HW 3 Report

Maurice Hanisch

January 23, 2026

AI Declaration

Most of the code and report text was written with use of AI tools. The ideas and analyses are my own.

1 Coding + Data Analysis

1.1 Heavy vs. light

Part (a): Law of Large Numbers

Plots of S_n vs. n for the Normal, Weibull, and Pareto distributions.

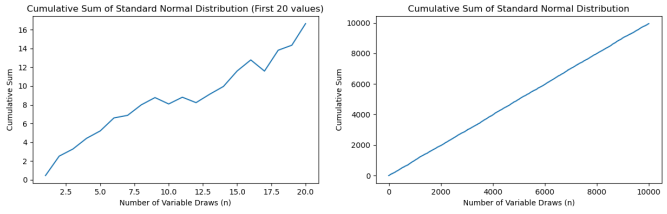


Figure 1: Partial sums for Normal Distribution ($\mu = 1, \sigma^2 = 1$). Left: First 20 samples. Right: All 10000 samples.

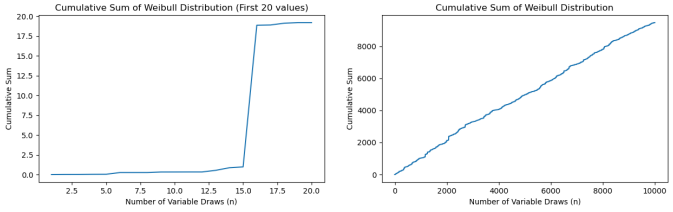


Figure 2: Partial sums for Weibull Distribution ($\alpha = 0.3, \mu = 1$). Left: First 20 samples. Right: All 10000 samples.

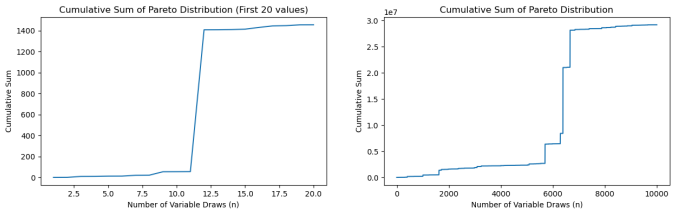


Figure 3: Partial sums for Pareto Distribution ($\alpha = 0.5, x_L = 1/3$). Left: First 20 samples. Right: All 10000 samples.

Interpretation The Normal distribution (light-tailed) has a linear plot of S_n vs n because it has finite mean and variance. The cumulative sum grows steadily with slope approximately 1.

The Weibull distribution ($\alpha = 0.3$) is heavy-tailed but has finite mean and variance. Despite more variability than the Normal distribution, the plot remains approximately linear, showing that the Law of Large Numbers holds.

The Pareto distribution ($\alpha = 0.5$) has infinite mean, so the Law of Large Numbers fails. The plot is dominated by rare, extreme jumps. Single large values dwarf the sum of many smaller values, creating a staircase pattern rather than steady growth. The plot looks similar at both small ($n = 20$) and large ($n = 10000$) scales, reflecting the scale-invariant nature of the distribution.

Part (b): Central Limit Theorem

Plot of $\frac{S_n - n\mathbb{E}[X]}{\sqrt{n}}$ vs. n for Normal and Weibull distributions.

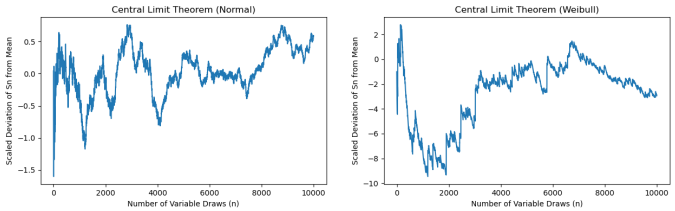


Figure 4: Scaled deviation of sum from mean ($\frac{S_n - n\mu}{\sqrt{n}}$) for Normal and Weibull distributions.

Interpretation The Central Limit Theorem states that for i.i.d. samples with mean μ and finite variance σ^2, the quantity $\frac{S_n - n\mu}{\sqrt{n}}$ converges to $N(0, \sigma^2)$ as $n \to \infty$. The deviations stabilize around zero but with variance σ^2.

For the **Normal distribution** ($\mu = 1, \sigma^2 = 1$), the scaled deviation fluctuates within a relatively constant band (roughly ± 3), consistent with $N(0, 1)$.

For the **Weibull distribution** ($\alpha = 0.3$), the CLT applies because the variance is finite. Calculating the variance:

$$\text{Var}(X) = \beta^2[\Gamma(1 + 2/\alpha) - \Gamma(1 + 1/\alpha)^2] \approx 29.24$$

with standard deviation $\sigma \approx 5.41$. The plot shows wider fluctuations (0 to -12) than the Normal case, consistent with the larger standard deviation. The predominantly negative values reflect the skewness of the distribution: most samples are small, so the sum lags behind $n\mu$ unless rare large values appear.

Why Pareto is not tested We cannot test the Pareto distribution because the mean is infinite ($\alpha = 0.5 \leq 1$). The term $n\mu$ is undefined, making the scaled deviation calculation impossible.

Part (c): The 80-20 Rule

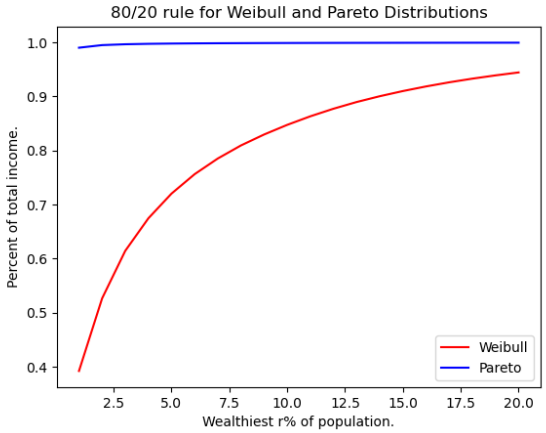
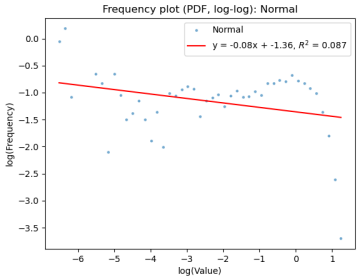


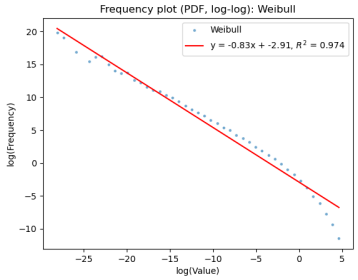
Figure 5: Fraction of total income held by the wealthiest $r\%$ of the population for Weibull and Pareto distributions.

Interpretation Pareto exhibits extreme wealth concentration: the wealthiest 2.5% holds nearly 100% of income. In contrast, the wealthiest 10% of Weibull holds about 80%. This difference is due to infinite variance in Pareto. The infinite variance allows rare but extraordinarily large values that dominate wealth distribution. Weibull, while heavy-tailed, has finite variance that constrains inequality.

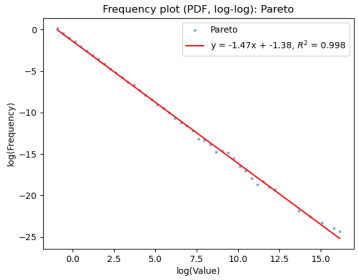
Part (d): Identifying Heavy Tails



(a) Normal: $R^2 = 0.039$



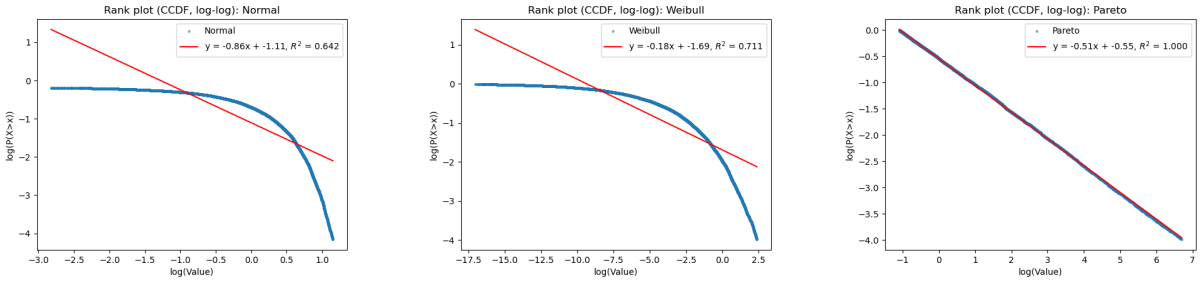
(b) Weibull: $R^2 = 0.975$, slope -0.83



(c) Pareto: $R^2 = 0.997$, slope -1.48

Figure 6: Frequency plots (PDF) on log-log scale for all three distributions.

Frequency Plots (PDF on log-log scale)



(a) Normal: $R^2 = 0.632$ (b) Weibull: $R^2 = 0.716$, slope -0.18 (c) Pareto: $R^2 = 1.000$, slope -0.50

Figure 7: Rank plots (CCDF) on log-log scale for all three distributions.

Rank Plots (CCDF on log-log scale)

Analysis Light-tailed (Normal): The frequency and rank plots are not linear on log-log scales. Normal decay is exponential (Gaussian), which curves on log-log scales. Both plots show poor linear fits ($R^2 = 0.039$ for frequency, $R^2 = 0.632$ for rank). This is expected for light-tailed distributions.

Heavy-tailed (Weibull): The frequency plot shows better linearity ($R^2 = 0.975$) with slope ≈ -0.83. The rank plot shows modest fit ($R^2 = 0.716$). Weibull's frequency plot approximates power-law behavior despite finite moments.

Very heavy-tailed (Pareto): Both plots show nearly perfect linearity. Frequency plot: $R^2 = 0.997$, slope -1.48. Rank plot: $R^2 = 1.000$, slope -0.50 (theoretical slope for $\alpha = 0.5$). Power-law distributions manifest as straight lines on log-log plots.

Light-tailed distributions do not appear linear on log-log plots. Heavy-tailed distributions show much better fits, with fit quality correlating to tail heaviness. Pareto's near-perfect fit ($R^2 \approx 1.0$) versus Weibull's good fit ($R^2 \approx 0.98$) shows that these plots effectively diagnose power laws.

The rank plot (CCDF) is more reliable than the frequency plot for identifying power laws. The Pareto rank plot shows perfect linearity while Weibull's shows curvature.

Outlier filtering removes noise from extreme values where few samples exist. By removing points beyond 3 standard deviations in log-space, we focus on the stable region of the power law and avoid noise at the extreme tail.

The complete implementation for Problem 1.1 is provided in Appendix 2.

1.2 The Devil is in the Details

Part (a): Preferential Attachment Model

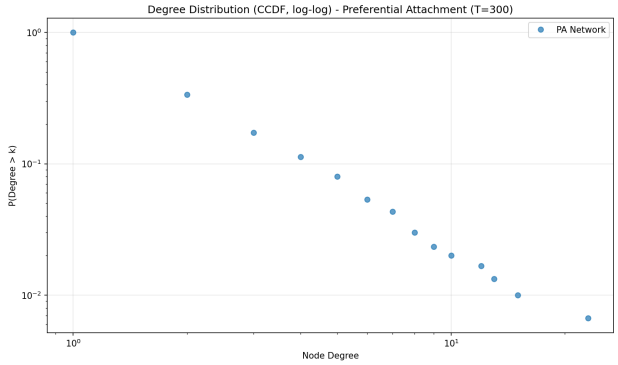


Figure 8: Degree distribution (CCDF, log-log) for preferential attachment network with $T = 300$ nodes.

The preferential attachment model generates a scale-free network. New nodes connect to existing nodes with probability proportional to their current degree, naturally producing a power-law degree distribution visible as approximately linear on the log-log CCDF plot.

Part (b): Configuration Model

Part (c): Comparison of Models

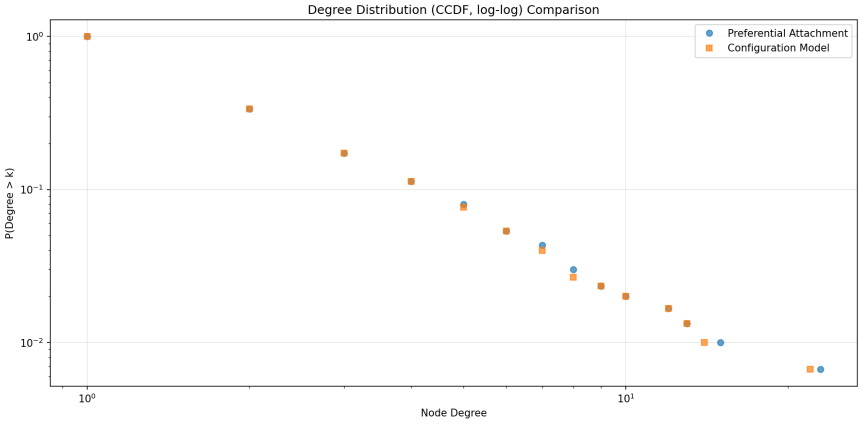


Figure 9: CCDF comparison of degree distributions. Both models have identical degree sequences by construction.

Degree Distribution Comparison

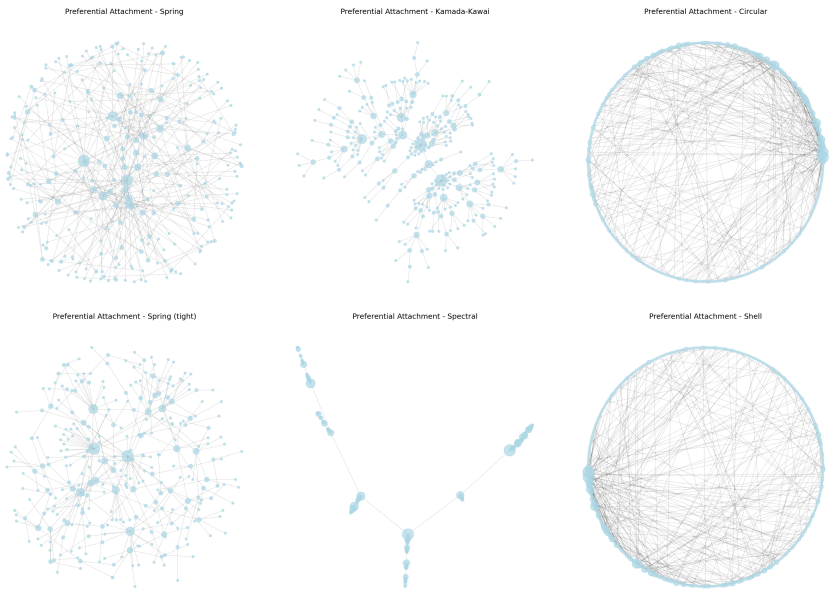


Figure 10: Preferential Attachment network with six layout algorithms.

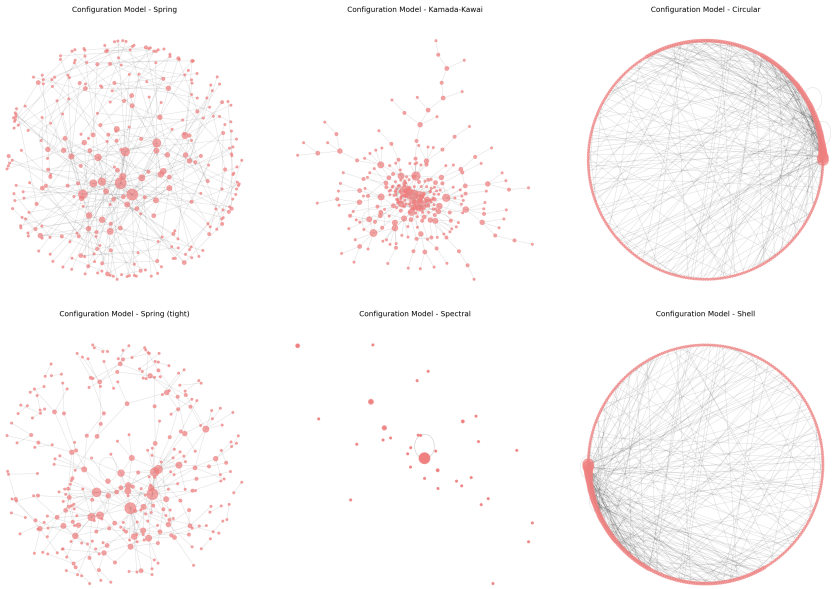


Figure 11: Configuration Model network with same six layout algorithms.

Network Structure Visualization - Layout Exploration

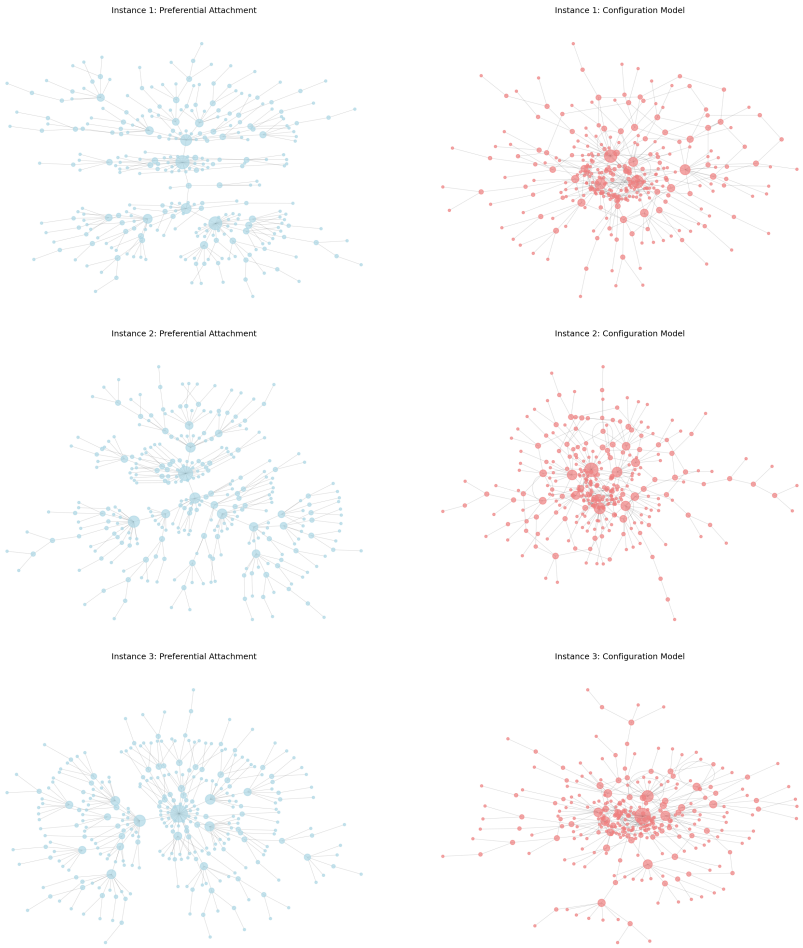


Figure 12: Multiple instances: PA (left, blue) vs CM (right, red) with Kamada-Kawai layout.

Multiple Instances Comparison - Kamada-Kawai Layout

Analysis Despite identical degree sequences, Preferential Attachment (PA) and Configuration Model (CM) networks show dramatically different structures:

Preferential Attachment: PA networks display ordered, hierarchical tree-like organization. The visualizations show a clear central core of highly connected hubs with progressively lower-degree nodes arranged hierarchically around them. High-degree nodes preferentially connect to each other, creating a natural social hierarchy. The Kamada-Kawai layout reveals this structure most clearly.

Configuration Model: CM networks show random, disorganized structure despite identical degree distribution. Nodes scatter uniformly without obvious hierarchy. The center lacks a structured core. This is because CM randomly matches edges without any preferential attachment mechanism.

Key Insight: While both networks have the same degree sequence, their structural properties differ fundamentally:

- **Clustering:** PA has higher clustering (more triangles) due to transitivity from preferential attachment
- **Mixing patterns:** PA shows assortative mixing (high-degree nodes connect to high-degree nodes), CM is neutral
- **Connectivity:** PA is always connected, CM may fragment
- **Small-world properties:** PA has strong core-periphery structure, CM has uniform connectivity

The Kamada-Kawai layout most effectively differentiates the two models. It transforms the abstract network structure into visible geometry, making PA's hierarchical nature immediately apparent and CM's randomness equally obvious.

The complete implementation for Problem 1.2 is provided in Appendix 2.

2 Appendix: Code Listings

2.1 Problem 1.1: Heavy vs. Light Tails (p1.py)

Listing 1: Problem 1.1 Implementation

```
# %% [markdown]
# # HW 3 Problem 1.1: Heavy vs. Light Tails

# %%
import numpy as np
from scipy import stats
from scipy.special import gamma
from sklearn import linear_model
import matplotlib.pyplot as plt
import heapq

n = 10000 # Arbitrary plotting value for large n

# %% [markdown]
# ## Part a: Law of Large Numbers
#
# Make two plots of $S_n$ vs. $n$ for each of the distributions - the first plot
    over $n \\in$ {1,2, . . . ,20} and the second one over the full range of $n$.
    **Interpret your plots, in light of the law of large numbers and write your
    analysis here.**

# %%
import os

def make_graph_a(
    xs,
    ys,
    distribution="Standard Normal",
    xlabel="Number of Variable Draws (n)",
    ylabel="Cumulative Sum"):
    """
    xs: List of x values to plot
    ys: List of y values to plot
    distribution: The name of the distribution that you are plotting
    """
    title = "{0} of {1} Distribution".format(ylabel, distribution)

    plt.subplots(1, 2, figsize=(15, 4))
    plt.subplot(1, 2, 1)
    plt.plot(xs[:20], ys[:20])
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.title(title + " (First 20 values)")

    plt.subplot(1, 2, 2)
    plt.plot(xs, ys)
    plt.xlabel(xlabel)
```

```
    plt.ylabel(ylabel)
    plt.title(title)

    if not os.path.exists('../latex/figs'):
        os.makedirs('../latex/figs')

    filename = f"plot_1.1_a_{distribution.replace(' ', '_')}.png"
    plt.savefig(f"../latex/figs/{filename}")

    plt.show()

# %%
normal_draws = np.random.normal(loc=1.0, scale=1.0, size=n)

alpha_weibull = 0.3
scale_weibull = 1.0 / gamma(1.0 + 1.0/alpha_weibull)
weibull_draws = scale_weibull * np.random.weibull(a=alpha_weibull, size=n)

alpha_pareto = 0.5
xm_pareto = 1.0 / 3.0
pareto_draws = stats.pareto.rvs(b=alpha_pareto, scale=xm_pareto, size=n)

x_range = np.linspace(1, n, num=n)
normal_cumsum = np.cumsum(normal_draws)
weibull_cumsum  = np.cumsum(weibull_draws)
pareto_cumsum = np.cumsum(pareto_draws)

# %%
make_graph_a(
    x_range,
    normal_cumsum,
    distribution="Standard Normal")

# %%
make_graph_a(
    x_range,
    weibull_cumsum,
    distribution="Weibull")

# %%
make_graph_a(
    x_range,
    pareto_cumsum,
    distribution="Pareto")

# %% [markdown]
# ## Part b: Central Limit Theorem
#
# The Central Limit Theorem tells us that deviations of $S_n$ from its mean are of
    size $\sqrt{n}$. That is, $S_n \approx nE[X] + O(\sqrt{n})$. Plot $\frac{S_n -
    nE[X]}{\sqrt{n}}$ vs. $n$ for each of the distributions. **Interpret your
   plots, in light of the central limit theorem, and write your analysis here. Why
    aren't we also testing the Pareto distribution here?**

# %%
mu = 1.0

normal_clt = (normal_cumsum - x_range * mu) / np.sqrt(x_range)
weibull_clt = (weibull_cumsum - x_range * mu) / np.sqrt(x_range)
```

```
plt.subplots(1, 2, figsize=(15, 4))
plt.subplot(1, 2, 1)
plt.plot(x_range, normal_clt)
plt.xlabel("Number of Variable Draws (n)")
plt.ylabel("Scaled Deviation of Sn from Mean")
plt.title("Central Limit Theorem (Normal)")

plt.subplot(1, 2, 2)
plt.plot(x_range, weibull_clt)
plt.xlabel("Number of Variable Draws (n)")
plt.ylabel("Scaled Deviation of Sn from Mean")
plt.title("Central Limit Theorem (Weibull)")

if not os.path.exists('../latex/figs'):
    os.makedirs('../latex/figs')
plt.savefig("../latex/figs/plot_1.1_b_CLT.png")

plt.show()

# %%
# Calculate Variance for Weibull (alpha=0.3, mean=1)
alpha_w = 0.3
beta_w = 1.0 / gamma(1 + 1.0/alpha_w)
variance_weibull = beta_w**2 * (gamma(1 + 2.0/alpha_w) - gamma(1 + 1.0/alpha_w)
    **2)
std_weibull = np.sqrt(variance_weibull)

print(f"Weibull Variance: {variance_weibull:.2f}")
print(f"Weibull Std Dev: {std_weibull:.2f}")


# %% [markdown]
# ## Part c: The 80-20 rule
#
# Vilfredo Pareto was motivated to define the Pareto distribution by this
    observation: 80% of the wealth in society is held by 20% of the population.
    This is an important distinguishing feature between heavy-tailed and light-
    tailed distributions. To observe this, suppose that your samples represent the
    incomes of 10000 individuals in a city. Since some of your samples for the
    Normal distribution might be negative, ignore the case of the Normal
    distribution for this part of the problem, since a negative income doesn't make
     much sense. Compute the fraction $f(r)$ of the total income of the city held
    by the wealthiest $r$% of the population, for $r$ = 1,2,...,20. For each of the
     distributions, plot $f(r)$ vs. $r$. (preferably both functions on a single
    plot). **Interpret your plot(s) and write your analysis here**.

# %%
def get_frac_wealth(draws, r_range):
    total_wealth = np.sum(draws)
    fracs = []
    for r in r_range:
        num_top = int((r / 100.0) * len(draws))
        if num_top == 0:
            fracs.append(0)
            continue
        largest = heapq.nlargest(num_top, draws)
        fracs.append(np.sum(largest) / total_wealth)
    return fracs
```

```
rRange = np.linspace(1, 20, num=20)
weibull_largest = get_frac_wealth(weibull_draws, rRange)
pareto_largest = get_frac_wealth(pareto_draws, rRange)

plt.figure()
plt.plot(rRange, weibull_largest, 'r', label="Weibull")
plt.plot(rRange, pareto_largest, 'b', label="Pareto")
plt.xlabel("Wealthiest r% of population.")
plt.ylabel("Percent of total income.")
plt.title("80/20 rule for Weibull and Pareto Distributions")
plt.legend()

if not os.path.exists('../latex/figs'):
    os.makedirs('../latex/figs')
plt.savefig("../latex/figs/plot_1.1_c_80_20.png")

plt.show()

# %% [markdown]
# ## Part d: Identifying Heavy Tails
#
# For each of the distributions (i)-(iii), plot the frequencies and ranks of the
    10000 samples on log-log scales, using separate plots for each distribution.
    Since we are using a log-log scale, filter out all negative and zero values
    before graphing. For the frequency plots, remember to experiment with various
    binsizes and to choose one such that the plots are useful. (Note that bins aren
    't needed for the rank plot.) Then, use linear regression to fit a line through
     the points on each plot. Display the best-fit lines on the plots as well as
    the R-squared values. What do your plots tell you about identifying heavy tails
     based on frequency and rank plots? **Interpret your plot(s) and write your
    analysis here**.

# %%
def pdf(data, bins=50):
    '''Takes an array with random samples from a distribution,
    and creates an approximate PDF of points.
    Returns a tuple of two vectors x, y where
    y_i = P(x_i - dx/2 <= data < x_i + dx/2) / dx'''

    min_val = np.min(data)
    max_val = np.max(data)

    if min_val <= 0:
        min_val = np.min(data[data > 0])

    bins_array = np.logspace(np.log10(min_val), np.log10(max_val), bins)

    hist, bin_edges = np.histogram(data, bins=bins_array, density=True)

    x = np.sqrt(bin_edges[:-1] * bin_edges[1:])
    y = hist

    mask = y > 0
    return x[mask], y[mask]

def ccdf(data):
    '''Takes an array with random samples from a distribution,
    and creates an approximate CCDF (complementary CDF) of points.
```

```
    Returns a tuple of two vectors x, y where y_i = P(data > x_i)'''
    sorted_data = np.sort(data)
    n = len(sorted_data)

    y = np.arange(n, 0, -1) / n
    x = sorted_data

    return x, y

def keep_positive(data_list):
    '''Takes a LIST of tuples (x, y), and filters out
    negative and zero entries (in both x and y) in each tuple.'''
    cleaned_data = []
    for x, y in data_list:
        mask = (x > 0) & (y > 0)
        cleaned_data.append((x[mask], y[mask]))
    return cleaned_data

def non_outliers(x, m):
    '''Takes an array x of data and an integer m,
    and returns a boolean array indicating whether each
    value is within m standard deviations of the mean.'''
    mu = np.mean(x)
    sigma = np.std(x)
    return np.abs(x - mu) < m * sigma

def reject_outliers(data, m=2):
    '''Takes a tuple (x, y) where x and y are log-transformed arrays
    and removes outliers using m-sigma filtering.'''
    x, y = data

    mask_x = non_outliers(x, m)
    mask_y = non_outliers(y, m)
    mask = mask_x & mask_y
    return (x[mask], y[mask])

def linear_regression(X, y):
    '''Fits a linear model y = mX + b and returns (m, b, r^2).'''

    model = linear_model.LinearRegression()
    X_reshaped = X.reshape(-1, 1)
    model.fit(X_reshaped, y)

    m = model.coef_[0]
    b = model.intercept_
    r2 = model.score(X_reshaped, y)

    return m, b, r2

def make_graphs_d(data, title, labels, ylabel='', xlabel='', filename_suffix=''):
    """Create log-log plots with best-fit lines for each distribution."""
    for i, ((X, y), label) in enumerate(zip(data, labels)):

        m, b, r2 = linear_regression(X, y)

        plt.figure()
        plt.scatter(X, y, label=label, s=5, alpha=0.5)
        plt.plot(X, b + m * X, 'r-', label=f'y = {m:.2f}x + {b:.2f}, $R^2$ = {r2
            :.3f}')
```

```
        plt.title(f"{title}: {label}")
        plt.xlabel(xlabel)
        plt.ylabel(ylabel)
        plt.legend()

        if not os.path.exists('../latex/figs'):
            os.makedirs('../latex/figs')

        fname = f"plot_1.1_d_{filename_suffix}_{label.replace(' ', '_')}.png"
        plt.savefig(f"../latex/figs/{fname}")
        plt.show()

# %%
Xi = [normal_draws, weibull_draws, pareto_draws]
names = ["Normal", "Weibull", "Pareto"]

data_pdf = [pdf(Xi[i], bins=50) for i in range(3)]
data_pdf = keep_positive(data_pdf)

data_pdf_log = [(np.log(X), np.log(y)) for (X, y) in data_pdf]
data_pdf_clean = [reject_outliers(d, m=3) for d in data_pdf_log]

make_graphs_d(data_pdf_clean, 'Frequency plot (PDF, log-log)', names,
              ylabel='log(Frequency)', xlabel='log(Value)', filename_suffix='
                  frequency')

# %%
data_ccdf = [ccdf(Xi[i]) for i in range(3)]
data_ccdf = keep_positive(data_ccdf)

data_ccdf_log = [(np.log(X), np.log(y)) for (X, y) in data_ccdf]
data_ccdf_clean = [reject_outliers(d, m=3) for d in data_ccdf_log]

make_graphs_d(data_ccdf_clean, 'Rank plot (CCDF, log-log)', names,
              ylabel='log(P(X>x))', xlabel='log(Value)', filename_suffix='rank')



# %%
```

2.2 Problem 1.2: The Devil is in the Details (p2.py)

Listing 2: Problem 1.2 Implementation

```
# %% [markdown]|
# # HW 3 Problem 1.2: The Devil is in the Details

# %% [markdown]
# ## Part (a): Preferential Attachment Model

# %%
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
from scipy import stats

def preferential_attachment(T):
```

```
    """
    Generate a network using the preferential attachment model.

    Parameters:
    T: int - Total number of nodes (residents)

    Returns:
    G: networkx Graph - The generated network
    degrees: list - Degrees of all nodes
    """
    G = nx.Graph()

    G.add_edge(0, 1)

    for t in range(2, T):
        current_degrees = dict(G.degree())
        total_degree = sum(current_degrees.values())

        probabilities = [current_degrees[node] / total_degree for node in G.nodes
            ()]

        new_connection = np.random.choice(list(G.nodes()), p=probabilities)

        G.add_edge(t, new_connection)

    degrees = [G.degree(node) for node in G.nodes()]

    return G, degrees

# %%
T = 300
G_pa, degrees_pa = preferential_attachment(T)

sorted_degrees = np.sort(degrees_pa)
unique_degrees, unique_indices = np.unique(sorted_degrees, return_index=True)
ccdf_unique = np.arange(len(sorted_degrees), 0, -1)[unique_indices] / len(
    sorted_degrees)

plt.figure(figsize=(10, 6))
plt.loglog(unique_degrees, ccdf_unique, 'o', markersize=6, alpha=0.7, label='PA
    Network')
plt.xlabel('Node Degree')
plt.ylabel('P(Degree > k)')
plt.title('Degree Distribution (CCDF, log-log) - Preferential Attachment (T=300)')
plt.grid(True, alpha=0.3)
plt.legend()
plt.tight_layout()
plt.savefig('../latex/figs/plot_1.2_a_pa_ccdf.png', dpi=150)
plt.show()

print(f"Preferential Attachment Network (T={T}):")
print(f"Number of nodes: {G_pa.number_of_nodes()}")
print(f"Number of edges: {G_pa.number_of_edges()}")
print(f"Average degree: {np.mean(degrees_pa):.2f}")
print(f"Max degree: {np.max(degrees_pa)}")
print(f"Min degree: {np.min(degrees_pa)}")

# %% [markdown]
# ## Part (b): Configuration Model
```

```
# %%
def configuration_model(degree_sequence):
    """
    Generate a network using the configuration model with stub-matching.

    Parameters:
    degree_sequence: list - Desired degree sequence

    Returns:
    G: networkx Graph - The generated network
    """
    n = len(degree_sequence)
    stubs = []

    for node_id, degree in enumerate(degree_sequence):
        stubs.extend([node_id] * degree)

    if len(stubs) % 2 != 0:
        return None

    np.random.shuffle(stubs)

    edges = []
    for i in range(0, len(stubs), 2):
        edges.append((stubs[i], stubs[i+1]))

    G = nx.Graph()
    G.add_nodes_from(range(n))
    G.add_edges_from(edges)

    return G

# %%
degree_sequence_pa = sorted(degrees_pa, reverse=True)

G_cm = configuration_model(degree_sequence_pa)

degrees_cm = [G_cm.degree(node) for node in G_cm.nodes()]

print(f"\nConfiguration Model Network:")
print(f"Number of nodes: {G_cm.number_of_nodes()}")
print(f"Number of edges: {G_cm.number_of_edges()}")
print(f"Average degree: {np.mean(degrees_cm):.2f}")
print(f"Max degree: {np.max(degrees_cm)}")
print(f"Min degree: {np.min(degrees_cm)}")

# %% [markdown]
# ## Comparing CCDF of Both Models

# %%
sorted_degrees_cm = np.sort(degrees_cm)
unique_degrees_cm, unique_indices_cm = np.unique(sorted_degrees_cm, return_index=
    True)
ccdf_unique_cm = np.arange(len(sorted_degrees_cm), 0, -1)[unique_indices_cm] / len
    (sorted_degrees_cm)

plt.figure(figsize=(12, 6))
```

```
plt.loglog(unique_degrees, ccdf_unique, 'o', markersize=6, alpha=0.7, label='
    Preferential Attachment')
plt.loglog(unique_degrees_cm, ccdf_unique_cm, 's', markersize=6, alpha=0.7, label=
    'Configuration Model')
plt.xlabel('Node Degree')
plt.ylabel('P(Degree > k)')
plt.title('Degree Distribution (CCDF, log-log) Comparison')
plt.grid(True, alpha=0.3)
plt.legend()
plt.tight_layout()
plt.savefig('../latex/figs/plot_1.2_ab_ccdf_comparison.png', dpi=150)
plt.show()

# %% [markdown]
# ## Part (c): Network Visualization and Comparison

# %%
fig, axes = plt.subplots(2, 3, figsize=(20, 14))

layouts = {
    'Spring': lambda G: nx.spring_layout(G, k=0.5, iterations=50, seed=42),
    'Kamada-Kawai': lambda G: nx.kamada_kawai_layout(G),
    'Circular': lambda G: nx.circular_layout(G),
    'Spring (tight)': lambda G: nx.spring_layout(G, k=0.2, iterations=50, seed=42)
        ,
    'Spectral': lambda G: nx.spectral_layout(G),
    'Shell': lambda G: nx.shell_layout(G)
}

layout_names = list(layouts.keys())

for idx, (layout_name, layout_func) in enumerate(layouts.items()):
    pos = layout_func(G_pa)
    ax = axes[idx // 3, idx % 3]

    node_sizes = [G_pa.degree(node) * 15 for node in G_pa.nodes()]

    nx.draw_networkx_nodes(G_pa, pos, node_size=node_sizes, node_color='lightblue'
        ,
                           alpha=0.7, ax=ax)
    nx.draw_networkx_edges(G_pa, pos, alpha=0.2, width=0.5, ax=ax)

    ax.set_title(f'Preferential Attachment - {layout_name}', fontsize=12)
    ax.axis('off')

plt.tight_layout()
plt.savefig('../latex/figs/plot_1.2_c_pa_layouts.png', dpi=150, bbox_inches='tight
    ')
plt.show()

# %%
fig, axes = plt.subplots(2, 3, figsize=(20, 14))

for idx, (layout_name, layout_func) in enumerate(layouts.items()):
    pos = layout_func(G_cm)
    ax = axes[idx // 3, idx % 3]

    node_sizes = [G_cm.degree(node) * 15 for node in G_cm.nodes()]
```

```
    nx.draw_networkx_nodes(G_cm, pos, node_size=node_sizes, node_color='lightcoral
        ',
                           alpha=0.7, ax=ax)
    nx.draw_networkx_edges(G_cm, pos, alpha=0.2, width=0.5, ax=ax)

    ax.set_title(f'Configuration Model - {layout_name}', fontsize=12)
    ax.axis('off')

plt.tight_layout()
plt.savefig('../latex/figs/plot_1.2_c_cm_layouts.png', dpi=150, bbox_inches='tight
    ')
plt.show()

# %% [markdown]
# ## Best Layouts - Side by Side Comparison (Spring vs Kamada-Kawai)

# %%
fig, axes = plt.subplots(1, 2, figsize=(16, 8))

pos_pa_spring = nx.spring_layout(G_pa, k=0.5, iterations=50, seed=42)
node_sizes_pa = [G_pa.degree(node) * 15 for node in G_pa.nodes()]

nx.draw_networkx_nodes(G_pa, pos_pa_spring, node_size=node_sizes_pa,
                       node_color='lightblue', alpha=0.7, ax=axes[0])
nx.draw_networkx_edges(G_pa, pos_pa_spring, alpha=0.2, width=0.5, ax=axes[0])
axes[0].set_title('Preferential Attachment (Spring Layout)', fontsize=14)
axes[0].axis('off')

pos_cm_spring = nx.spring_layout(G_cm, k=0.5, iterations=50, seed=42)
node_sizes_cm = [G_cm.degree(node) * 15 for node in G_cm.nodes()]

nx.draw_networkx_nodes(G_cm, pos_cm_spring, node_size=node_sizes_cm,
                       node_color='lightcoral', alpha=0.7, ax=axes[1])
nx.draw_networkx_edges(G_cm, pos_cm_spring, alpha=0.2, width=0.5, ax=axes[1])
axes[1].set_title('Configuration Model (Spring Layout)', fontsize=14)
axes[1].axis('off')

plt.tight_layout()
plt.savefig('../latex/figs/plot_1.2_pa_vs_cm_spring.png', dpi=150, bbox_inches='
    tight')
plt.show()

# %%
fig, axes = plt.subplots(1, 2, figsize=(16, 8))

pos_pa_kk = nx.kamada_kawai_layout(G_pa)
node_sizes_pa = [G_pa.degree(node) * 15 for node in G_pa.nodes()]

nx.draw_networkx_nodes(G_pa, pos_pa_kk, node_size=node_sizes_pa,
                       node_color='lightblue', alpha=0.7, ax=axes[0])
nx.draw_networkx_edges(G_pa, pos_pa_kk, alpha=0.2, width=0.5, ax=axes[0])
axes[0].set_title('Preferential Attachment (Kamada-Kawai Layout)', fontsize=14)
axes[0].axis('off')

pos_cm_kk = nx.kamada_kawai_layout(G_cm)
node_sizes_cm = [G_cm.degree(node) * 15 for node in G_cm.nodes()]

nx.draw_networkx_nodes(G_cm, pos_cm_kk, node_size=node_sizes_cm,
                       node_color='lightcoral', alpha=0.7, ax=axes[1])
```

```
nx.draw_networkx_edges(G_cm, pos_cm_kk, alpha=0.2, width=0.5, ax=axes[1])
axes[1].set_title('Configuration Model (Kamada-Kawai Layout)', fontsize=14)
axes[1].axis('off')

plt.tight_layout()
plt.savefig('../latex/figs/plot_1.2_pa_vs_cm_kk.png', dpi=150, bbox_inches='tight'
    )
plt.show()

# %%
print("\nNetwork Statistics Comparison:")
print("\nPreferential Attachment:")
print(f"  Clustering coefficient: {nx.average_clustering(G_pa):.4f}")
print(f"  Average shortest path length: {nx.average_shortest_path_length(G_pa):.4f
    }")
print(f"  Density: {nx.density(G_pa):.4f}")
print(f"  Is connected: {nx.is_connected(G_pa)}")

print("\nConfiguration Model:")
print(f"  Clustering coefficient: {nx.average_clustering(G_cm):.4f}")
if nx.is_connected(G_cm):
    print(f"  Average shortest path length: {nx.average_shortest_path_length(G_cm)
        :.4f}")
else:
    largest_cc = max(nx.connected_components(G_cm), key=len)
    G_cm_largest = G_cm.subgraph(largest_cc).copy()
    print(f"  Average shortest path length (largest component): {nx.
        average_shortest_path_length(G_cm_largest):.4f}")
print(f"  Density: {nx.density(G_cm):.4f}")
print(f"  Is connected: {nx.is_connected(G_cm)}")

# %% [markdown]
# ## Multiple Instances Comparison (PA vs CM, Spring Layout)

# %%
fig, axes = plt.subplots(3, 2, figsize=(18, 20))

for i in range(3):
    G_pa_inst, _ = preferential_attachment(T)
    degree_seq_inst = sorted([G_pa_inst.degree(node) for node in G_pa_inst.nodes()
        ], reverse=True)
    G_cm_inst = configuration_model(degree_seq_inst)

    pos_pa = nx.kamada_kawai_layout(G_pa_inst)
    pos_cm = nx.kamada_kawai_layout(G_cm_inst)

    node_sizes_pa = [G_pa_inst.degree(node) * 15 for node in G_pa_inst.nodes()]
    node_sizes_cm = [G_cm_inst.degree(node) * 15 for node in G_cm_inst.nodes()]

    nx.draw_networkx_nodes(G_pa_inst, pos_pa, node_size=node_sizes_pa,
                           node_color='lightblue', alpha=0.7, ax=axes[i, 0])
    nx.draw_networkx_edges(G_pa_inst, pos_pa, alpha=0.2, width=0.5, ax=axes[i, 0])
    axes[i, 0].set_title(f'Instance {i+1}: Preferential Attachment', fontsize=12)
    axes[i, 0].axis('off')

    nx.draw_networkx_nodes(G_cm_inst, pos_cm, node_size=node_sizes_cm,
                           node_color='lightcoral', alpha=0.7, ax=axes[i, 1])
    nx.draw_networkx_edges(G_cm_inst, pos_cm, alpha=0.2, width=0.5, ax=axes[i, 1])
    axes[i, 1].set_title(f'Instance {i+1}: Configuration Model', fontsize=12)
```

```
    axes[i, 1].axis('off')

plt.tight_layout()
plt.savefig('../latex/figs/plot_1.2_c_pa_vs_cm_multiple.png', dpi=150, bbox_inches
    ='tight')
plt.show()

# %%
```