_connected(G):
    avg_path_length = nx.average_shortest_path_length(G)
else:
    largest_cc = max(nx.connected_components(G), key=len)
    G_largest = G.subgraph(largest_cc)
    avg_path_length = nx.average_shortest_path_length(G_largest)

# Clustering coefficient (average over all nodes)
avg_clustering = nx.average_clustering(G)

# Print results
print(f"Number of Nodes: {num_nodes}")
print(f"Number of Edges: {num_edges}")
print(f"Graph Density: {density:.4f}")
print(f"Average Path Length: {avg_path_length:.4f}")
print(f"Average Clustering Coefficient: {avg_clustering:.4f}")
```

```
Number of Nodes: 7047
Number of Edges: 177891
Graph Density: 0.0072
Average Path Length: 1.9992
Average Clustering Coefficient: 0.8532
```

```
import networkx as nx
import pandas as pd

# Compute only degree centrality first
degree_centrality = nx.degree_centrality(G)

# Convert to DataFrame
degree_df = pd.DataFrame({
    "Node": list(degree_centrality.keys()),
    "Degree Centrality": list(degree_centrality.values())
})

# Sort by Degree Centrality
```

```
degree_df = degree_df.sort_values(by="Degree Centrality", ascending=False)

# Display top 10 most central users
print(degree_df.head(10))
```

```
    Node  Degree Centrality
1      1           0.950043
3      3           0.874113
4      4           0.791939
7      7           0.740420
8      8           0.711609
5      5           0.703662
9      9           0.654272
14    14           0.647885
13    13           0.624184
6      6           0.608288
```

```
import pandas as pd

df = pd.read_csv("merged_mooc_data.csv")  # Ensure the correct file path
print("Dataset Shape:", df.shape)  # Check if data loads
print(df.head())  # Preview the first few rows
```

```
import os

# List all files in the current directory
print("Files in Directory:", os.listdir())

# Check if the dataset file exists
file_path = "merged_mooc_data.csv"
print("Does file exist?", os.path.exists(file_path))
```

```
#Laod a Small Sample

import pandas as pd

df = pd.read_csv("merged_mooc_data.csv", nrows=5)  # Load only 5 rows
print("Small dataset successfully loaded!")
print(df)
```

```
Small dataset successfully loaded!
   ACTIONID  USERID  TARGETID            TIMESTAMP  FEATURE0  FEATURE1  \
0         1       0         1  1970-01-01 00:00:06 -0.319991 -0.435701
1         2       0         2  1970-01-01 00:00:41 -0.319991 -0.435701
2         3       0         1  1970-01-01 00:00:49 -0.319991 -0.435701
3         4       0         2  1970-01-01 00:00:51 -0.319991 -0.435701
4         5       0         3  1970-01-01 00:00:55 -0.319991 -0.435701

   FEATURE2  FEATURE3  LABEL
0  0.106784 -0.067309    0.0
1  0.106784 -0.067309    0.0
2  0.106784 -0.067309    0.0
3  0.106784 -0.067309    0.0
4  0.106784 -0.067309    0.0
```