Network Construction - Netflix

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

```
# Montiamo Google Drive per accedere ai file salvati
from google.colab import drive
drive.mount('/content/drive')

path_netflix="/content/drive/Shareddrives/information project/ANALISI X (Twitter)/Fase 2 - Scraping twitter/tw
path_svb="/content/drive/Shareddrives/information project/ANALISI X (Twitter)/Fase 2 - Scraping twitter/tweets
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", f
```

Netflix

```
# Carica il dataset (SVB o Netflix)
df = pd.read_csv(path_netflix)
df.head()
```

	id	text	twitterUrl	retweetCount	replyCount	likeCount	quo
0	1516532661803573258	#netflix 👀👀👀 this stock needs more than a pray...	https://twitter.com/live2beingu/status/1516532...	28	11	156	
1	1516544297147179009	◆ の \nさきほどの NETFLIX 3 の は2.21 で、 で ...	https://twitter.com/goto_finance/status/151654...	382	6	2201	

```
def extract_author(url):
    """
    Estrae l'username autore dal link Twitter
    """
    try:
        return url.split("twitter.com/")[1].split("/")[0]
    except:
        return None

df["author"] = df["twitterUrl"].apply(extract_author)
df = df.dropna(subset=["author"])
```

to spend a

```
def extract_mentions(text):
    """
    Estrae tutte le menzioni @username dal testo
    """
    return re.findall(r'@(\w+)', text)

def extract_retweet(text):
    """
    Estrae l'utente retweettato se il tweet è un RT
    """
    match = re.match(r'RT @(\w+):', text)
    return match.group(1) if match else None
```

```
df["mentions"] = df["text"].apply(extract_mentions)
df["retweet_user"] = df["text"].apply(extract_retweet)
```

```
G_net = nx.DiGraph()

for _, row in df.iterrows():
    author = row["author"]

    # Aggiungiamo il nodo autore
    G_net.add_node(author)

    # Retweet: A → B
    if row["retweet_user"]:
        target = row["retweet_user"]
        if G_net.has_edge(author, target):
            G_net[author][target]["weight"] += 1
        else:
            G_net.add_edge(author, target, weight=1)

    # Mentions: A → B
    for target in row["mentions"]:
        if G_net.has_edge(author, target):
            G_net[author][target]["weight"] += 1
        else:
            G_net.add_edge(author, target, weight=1)
```

Network Pre-processing (Netflix)

```
# Rimuove nodi con grado totale minore o uguale a 1
low_degree_nodes = [n for n in G_net.nodes if G_net.degree(n) <= 1]
G_net.remove_nodes_from(low_degree_nodes)

print(f"Nodi a bassa interazione rimossi: {len(low_degree_nodes)}")
```

```
Nodi a bassa interazione rimossi: 813
```

```
# Rimuove nodi completamente isolati
isolated_nodes = list(nx.isolates(G_net))
G_net.remove_nodes_from(isolated_nodes)

print(f"Nodi isolati rimossi: {len(isolated_nodes)}")
```

```
Nodi isolati rimossi: 28
```

```
print(f"Nodi rimanenti: {G_net.number_of_nodes()}")
print(f"Archi rimanenti: {G_net.number_of_edges()}")
```

```
Nodi rimanenti: 56
Archi rimanenti: 52
```

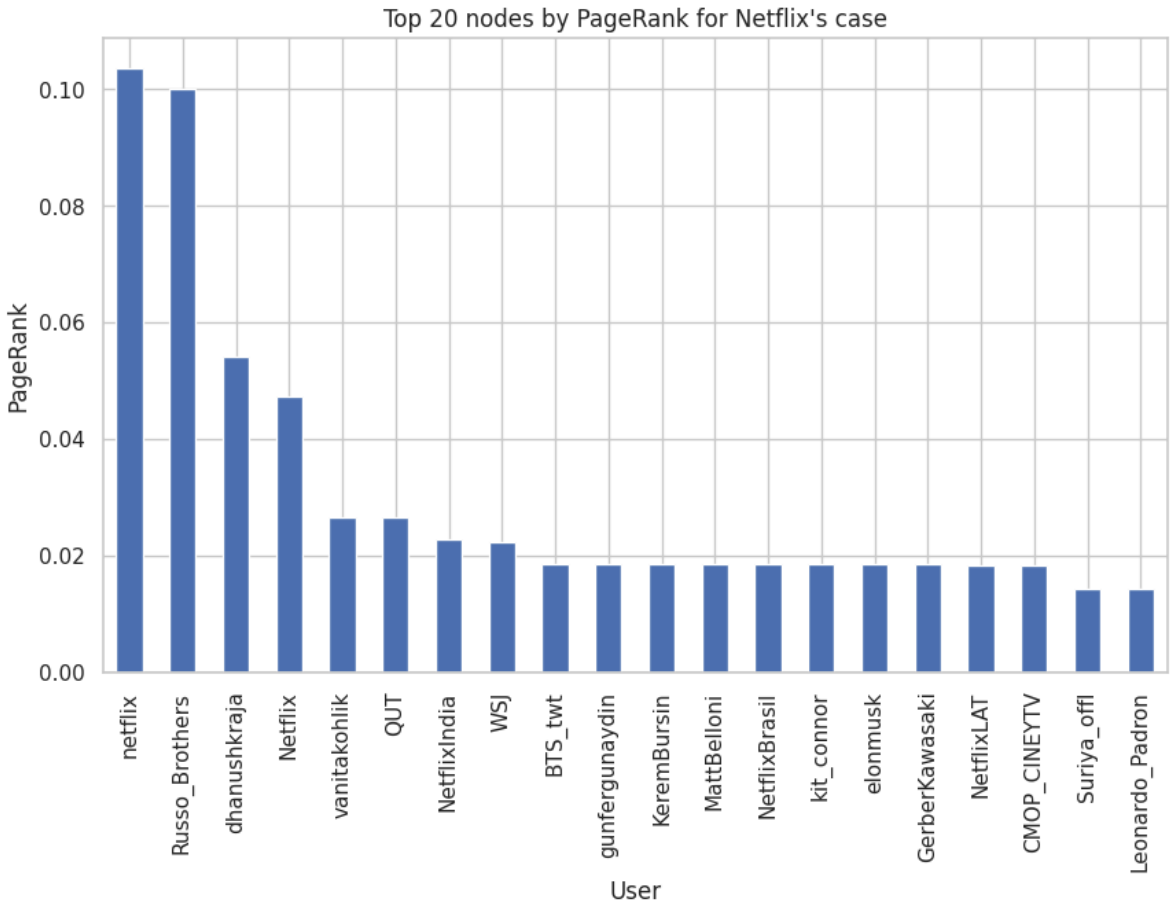
Centrality Analysis (Netflix)

```
# In-degree centrality
in_degree_nflx = nx.in_degree_centrality(G_net)

# PageRank (pesato)
pagerank_nflx = nx.pagerank(G_net, weight="weight")

# Betweenness centrality
betweenness_nflx = nx.betweenness_centrality(G_net, normalized=True)
```

```
centrality_df = pd.DataFrame({
    "in_degree": in_degree_nflx,
    "pagerank": pagerank_nflx,
    "betweenness": betweenness_nflx
})

centrality_df = centrality_df.sort_values("pagerank", ascending=False)
centrality_df.head(10)
```

	in_degree	pagerank	betweenness
netflix	0.218182	0.103595	0.008754
Russo_Brothers	0.036364	0.100006	0.000000
dhanushkraja	0.018182	0.054054	0.000000
Netflix	0.109091	0.047115	0.000000
vanitakohlik	0.036364	0.026440	0.000000
QUT	0.036364	0.026440	0.000000
NetflixIndia	0.036364	0.022817	0.000000
WSJ	0.018182	0.022178	0.000000
BTS_twt	0.018182	0.018555	0.000000
gunfergunaydin	0.036364	0.018555	0.000000

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
centrality_df["pagerank"].head(20).plot(kind="bar")
plt.title("Top 20 nodes by PageRank for Netflix's case")
plt.ylabel("PageRank")
plt.xlabel("User")
plt.show()
```



Community detection (Netflix)

```
!pip install python-igraph leidenalg
```

```
Requirement already satisfied: python-igraph in /usr/local/lib/python3.12/dist-packages (1.0.0)
Requirement already satisfied: leidenalg in /usr/local/lib/python3.12/dist-packages (0.11.0)
Requirement already satisfied: igraph==1.0.0 in /usr/local/lib/python3.12/dist-packages (from python-igraph)
Requirement already satisfied: texttable>=1.6.2 in /usr/local/lib/python3.12/dist-packages (from igraph==1.0.
```

```
import igraph as ig
import leidenalg
import pandas as pd
import networkx as nx
import matplotlib.pyplot as plt
```

```
# Preparazione nodi
nodes = list(G_net.nodes())
node_index = {node: i for i, node in enumerate(nodes)}

# Preparazione archi
edges = []
weights = []

for u, v, d in G_net.edges(data=True):
    edges.append((node_index[u], node_index[v]))
    weights.append(d["weight"])

# Costruzione grafo iGraph
g_ig = ig.Graph(
    n=len(nodes),
    edges=edges,
    directed=True
)

g_ig.es["weight"] = weights
g_ig.vs["name"] = nodes
```

```
partition = leidenalg.find_partition(
    g_ig,
    leidenalg.RBConfigurationVertexPartition,
    weights="weight",
    resolution_parameter=1.0
)
```

```
# Assegniamo community a ogni nodo
community_map = {
    g_ig.vs[i]["name"]: partition.membership[i]
    for i in range(len(g_ig.vs))
}

nx.set_node_attributes(G, community_map, "community")
```

```
num_communities = len(set(partition.membership))
print(f"Numero di community individuate: {num_communities}")
```

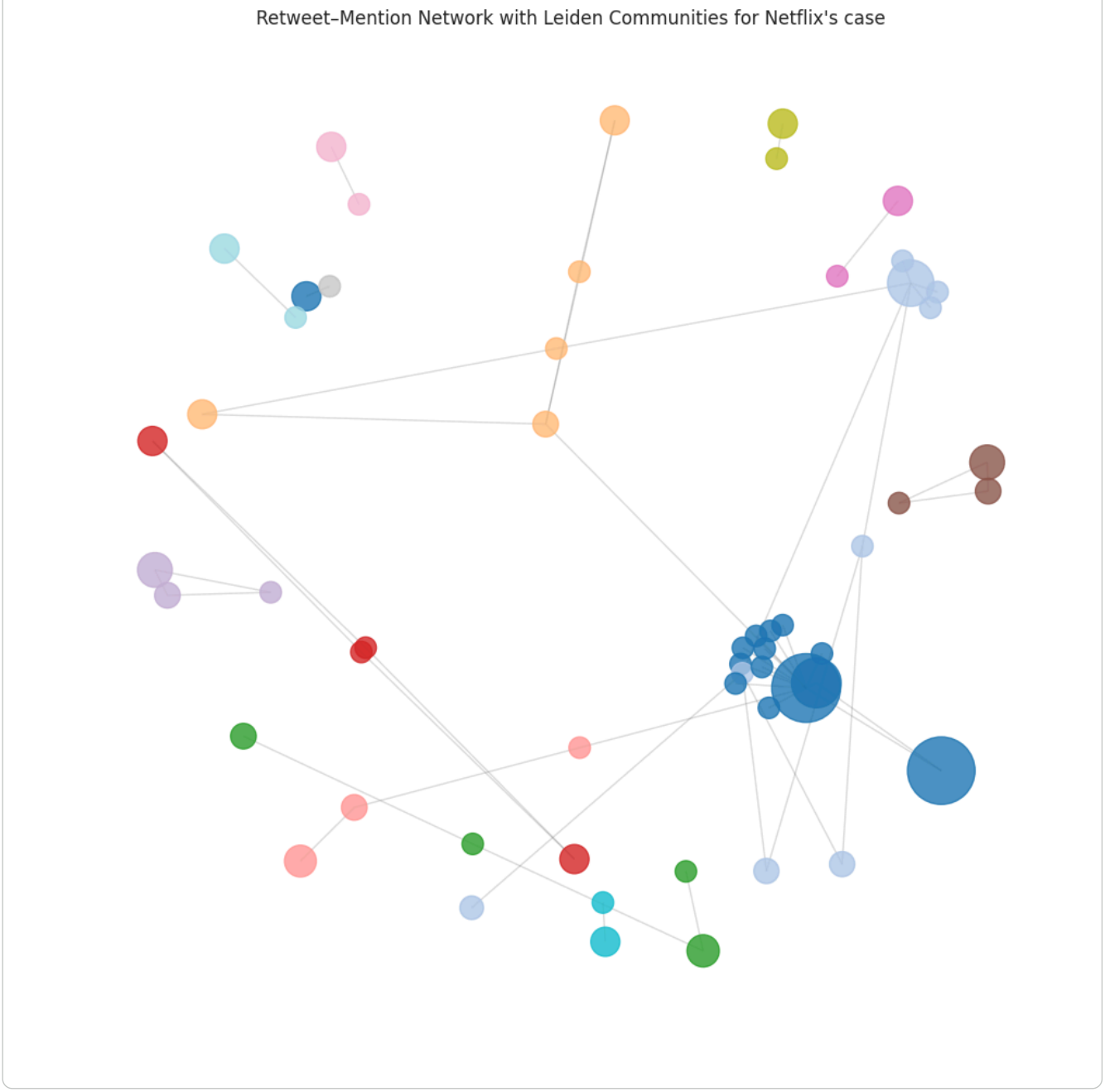
```
Numero di community individuate: 14
```

```
community_df = (
    pd.DataFrame.from_dict(community_map, orient="index", columns=["community"])
    .reset_index()
    .rename(columns={"index": "user"})
)

community_sizes = community_df["community"].value_counts().reset_index()
community_sizes.columns = ["community", "size"]
community_sizes
```

	community	size
0	0	13
1	1	9
2	2	5
3	3	4
4	4	4
5	5	3
6	6	3
7	7	3
8	10	2
9	13	2
10	12	2
11	9	2
12	8	2
13	11	2

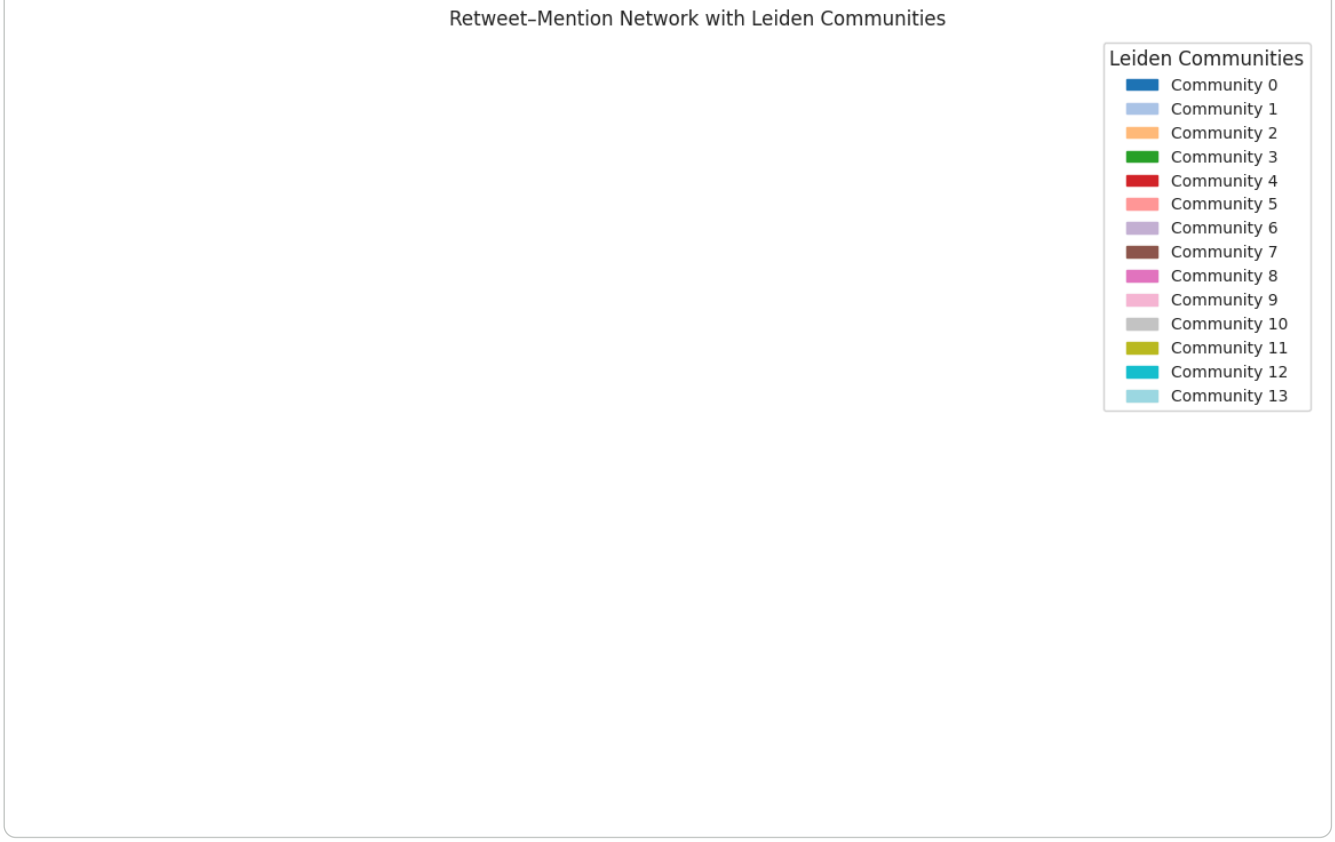
```
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 12))
pos = nx.spring_layout(G_net, k=0.15, seed=42)

node_colors = [G_net.nodes[n]["community"] for n in G_net.nodes]
node_sizes = [pagerank_nflx[n] * 20000 for n in G_net.nodes]

nx.draw_networkx_nodes(
    G_net,
    pos,
    node_color=node_colors,
    cmap="tab20",
    node_size=node_sizes,
    alpha=0.8
)

# Get the list of unique community IDs from G_net
# Assuming G_net nodes have 'community' attribute set from cell 2cnCJ4hZlUOv
unique_communities = sorted(list(set(nx.get_node_attributes(G_net, 'community').values())))

# Get the 'tab20' colormap
cmap = plt.cm.get_cmap('tab20', len(unique_communities))

# Create manual patches for the legend
patches = [mpatches.Patch(color=cmap(c_id), label=f"Community {c_id}") for c_id in unique_communities]

nx.draw_networkx_edges(G_net, pos, alpha=0.15, arrows=False)

plt.title("Retweet–Mention Network with Leiden Communities for Netflix's case")
plt.axis("off")
plt.show()
```

```
/tmp/ipython-input-2519955695.py:23: MatplotlibDeprecationWarning: The get_cmap function was deprecated in M
  cmap = plt.cm.get_cmap('tab20', len(unique_communities))
```




```
import matplotlib.patches as mpatches

# Get the list of unique community IDs from G_net
unique_communities = sorted(list(set(nx.get_node_attributes(G_net, 'community').values())))

# Get the 'tab20' colormap
cmap = plt.cm.get_cmap('tab20', len(unique_communities))

# Creazione manuale delle patch per la leggenda
patches = [mpatches.Patch(color=cmap(c_id), label=f"Community {c_id}") for c_id in unique_communities]

plt.legend(handles=patches, loc='best', fontsize=10, title="Leiden Communities")
plt.title("Retweet-Mention Network with Leiden Communities")
plt.axis("off")
plt.show()
```

```
/tmp/ipython-input-3354434501.py:7: MatplotlibDeprecationWarning: The get_cmap function was deprecated in Mat
  cmap = plt.cm.get_cmap('tab20', len(unique_communities))
```



Struttura delle community ed echo-like behavior (Netflix)

```
intra_edges = 0
inter_edges = 0

for u, v in G_net.edges():
    if G_net.nodes[u]["community"] == G_net.nodes[v]["community"]:
        intra_edges += 1
    else:
        inter_edges += 1

total_edges = intra_edges + inter_edges

print(f"Intra-community edges: {intra_edges}")
print(f"Inter-community edges: {inter_edges}")
print(f"Share intra-community: {intra_edges / total_edges:.2f}")
```

```
Intra-community edges: 48
Inter-community edges: 4
Share intra-community: 0.92
```

```
from collections import defaultdict

community_edges = defaultdict(lambda: {"intra": 0, "inter": 0})

for u, v in G_net.edges():
    cu = G_net.nodes[u]["community"]
    cv = G_net.nodes[v]["community"]

    if cu == cv:
        community_edges[cu]["intra"] += 1
    else:
        community_edges[cu]["inter"] += 1

community_structure = pd.DataFrame.from_dict(
    community_edges, orient="index"
).reset_index().rename(columns={"index": "community"})

community_structure["intra_share"] = (
    community_structure["intra"] /
    (community_structure["intra"] + community_structure["inter"])
)
```

```
community_structure.sort_values("intra_share", ascending=False)
```

	community	intra	inter	intra_share
0	1	10	0	1.000000
1	0	13	0	1.000000
3	3	3	0	1.000000
4	8	1	0	1.000000
7	10	1	0	1.000000
6	9	1	0	1.000000
10	12	1	0	1.000000
11	7	3	0	1.000000
8	11	1	0	1.000000
9	6	3	0	1.000000
12	4	4	0	1.000000
13	13	1	0	1.000000
5	2	5	2	0.714286
2	5	2	1	0.666667

SVB

```
# Carica il dataset (SVB o Netflix)
df = pd.read_csv(path_svb)  # cambia nome file se serve
df.head()
```

	id	text	twitterUrl	retweetCount	replyCount	likeCount	qu
0	1634329302845079555	There is more than $22 Trillion in the U.S. ba...	https://twitter.com/gaborgurbacs/status/163432...	1780	307	4956	
1	1634029130923794435	Silicon Valley Bank in the US is struggling fo...	https://twitter.com/AdityaD_Shah/status/163402...	478	367	2323	
2	1634306547638566916	Earlier today, America's	https://twitter.com/itsDanny_V/status/16343065...	487	57	1657	

```
def extract_author(url):
    """
    Estrae l'username autore dal link Twitter
    """
    try:
        return url.split("twitter.com/")[1].split("/")[0]
    except:
        return None

df["author"] = df["twitterUrl"].apply(extract_author)
df = df.dropna(subset=["author"])
```

BILLION

```
def extract_mentions(text):
    """
    Estrae tutte le menzioni @username dal testo
    """
    return re.findall(r'@(\w+)', text)

def extract_retweet(text):
    """
    Estrae l'utente retweettato se il tweet è un RT
    """
    match = re.match(r'RT @(\w+):', text)
```

```
    return match.group(1) if match else None
```

```
df["mentions"] = df["text"].apply(extract_mentions)
df["retweet_user"] = df["text"].apply(extract_retweet)
```

```
G_svb = nx.DiGraph()

for _, row in df.iterrows():
    author = row["author"]

    # Aggiungiamo il nodo autore
    G_svb.add_node(author)

    # Retweet: A → B
    if row["retweet_user"]:
        target = row["retweet_user"]
        if G_svb.has_edge(author, target):
           G_svb[author][target]["weight"] += 1
        else:
            G_svb.add_edge(author, target, weight=1)

    # Mentions: A → B
    for target in row["mentions"]:
        if G_svb.has_edge(author, target):
            G_svb[author][target]["weight"] += 1
        else:
           G_svb.add_edge(author, target, weight=1)
```

Network Pre-processiong (SVB)

```
# Rimuove nodi con grado totale minore o uguale a 1
low_degree_nodes = [n for n in G_svb.nodes if G_svb.degree(n) <= 1]
G_svb.remove_nodes_from(low_degree_nodes)

print(f"Nodi a bassa interazione rimossi: {len(low_degree_nodes)}")
```

```
Nodi a bassa interazione rimossi: 489
```

```
# Rimuove nodi completamente isolati
isolated_nodes = list(nx.isolates(G_svb))
G_svb.remove_nodes_from(isolated_nodes)

print(f"Nodi isolati rimossi: {len(isolated_nodes)}")
```

```
Nodi isolati rimossi: 13
```

```
print(f"Nodi rimanenti: {G_svb.number_of_nodes()}")
print(f"Archi rimanenti: {G_svb.number_of_edges()}")
```

```
Nodi rimanenti: 29
Archi rimanenti: 20
```

Centrality Analysis(svb)

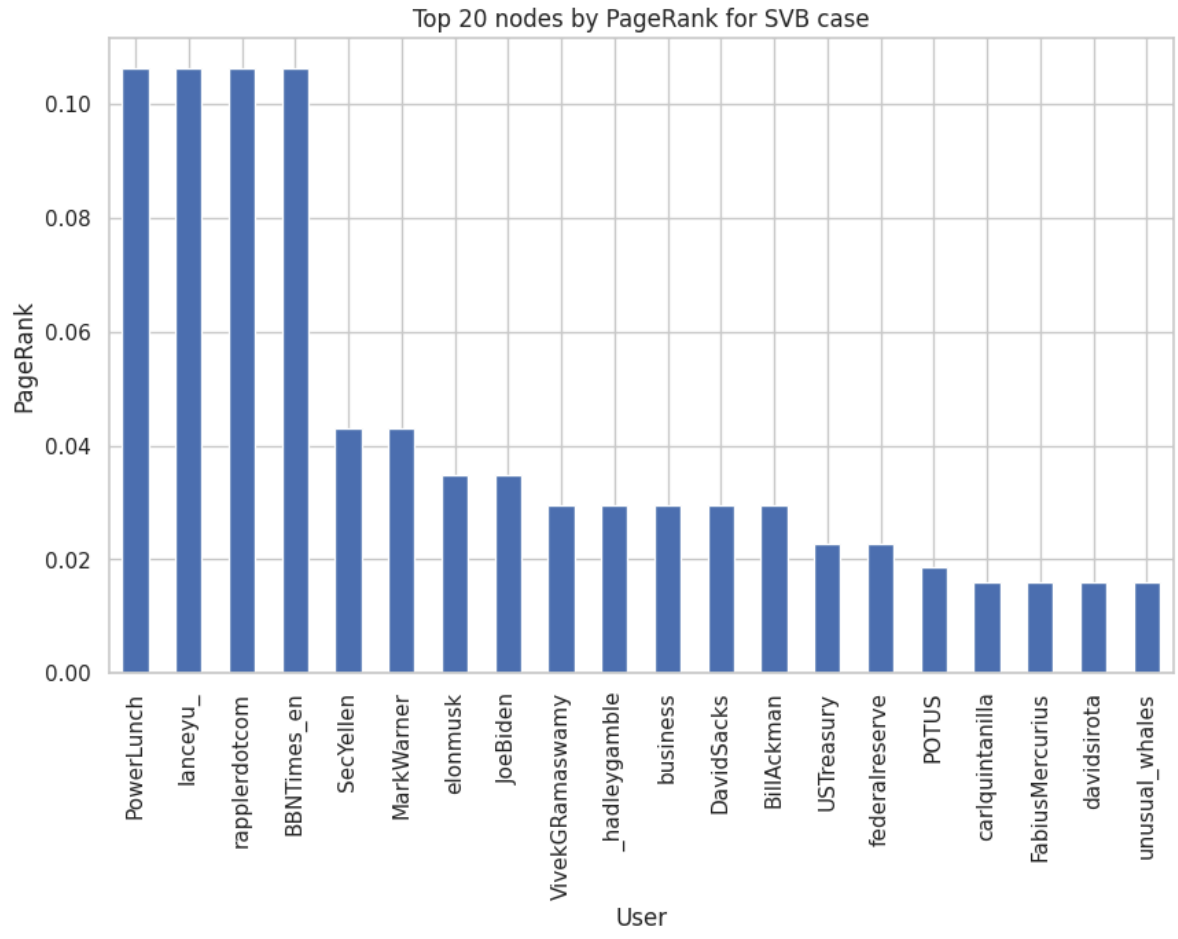
```
# In-degree centrality
in_degree_svb = nx.in_degree_centrality(G_svb)

# PageRank (pesato)
pagerank_svb = nx.pagerank(G_svb, weight="weight")

# Betweenness centrality
betweenness_svb = nx.betweenness_centrality(G_svb, normalized=True)
```

```
centrality_df = pd.DataFrame({
    "in_degree": in_degree_svb,
    "pagerank": pagerank_svb,
    "betweenness": betweenness_svb
})
```

```
centrality_df = centrality_df.sort_values("pagerank", ascending=False)
centrality_df.head(10)
```

	in_degree	pagerank	betweenness
PowerLunch	0.035714	0.106287	0.0
lanceyu_	0.035714	0.106287	0.0
rapplerdotcom	0.035714	0.106287	0.0
BBNTimes_en	0.035714	0.106287	0.0
SecYellen	0.071429	0.043054	0.0
MarkWarner	0.071429	0.043054	0.0
elonmusk	0.071429	0.034922	0.0
JoeBiden	0.071429	0.034922	0.0
VivekGRamaswamy	0.035714	0.029500	0.0
_hadleygamble	0.035714	0.029500	0.0

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
centrality_df["pagerank"].head(20).plot(kind="bar")
plt.title("Top 20 nodes by PageRank for SVB case")
plt.ylabel("PageRank")
plt.xlabel("User")
plt.show()
```




```
import igraph as ig
import leidenalg
import pandas as pd
import networkx as nx
import matplotlib.pyplot as plt
```

```
# Preparazione nodi
nodes = list(G_svb.nodes())
node_index = {node: i for i, node in enumerate(nodes)}

# Preparazione archi
edges = []
weights = []

for u, v, d in G_svb.edges(data=True):
    edges.append((node_index[u], node_index[v]))
    weights.append(d["weight"])

# Costruzione grafo iGraph
g_ig = ig.Graph(
    n=len(nodes),
    edges=edges,
    directed=True
)

g_ig.es["weight"] = weights
g_ig.vs["name"] = nodes
```

```
partition = leidenalg.find_partition(
    g_ig,
    leidenalg.RBConfigurationVertexPartition,
    weights="weight",
    resolution_parameter=1.0
)
```

```
# Assegniamo community a ogni nodo
community_map = {
    g_ig.vs[i]["name"]: partition.membership[i]
    for i in range(len(g_ig.vs))
}

nx.set_node_attributes(G, community_map, "community")
```

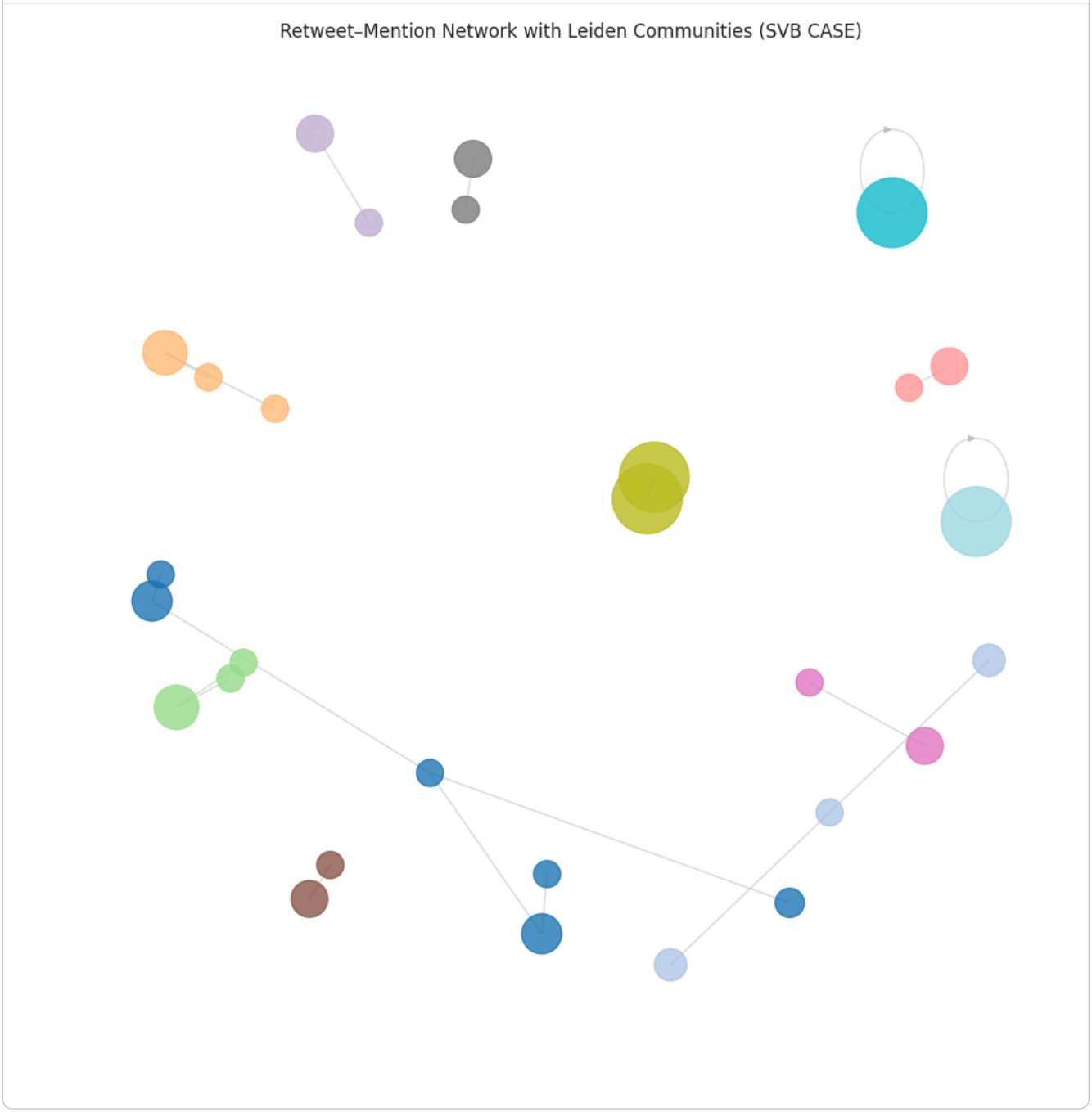
```
num_communities = len(set(partition.membership))
print(f"Numero di community individuate: {num_communities}")
```

```
Numero di community individuate: 12
```

```
community_df = (
    pd.DataFrame.from_dict(community_map, orient="index", columns=["community"])
    .reset_index()
    .rename(columns={"index": "user"})
)

community_sizes = community_df["community"].value_counts().reset_index()
community_sizes.columns = ["community", "size"]
community_sizes
```

	community	size
0	0	6
1	3	3
2	1	3
3	2	3
4	4	2
5	6	2
6	5	2
7	7	2
8	9	2
9	8	2
10	11	1
11	10	1

```
import matplotlib.pyplot as plt

nx.set_node_attributes(G_svb, community_map, "community")

plt.figure(figsize=(12, 12))
pos = nx.spring_layout(G_svb, k=0.15, seed=42)

node_colors = [G_svb.nodes[n]["community"] for n in G_svb.nodes]
node_sizes = [pagerank[n] * 20000 for n in G_svb.nodes]

nx.draw_networkx_nodes(
    G_svb,
    pos,
    node_color=node_colors,
    cmap="tab20",
    node_size=node_sizes,
    alpha=0.8
)

nx.draw_networkx_edges(G_svb, pos, alpha=0.15, arrows=False)

plt.title("Retweet–Mention Network with Leiden Communities (SVB CASE)")
plt.axis("off")
plt.show()
```




```
import matplotlib.patches as mpatches
```

```
# Get the list of unique community IDs from G_net
unique_communities = sorted(list(set(nx.get_node_attributes(G_net, 'community').values())))

# Get the 'tab20' colormap
cmap = plt.cm.get_cmap('tab20', len(unique_communities))

# Creazione manuale delle patch per la leggenda
patches = [mpatches.Patch(color=cmap(c_id), label=f"Community {c_id}") for c_id in unique_communities]

plt.legend(handles=patches, loc='best', fontsize=10, title="Leiden Communities")
plt.title("Retweet-Mention Network with Leiden Communities")
plt.axis("off")
plt.show()
```

```
/tmp/ipython-input-3354434501.py:7: MatplotlibDeprecationWarning: The get_cmap function was deprecated in Mat
  cmap = plt.cm.get_cmap('tab20', len(unique_communities))
```

Struttura delle community ed echo-like behavior (svb)

```
intra_edges = 0
inter_edges = 0

for u, v in G_svb.edges():
    if G_svb.nodes[u]["community"] == G_svb.nodes[v]["community"]:
        intra_edges += 1
    else:
        inter_edges += 1

total_edges = intra_edges + inter_edges

print(f"Intra-community edges: {intra_edges}")
print(f"Inter-community edges: {inter_edges}")
print(f"Share intra-community: {intra_edges / total_edges:.2f}")
```

```
Intra-community edges: 20
Inter-community edges: 0
Share intra-community: 1.00
```

```
from collections import defaultdict

community_edges = defaultdict(lambda: {"intra": 0, "inter": 0})

for u, v in G_svb.edges():
    cu = G_svb.nodes[u]["community"]
    cv = G_svb.nodes[v]["community"]
```

```
    if cu == cv:
        community_edges[cu]["intra"] += 1
    else:
        community_edges[cu]["inter"] += 1

community_structure = pd.DataFrame.from_dict(
    community_edges, orient="index"
).reset_index().rename(columns={"index": "community"})

community_structure["intra_share"] = (
    community_structure["intra"] /
    (community_structure["intra"] + community_structure["inter"])
)

community_structure.sort_values("intra_share", ascending=False)
```

	community	intra	inter	intra_share
0	4	1	0	1.0
1	3	2	0	1.0
2	0	5	0	1.0
3	6	1	0	1.0
4	2	2	0	1.0
5	7	1	0	1.0