FacebookNetworkAnalysis

March 19, 2025

1 Network Analysis on Facebook Dataset

1.1 Step 1: Importing the libraries needed and loading the datasets required for this analysis

[9]:
```
import pandas as pd
import networkx as nx

# 1. Load the datasets
nodes_df = pd.read_csv('musae_facebook_target.csv')  # <-- this is your nodes␣
 ↪file
edges_df = pd.read_csv('musae_facebook_edges.csv')   # <-- this is your edges␣
 ↪file
```

1.2 Step 2: Build the Network Graph

[18]:
```
# Create an empty Graph
G = nx.Graph()

# Add nodes and their attributes
for _, row in nodes_df.iterrows():
    G.add_node(row['id'],
               facebook_id=row['facebook_id'],
               page_name=row['page_name'],
               page_type=row['page_type'])

# Add edges (connections between pages)
for _, row in edges_df.iterrows():
    G.add_edge(row['id_1'], row['id_2'])

# Quick check: print graph info
print(f"Graph has {G.number_of_nodes()} nodes and {G.number_of_edges()} edges")
```

```
Graph has 22470 nodes and 171002 edges
```

1.3 Exploratory Analysis

1.4 What's the average number of connections per page?

[59]:
```python
average_degree = sum(dict(G.degree()).values()) / G.number_of_nodes()
print("Average number of connections per page:", average_degree)
```

```
Average number of connections per page: 15.220471740097908
```

1.5 Community Detection

[65]:
```python
import community as community_louvain

partition = community_louvain.best_partition(G)

# Count how many communities exist
community_count = len(set(partition.values()))
print("Number of communities detected:", community_count)
```

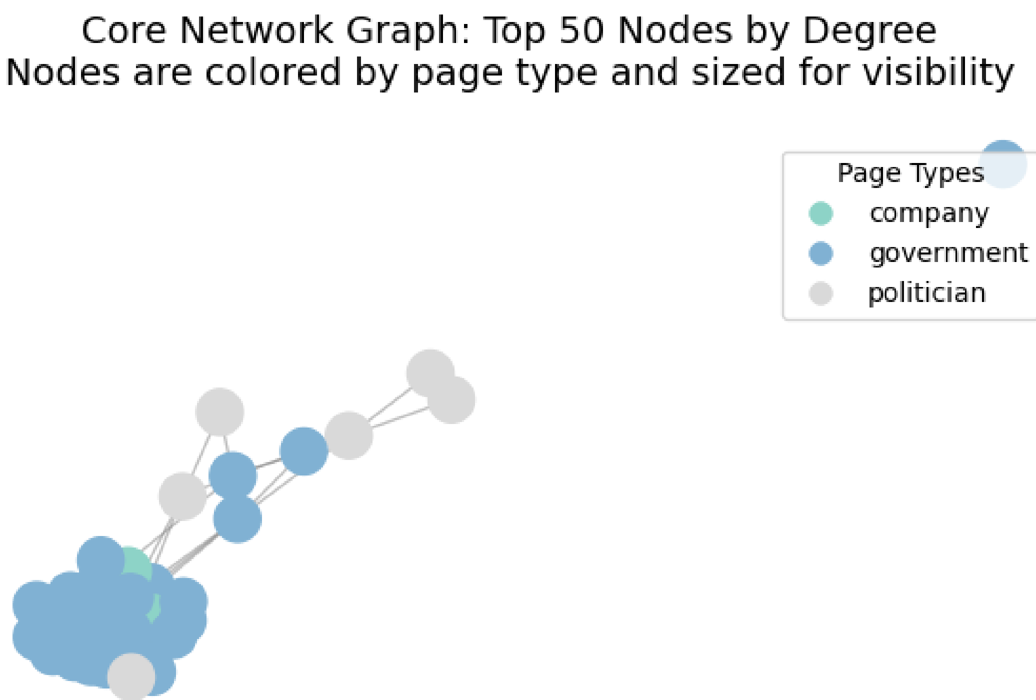
```
Number of communities detected: 60
```

1.6 Core Network Visualization: Top 50 Nodes by Degree

[33]:
```python
import matplotlib.pyplot as plt
from matplotlib.lines import Line2D  # For custom legend elements

# Calculate degrees and get top 50 nodes
degrees = dict(G.degree())
top_nodes = sorted(degrees, key=degrees.get, reverse=True)[:50]
G_core = G.subgraph(top_nodes).copy()

# Generate layout for the graph (spring layout for better spread)
pos = nx.spring_layout(G_core, seed=42)  # Fixed seed for reproducibility

# Extract page types and create a color mapping
page_types = [G_core.nodes[n]['page_type'] for n in G_core.nodes()]
unique_page_types = sorted(set(page_types))  # Get unique page types
color_map = plt.cm.Set3  # Use a categorical colormap
colors = pd.factorize(page_types)[0]  # Numeric encoding for colors
color_dict = {ptype: color_map(i / len(unique_page_types)) for i, ptype in␣
 ↪enumerate(unique_page_types)}

# Create a list of colors for each node based on its page type
node_colors = [color_dict[G_core.nodes[n]['page_type']] for n in G_core.nodes()]

# Plot the core network with detailed annotations
plt.figure(figsize=(6, 4))
# Draw nodes with colors and larger size for visibility
nx.draw_networkx_nodes(G_core, pos, node_color=node_colors, node_size=300)
```

```python
# Draw edges with a light gray color and slight transparency
nx.draw_networkx_edges(G_core, pos, edge_color='gray', alpha=0.5)
# Removed: nx.draw_networkx_labels(G_core, pos, font_size=8, font_color='black')

# Create a custom legend for page types
legend_elements = [
    Line2D([0], [0], marker='o', color='w', label=ptype,
           markerfacecolor=color_dict[ptype], markersize=10)
    for ptype in unique_page_types
]
plt.legend(handles=legend_elements, title="Page Types", loc='upper right',␣
 ↪fontsize=10)

# Add a detailed title and description
plt.title('Core Network Graph: Top 50 Nodes by Degree\n'
          'Nodes are colored by page type and sized for visibility',
          fontsize=14, pad=20)
plt.axis('off')  # Hide axes for a cleaner look
plt.tight_layout()
plt.show()

# Print additional information for context
print("Summary of the Core Network:")
print(f"Number of nodes: {G_core.number_of_nodes()}")
print(f"Number of edges: {G_core.number_of_edges()}")
print("\nPage Type Distribution in Core Network:")
page_type_counts = pd.Series(page_types).value_counts()
for ptype, count in page_type_counts.items():
    print(f"{ptype}: {count} nodes")
```

```
C:\Users\user\AppData\Local\Temp\ipykernel_3368\3570939041.py:16: FutureWarning:
factorize with argument that is not not a Series, Index, ExtensionArray, or
np.ndarray is deprecated and will raise in a future version.
  colors = pd.factorize(page_types)[0]  # Numeric encoding for colors
```



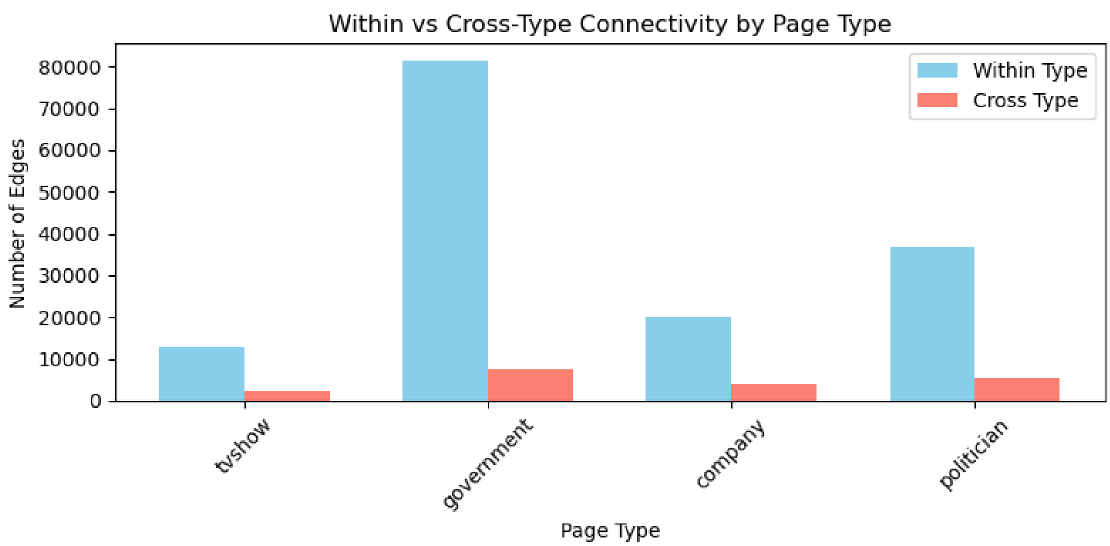

```
Summary of the Core Network:
Number of nodes: 50
Number of edges: 503

Page Type Distribution in Core Network:
government: 42 nodes
politician: 6 nodes
company: 2 nodes
```

1.7 Analyzing Connectivity Patterns by Page Type

[93]:

```python
# Calculate within-type and cross-type edges
page_type_edges = {'within': {}, 'cross': {}}
for edge in G.edges():
    type1 = G.nodes[edge[0]]['page_type']
    type2 = G.nodes[edge[1]]['page_type']
    if type1 == type2:
        page_type_edges['within'][type1] = page_type_edges['within'].get(type1, 0) + 1
    else:
        # Split edge count between both types
        page_type_edges['cross'][type1] = page_type_edges['cross'].get(type1, 0) + 0.5
```

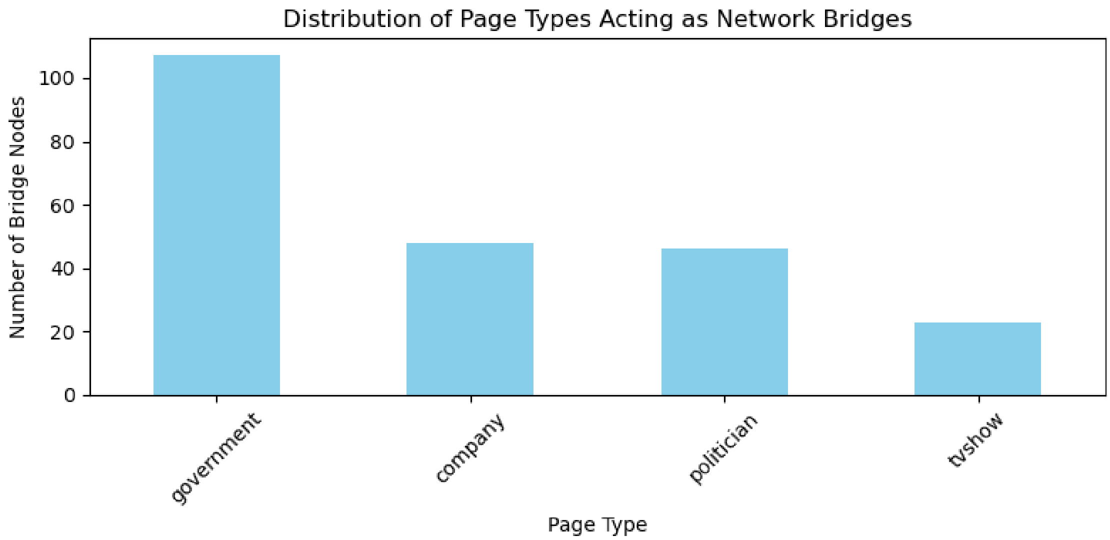
```
        page_type_edges['cross'][type2] = page_type_edges['cross'].get(type2,␣
↪0) + 0.5

# Prepare data for plotting
types = nodes_df['page_type'].unique()
within_counts = [page_type_edges['within'].get(t, 0) for t in types]
cross_counts = [page_type_edges['cross'].get(t, 0) for t in types]

# Create bar chart
plt.figure(figsize=(8, 4))
bar_width = 0.35
x = range(len(types))
plt.bar(x, within_counts, bar_width, label='Within Type', color='skyblue')
plt.bar([i + bar_width for i in x], cross_counts, bar_width, label='Cross␣
↪Type', color='salmon')
plt.xticks([i + bar_width/2 for i in x], types, rotation=45)
plt.xlabel('Page Type')
plt.ylabel('Number of Edges')
plt.title('Within vs Cross-Type Connectivity by Page Type')
plt.legend()
plt.tight_layout()
plt.show()
```



1.8 Research Question 1: How do page types influence the formation of "bridge" connections across densely connected subgraphs?

[95]:

```
import matplotlib.pyplot as plt
from community import community_louvain
import random

# Optimize: Work with the largest connected component to reduce size
largest_cc = max(nx.connected_components(G), key=len)
G = G.subgraph(largest_cc).copy()

# Optimize: Subsample nodes (e.g., 20% of the graph) for faster computation
sample_size = int(0.2 * G.number_of_nodes())
sampled_nodes = random.sample(list(G.nodes()), sample_size)
G_sampled = G.subgraph(sampled_nodes).copy()

# Detect communities using Louvain algorithm (on sampled graph)
partition = community_louvain.best_partition(G_sampled)

# Calculate approximate betweenness centrality (faster than exact)
# k parameter limits the number of nodes used for approximation
betweenness = nx.betweenness_centrality(G_sampled, k=min(100, G_sampled.
 ↪number_of_nodes()), seed=42)
bridge_nodes = {node: score for node, score in betweenness.items() if score >␣
 ↪sorted(betweenness.values(), reverse=True)[int(0.05 * len(betweenness))]}  #␣
 ↪Top 5%

# Analyze page types of bridge nodes
bridge_page_types = {node: G_sampled.nodes[node]['page_type'] for node in␣
 ↪bridge_nodes}
page_type_counts = pd.Series(bridge_page_types.values()).value_counts()

# Visualization
plt.figure(figsize=(8, 4))
page_type_counts.plot(kind='bar', color='skyblue')
plt.title('Distribution of Page Types Acting as Network Bridges')
plt.xlabel('Page Type')
plt.ylabel('Number of Bridge Nodes')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

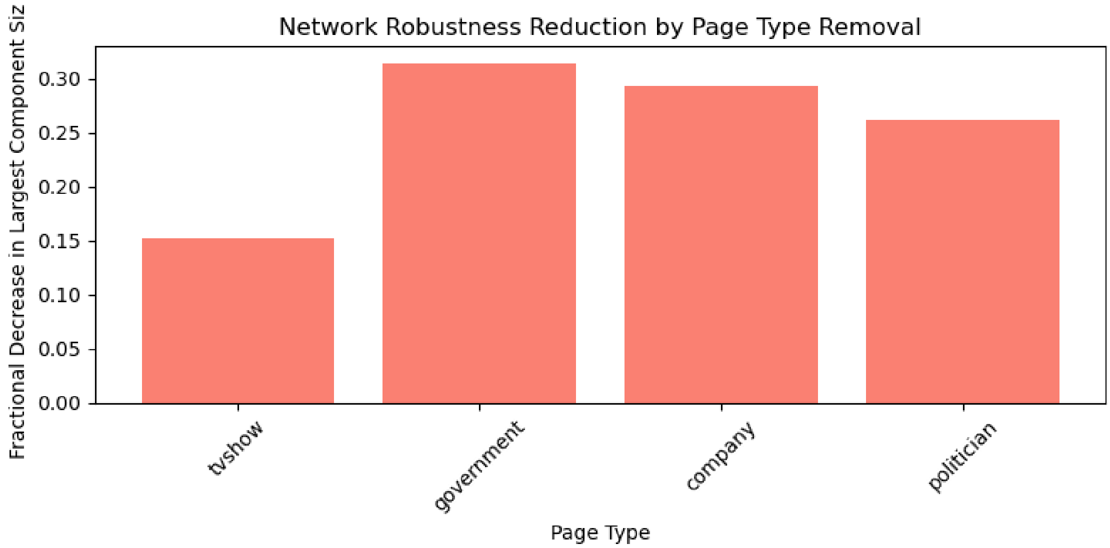


1.9 Research Question 2: How does the structural robustness of the network change when removing pages of specific types?

[101]:

```
import matplotlib.pyplot as plt

# Measure initial robustness (largest connected component)
initial_size = len(max(nx.connected_components(G), key=len))

# Simulate removal by page type
page_types = nodes_df['page_type'].unique()
robustness_changes = {}
for ptype in page_types:
    G_copy = G.copy()
    nodes_to_remove = [n for n, attr in G_copy.nodes(data=True) if␣
 ↪attr['page_type'] == ptype]
    G_copy.remove_nodes_from(nodes_to_remove)
    new_size = len(max(nx.connected_components(G_copy), key=len)) if G_copy.
 ↪number_of_nodes() > 0 else 0
    robustness_changes[ptype] = (initial_size - new_size) / initial_size

# Visualization
plt.figure(figsize=(8,4))
plt.bar(robustness_changes.keys(), robustness_changes.values(), color='salmon')
plt.title('Network Robustness Reduction by Page Type Removal')
plt.xlabel('Page Type')
plt.ylabel('Fractional Decrease in Largest Component Size')
plt.xticks(rotation=45)
plt.tight_layout()
```

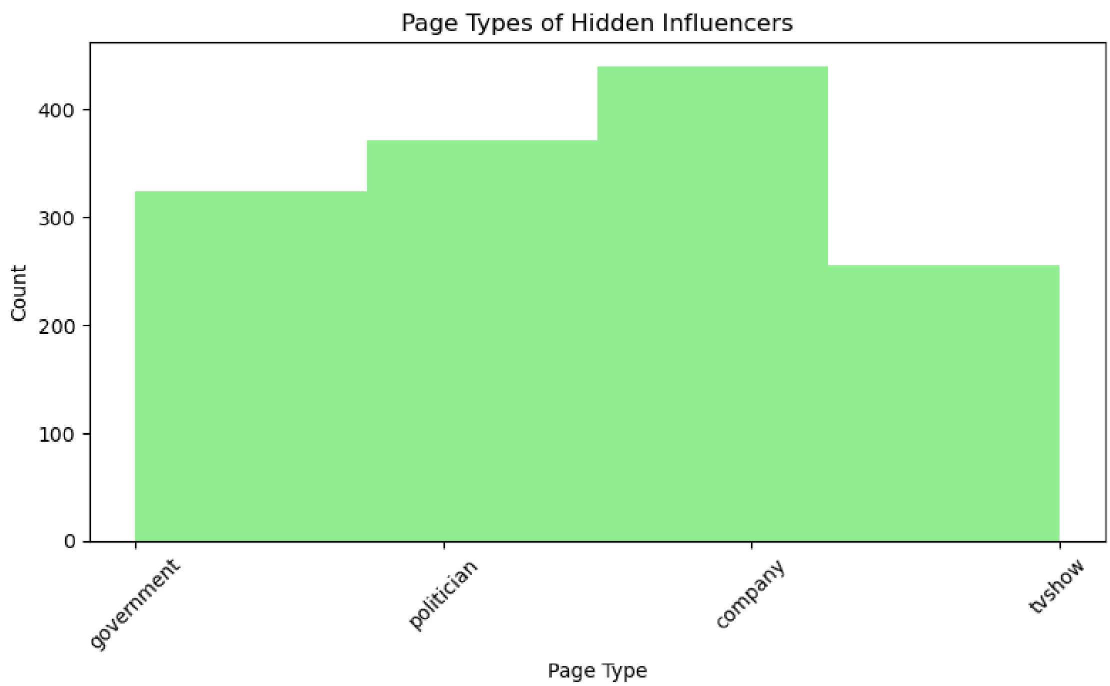
```
plt.show()
```



1.10 Research Question 3: Are there "hidden influencers" with low degree but high control over information flow between page types?

[114]:
```
import matplotlib.pyplot as plt
# Calculate degree and clustering for sampled graph
degrees = dict(G.degree())
clustering = nx.clustering(G)
# Find hidden influencers: low degree (bottom 25%), high clustering (top 5%)
threshold_degree = sorted(degrees.values())[int(0.25 * len(degrees))] if␣
↪degrees else 0
threshold_clustering = sorted(clustering.values(), reverse=True)[int(0.05 *␣
↪len(clustering))] if clustering else 0
hidden_influencers = {
n: (degrees[n], clustering[n])
for n in G.nodes()
if degrees[n] <= threshold_degree and clustering[n] >= threshold_clustering
}
# Analyze their page types
influencer_types = [G.nodes[n]['page_type'] for n in hidden_influencers.keys()]
# Visualization
plt.figure(figsize=(8, 5))
plt.hist(influencer_types, bins=len(set(influencer_types)), color='lightgreen')
plt.title('Page Types of Hidden Influencers')
plt.xlabel('Page Type')
plt.ylabel('Count')
plt.xticks(rotation=45)
```

```
plt.tight_layout()
plt.show()
# Print stats
print(f"Sampled nodes: {len(G.nodes())}, edges: {len(G.edges())}")
print(f"Hidden influencers found: {len(hidden_influencers)}")
```

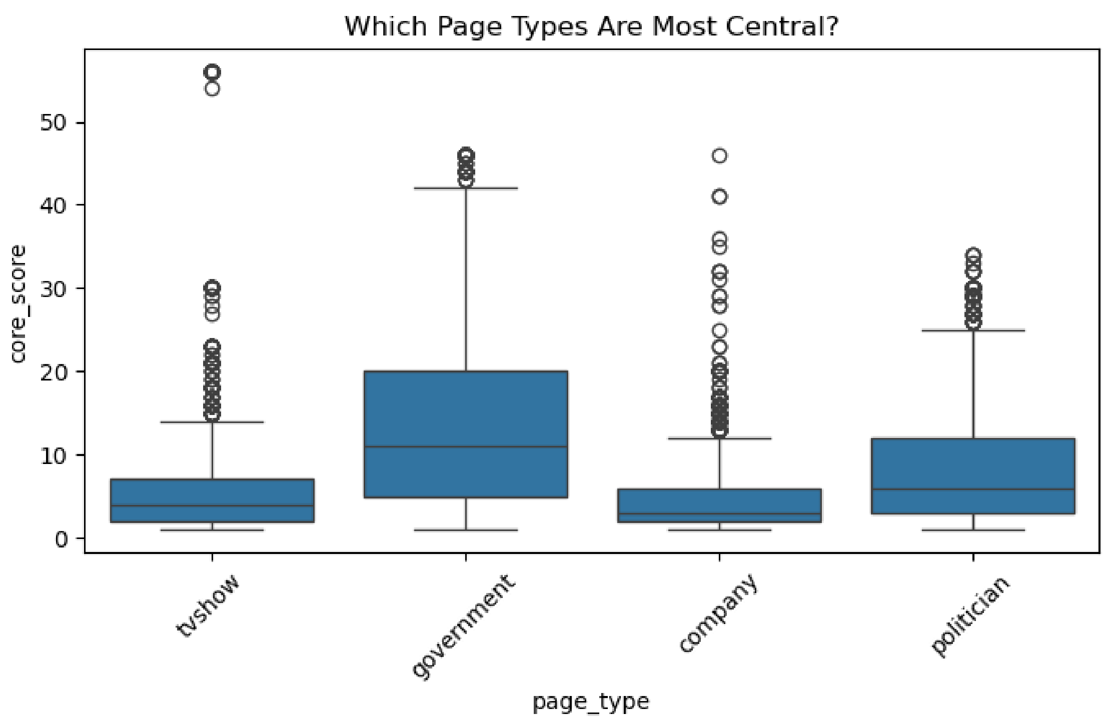



```
Sampled nodes: 22470, edges: 170823
Hidden influencers found: 1392
```

1.11 Research Question 4: Which page types are at the center vs. the edges of the network? (Simplified Core-Periphery Analysis)

[116]:
```
import seaborn as sns  # <-- Add this import

# Remove self-loops from the graph
G.remove_edges_from(nx.selfloop_edges(G))

# Calculate how "core" each node is (higher = more central)
core_scores = nx.core_number(G)

# Group by page type
type_core = nodes_df.copy()
type_core['core_score'] = type_core['id'].map(core_scores)

# Visualize
```

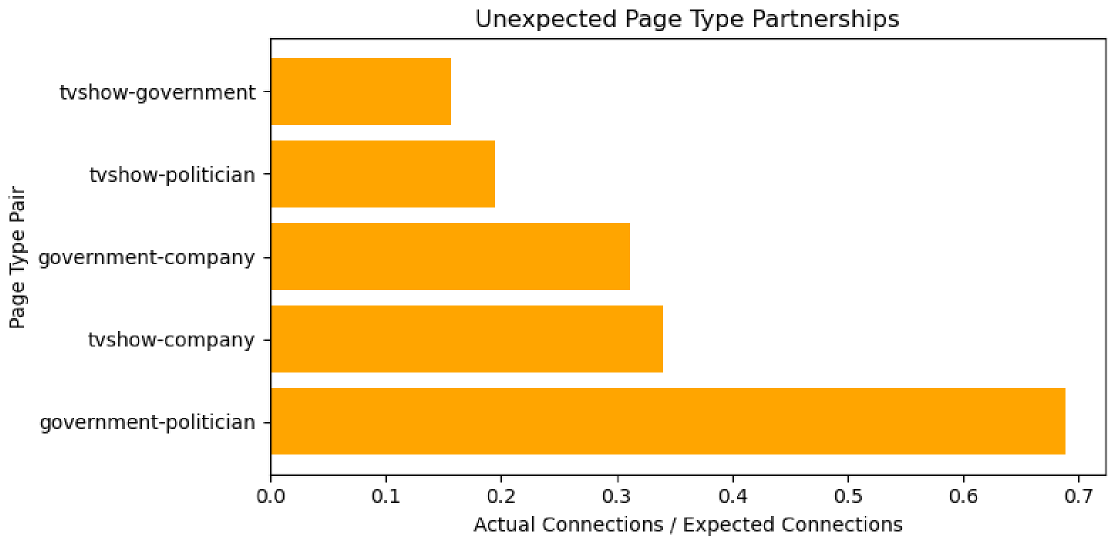
```
plt.figure(figsize=(8,4))
sns.boxplot(data=type_core, x='page_type', y='core_score')
plt.title("Which Page Types Are Most Central?")
plt.xticks(rotation=45)
plt.show()
```



1.12 Research Question 5: Which page types unexpectedly work together?

[121]:
```
from itertools import combinations
from collections import defaultdict
import pandas as pd
import matplotlib.pyplot as plt

# Assuming G and nodes_df are already loaded
# Count actual connections between types
actual_pairs = defaultdict(int)
for u, v in G.edges():
    pair = tuple(sorted([G.nodes[u]['page_type'], G.nodes[v]['page_type']]))
    actual_pairs[pair] += 1

# Find most over-connected pairs
all_types = nodes_df['page_type'].unique()
```

```
pair_df = pd.DataFrame(list(combinations(all_types, 2)), columns=['Type1',␣
↪'Type2'])
pair_df['Actual'] = pair_df.apply(lambda x: actual_pairs.get(tuple(sorted([x.
↪Type1, x.Type2])), 0), axis=1)

# Calculate expected (if connections were random)
total_edges = len(G.edges())
type_counts = nodes_df['page_type'].value_counts()
pair_df['Expected'] = pair_df.apply(lambda x: (type_counts[x.Type1]/len(G)) *␣
↪(type_counts[x.Type2]/len(G)) * total_edges, axis=1)

# Calculate surprise
pair_df['Surprise'] = pair_df['Actual'] / pair_df['Expected']

# Create a new column for pair labels
pair_df['Pair'] = pair_df['Type1'] + '-' + pair_df['Type2']

# Sort by surprise and select top 5
top_pairs = pair_df.nlargest(5, 'Surprise')

# Visualize with full pair labels
plt.figure(figsize=(8, 4))
plt.barh(top_pairs['Pair'], top_pairs['Surprise'], color='orange')
plt.title('Unexpected Page Type Partnerships')
plt.xlabel('Actual Connections / Expected Connections')
plt.ylabel('Page Type Pair')
plt.tight_layout()
plt.show()

# Print the top pairs for reference
print("Top 5 Unexpected Page Type Partnerships:")
print(top_pairs[['Pair', 'Actual', 'Expected', 'Surprise']])
```

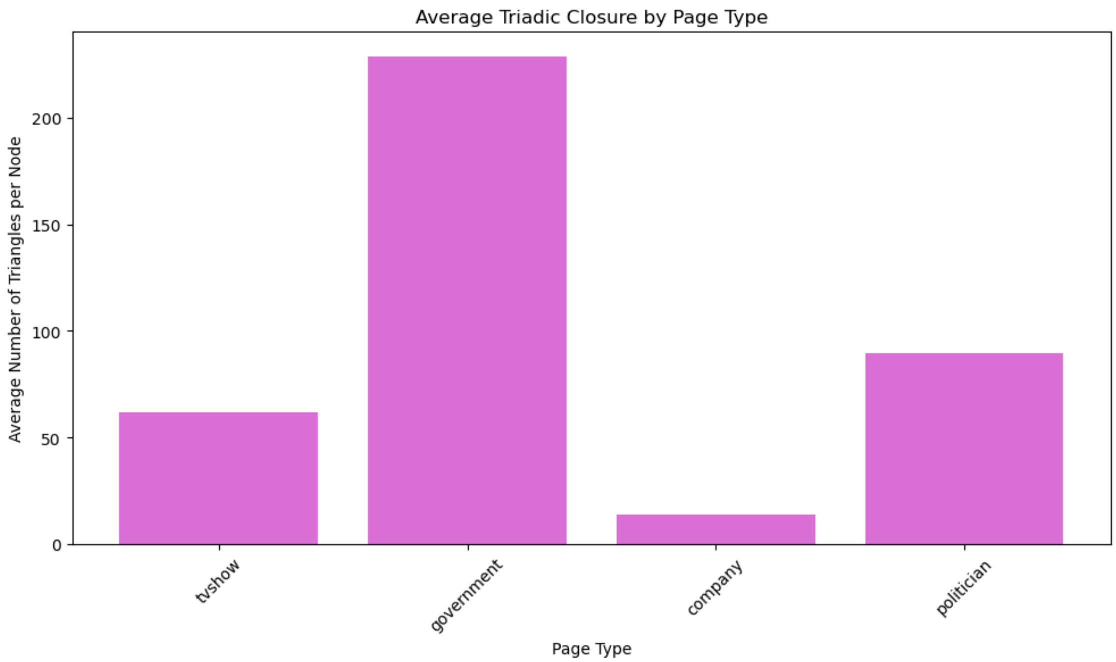



```
Top 5 Unexpected Page Type Partnerships:
                    Pair  Actual      Expected  Surprise
4  government-politician    9245  13426.224038  0.688578
1         tvshow-company    2484   7310.922096  0.339766
3     government-company    4707  15118.468295  0.311341
2      tvshow-politician    1268   6492.594095  0.195299
0      tvshow-government    1208   7744.286993  0.155986
```

1.13 Research Question 6: How do triadic closure tendencies differ across page types in the network?

[90]:
```
import matplotlib.pyplot as plt

# Calculate triadic closure per page type
page_types = nodes_df['page_type'].unique()
triadic_scores = {}
for ptype in page_types:
    nodes = [n for n, attr in G.nodes(data=True) if attr['page_type'] == ptype]
    triangles = sum(nx.triangles(G, n) for n in nodes) / len(nodes) if nodes␣
 ↪else 0
    triadic_scores[ptype] = triangles

# Visualization
plt.figure(figsize=(10, 6))
plt.bar(triadic_scores.keys(), triadic_scores.values(), color='orchid')
plt.title('Average Triadic Closure by Page Type')
plt.xlabel('Page Type')
plt.ylabel('Average Number of Triangles per Node')
plt.xticks(rotation=45)
```

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




```
[ ]:
```