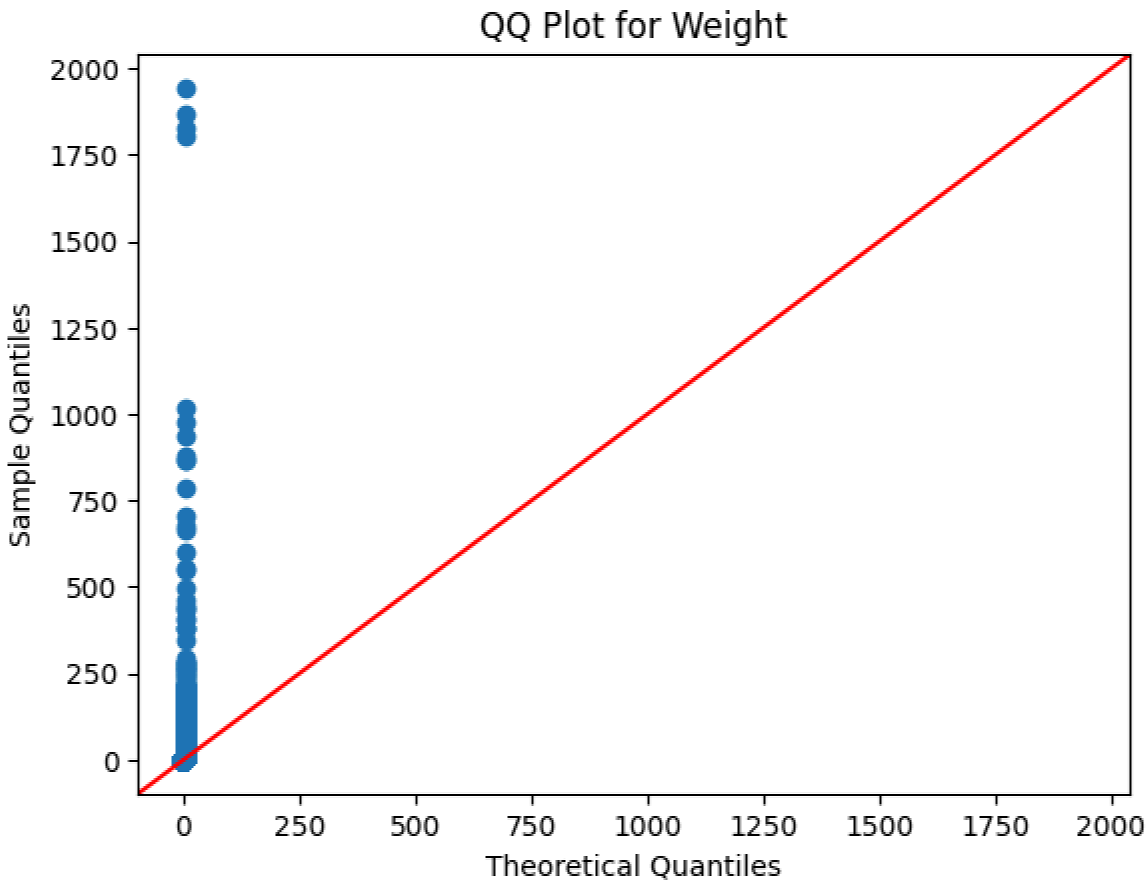
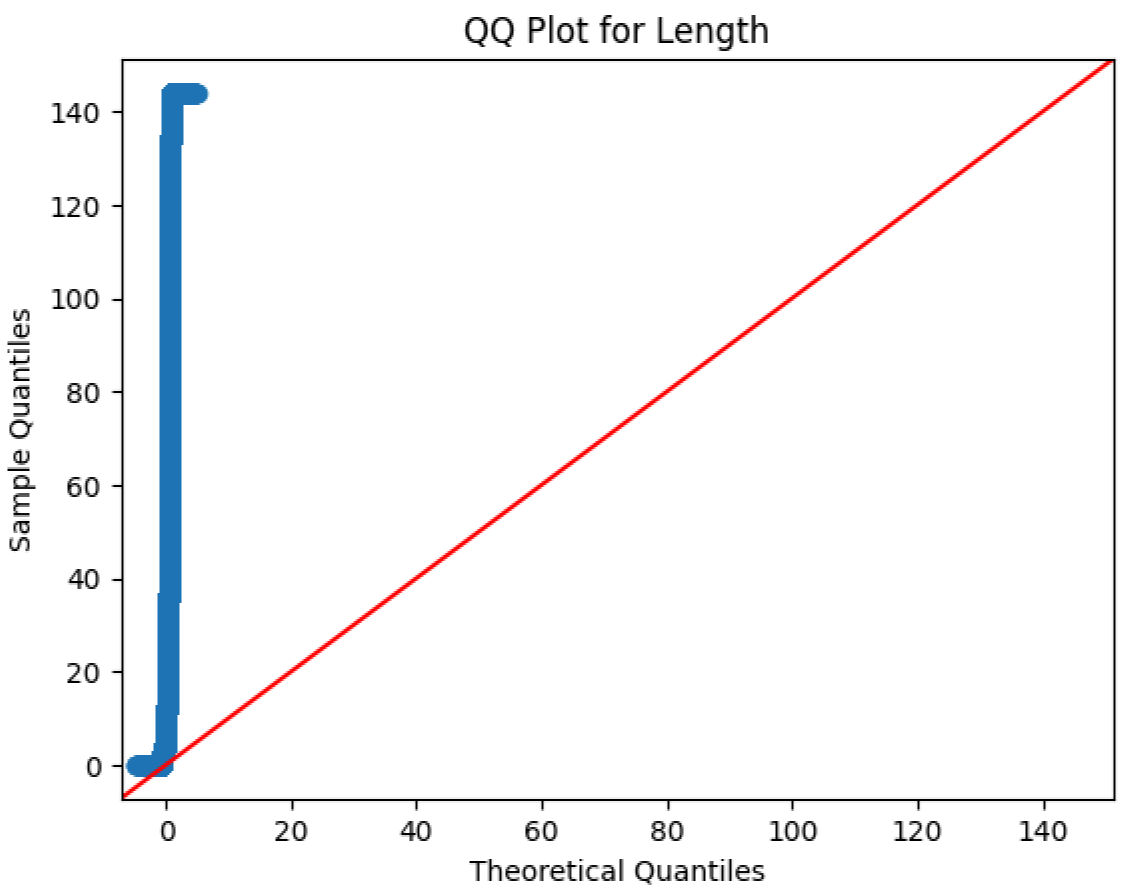
In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from datetime import datetime
import seaborn as sns
from scipy import stats
```

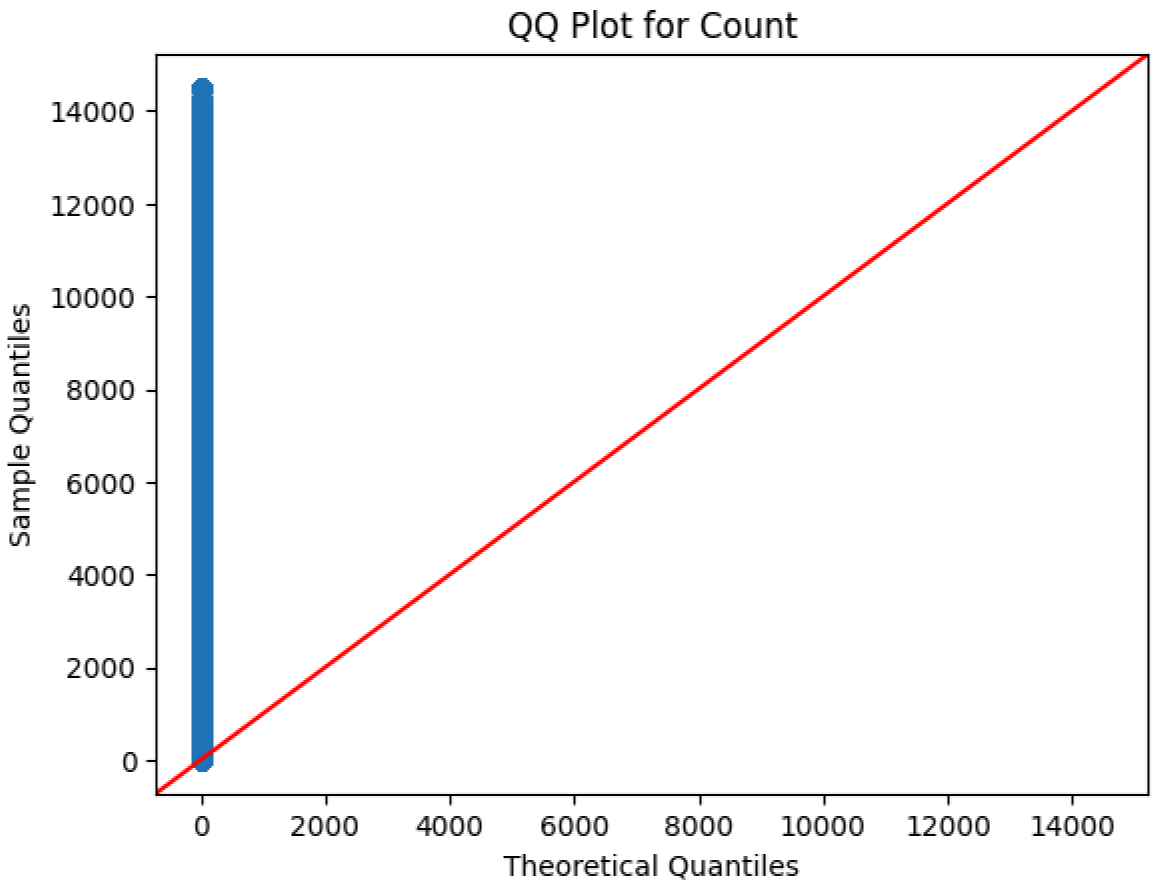
In [ ]:
```
data = pd.read_csv("BitcoinHeistData.csv")
```

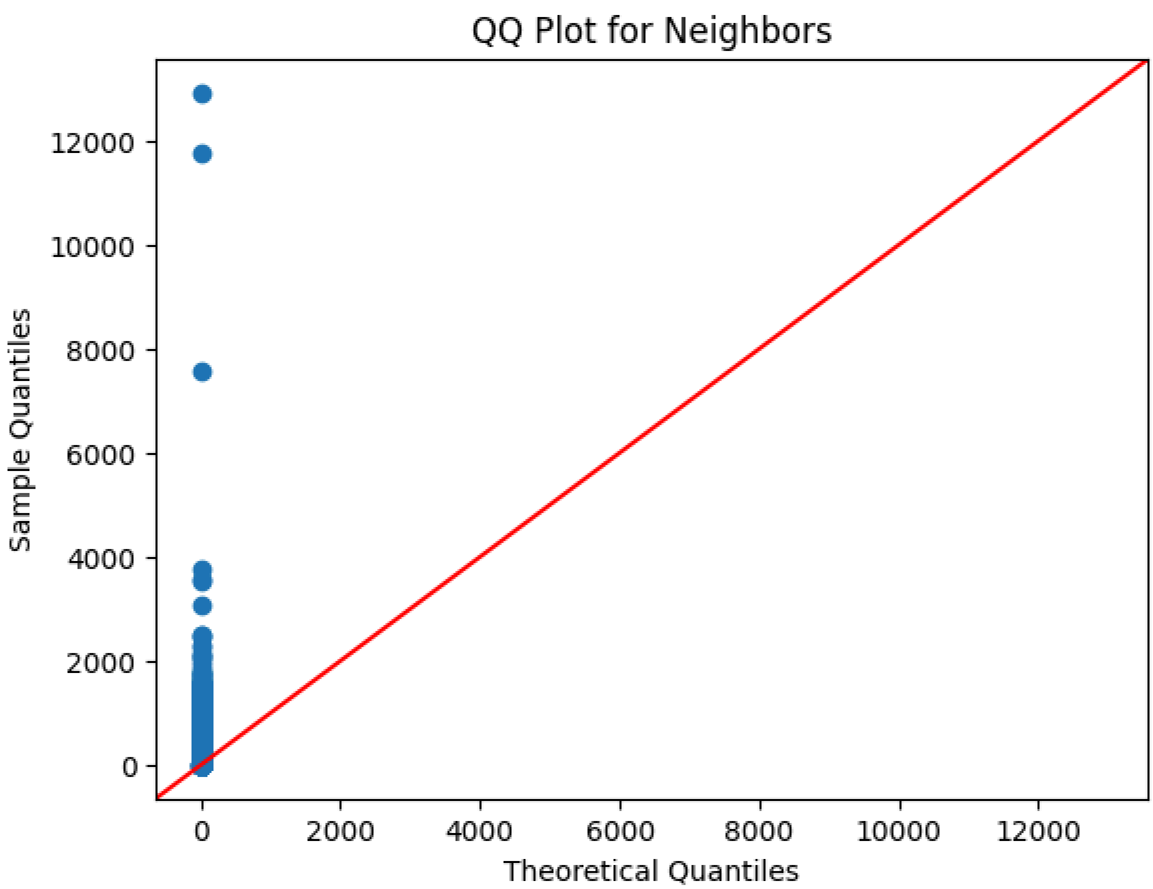
In [ ]:
```
y = data['label']
data = data[['weight', 'length', 'count', 'neighbors', 'label']]
```

In [ ]:
```
import statsmodels.api as sm
import pylab as py


sm.qqplot(data['weight'], line ='45')
py.title("QQ Plot for Weight")
py.show()

sm.qqplot(data['length'], line ='45')
py.title("QQ Plot for Length")
py.show()

sm.qqplot(data['count'], line ='45')
py.title("QQ Plot for Count")
py.show()

sm.qqplot(data['neighbors'], line ='45')
py.title("QQ Plot for Neighbors")
py.show()
```


QQ Plot for Weight
Sample Quantiles
Theoretical Quantiles
2000
1750
1500
1250
1000
750
500
250
0
0
250
500
750
1000
1250
1500
1750
2000


QQ Plot for Length
Sample Quantiles
Theoretical Quantiles
140
120
100
80
60
40
20
0
0
20
40
60
80
100
120
140





In [ ]:
```
data[['weight', 'length', 'count', 'neighbors']].describe()
```

Out[ ]:

	weight	length	count	neighbors
count	2.916697e+06	2.916697e+06	2.916697e+06	2.916697e+06
mean	5.455192e-01	4.500859e+01	7.216446e+02	2.206516e+00
std	3.674255e+00	5.898236e+01	1.689676e+03	1.791877e+01
min	3.606469e-94	0.000000e+00	1.000000e+00	1.000000e+00
25%	2.148438e-02	2.000000e+00	1.000000e+00	1.000000e+00
50%	2.500000e-01	8.000000e+00	1.000000e+00	2.000000e+00
75%	8.819482e-01	1.080000e+02	5.600000e+01	2.000000e+00
max	1.943749e+03	1.440000e+02	1.449700e+04	1.292000e+04

In [ ]:

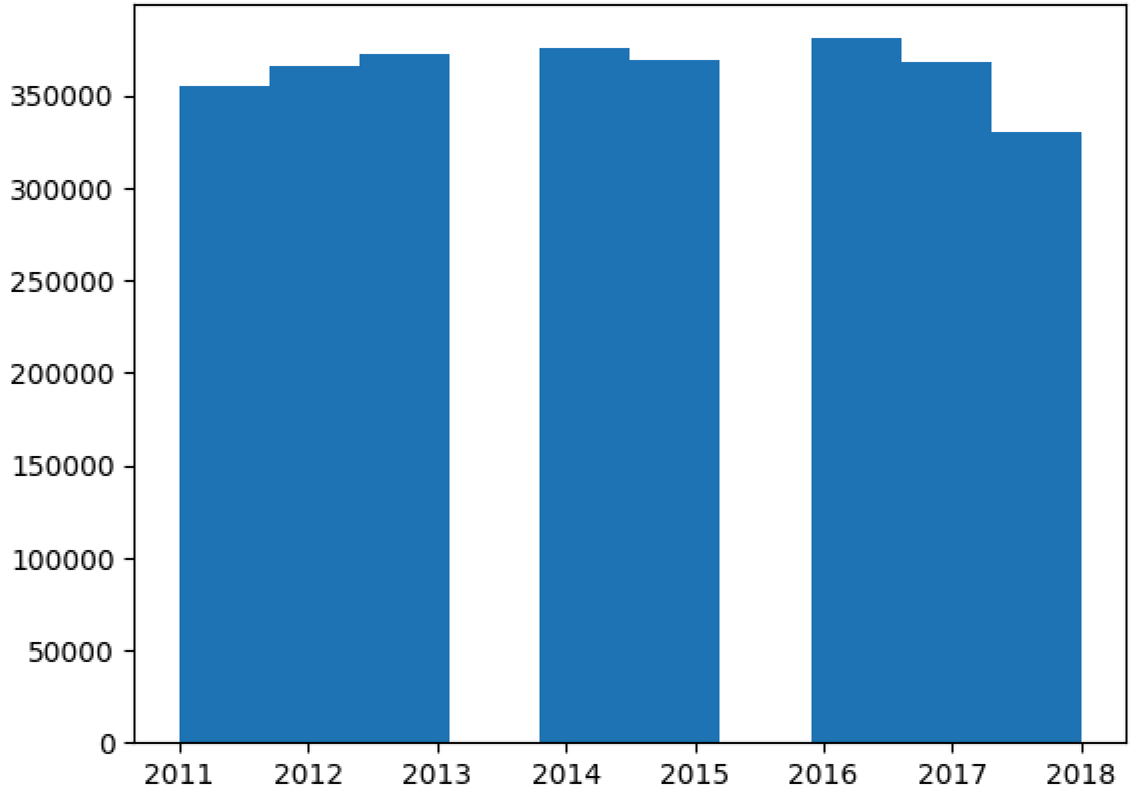
```
data.head()
```

Out[ ]:

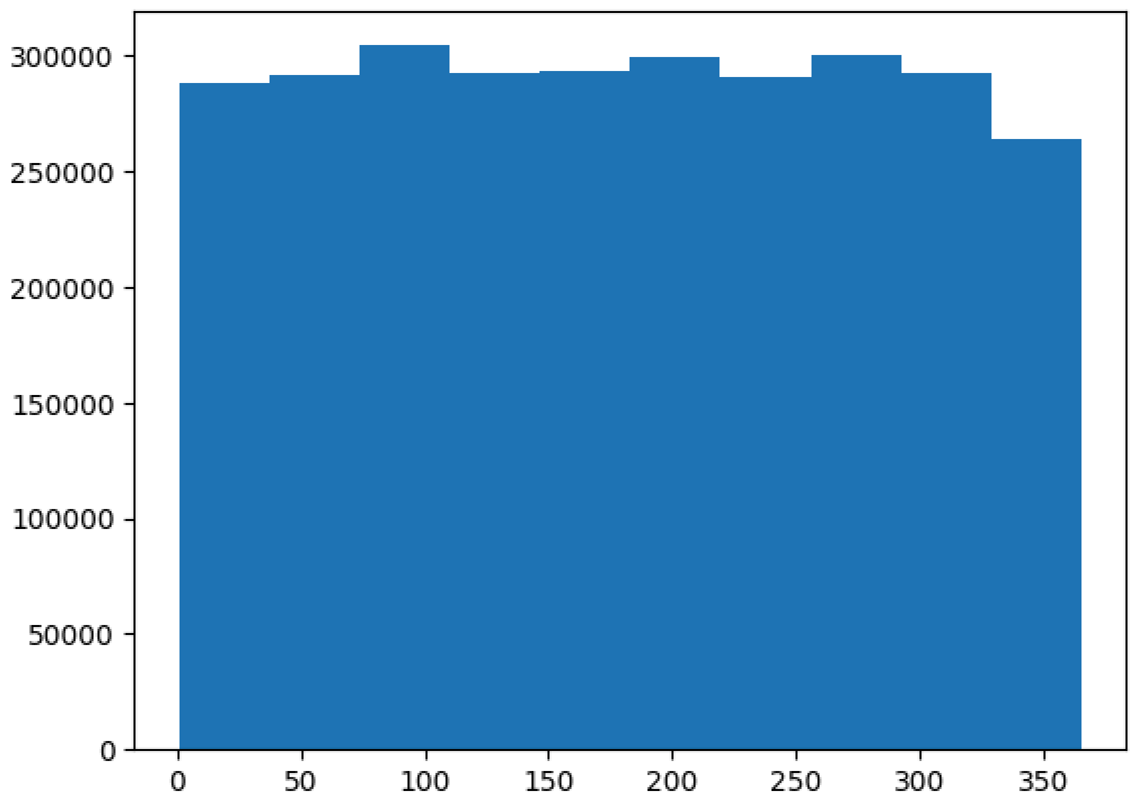
	address	year	day	length	weight	count	looped	neighbors	
0	111K8kZAEnJg245r2cM6y9zgJGHZtJPy6	2017	11	18	0.008333	1	0	2	1
1	1123pJv8jzeFQaCV4w644pzQJzVWay2zcA	2016	132	44	0.000244	1	0	1	1
2	112536im7hy6wtKbpH1qYDWtTyMRAcA2p7	2016	246	0	1.000000	1	0	2	2
3	1126eDRw2wqSkWosjTCre8cjjQW8sSeWH7	2016	322	72	0.003906	1	0	2	
4	1129TSjKtx65E35GiUo4AYVeyo48twbrGX	2016	238	144	0.072848	456	0	1	2

In [ ]:

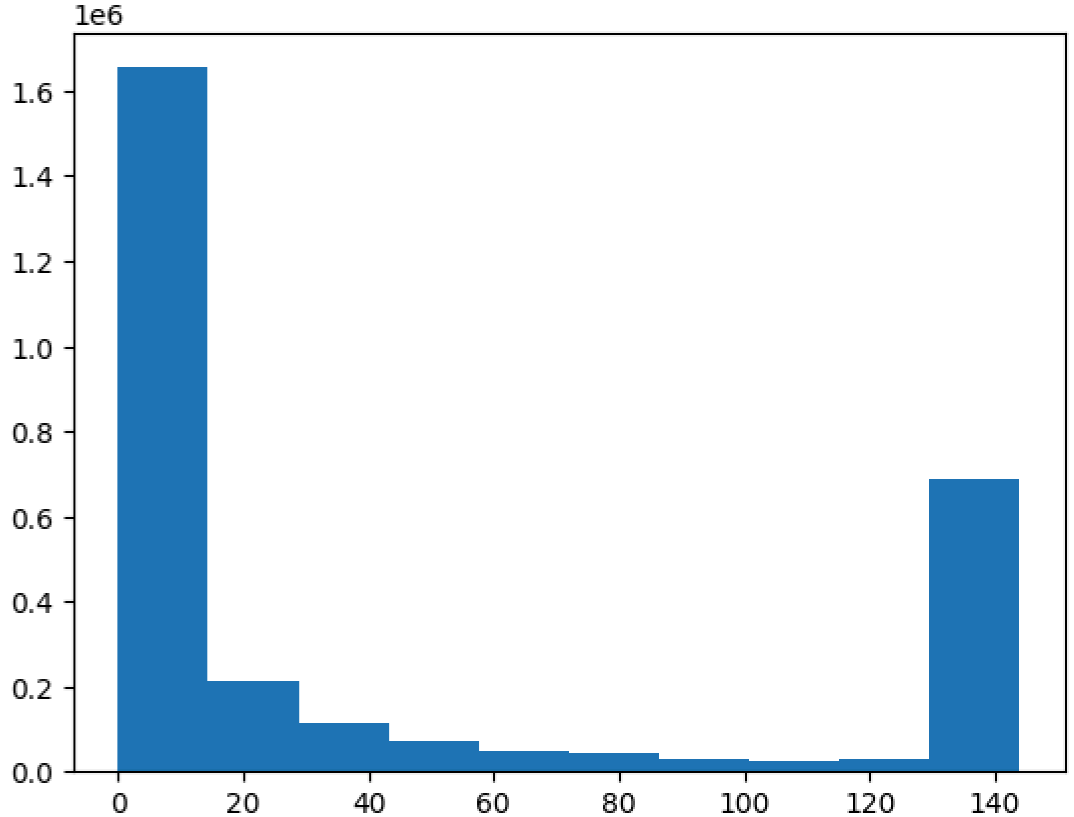
```
plt.hist(data["year"])
plt.show()
```



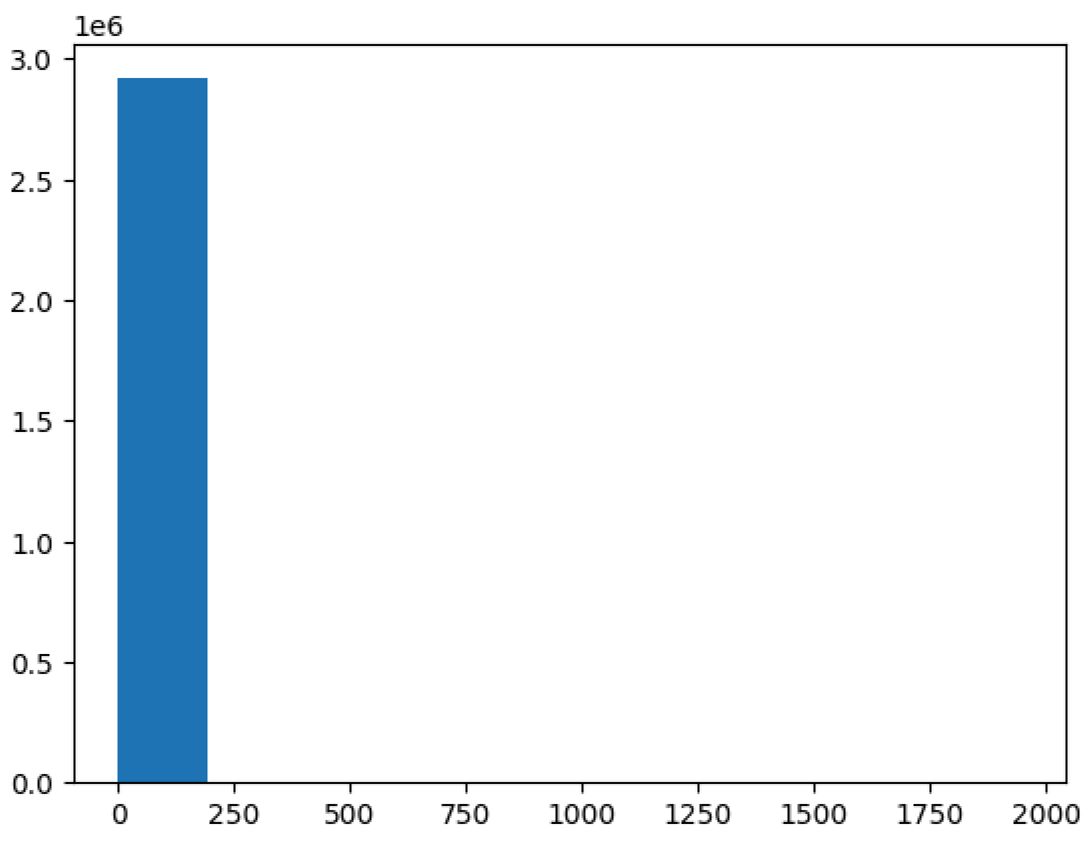
In [ ]:
```
plt.hist(data["day"])
plt.show()
```



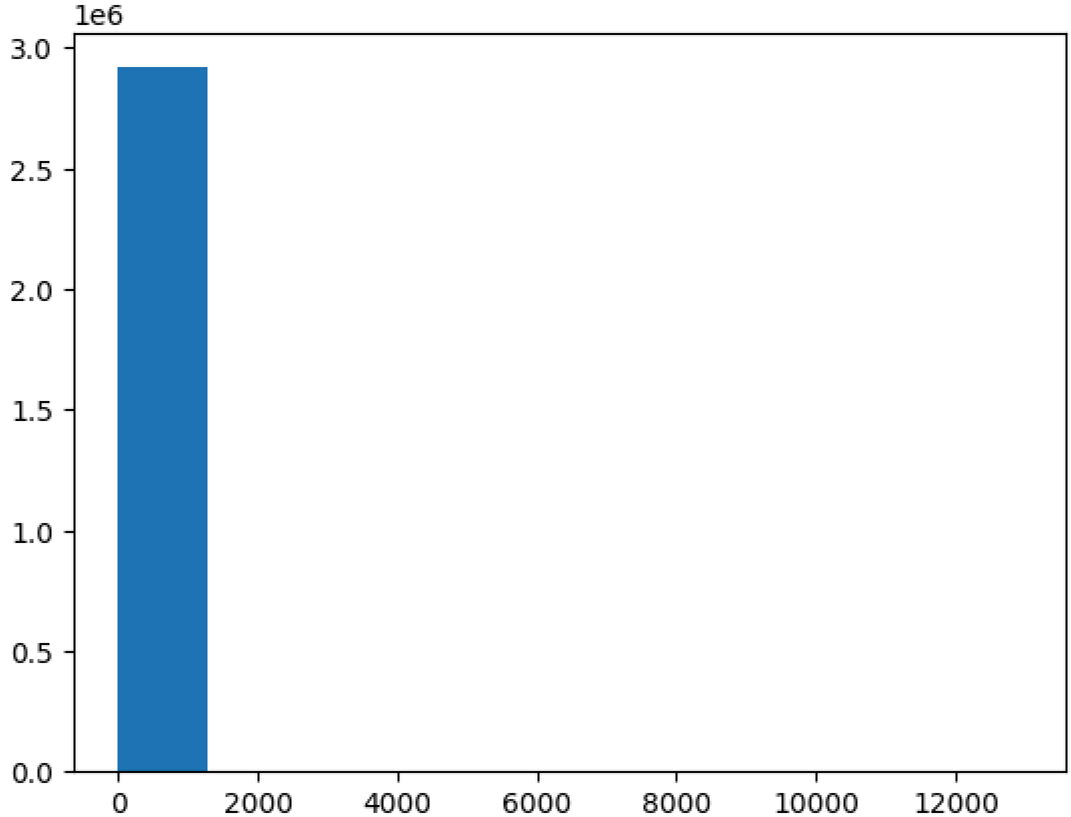
In [ ]:
```
plt.hist(data["length"])
plt.show()
```



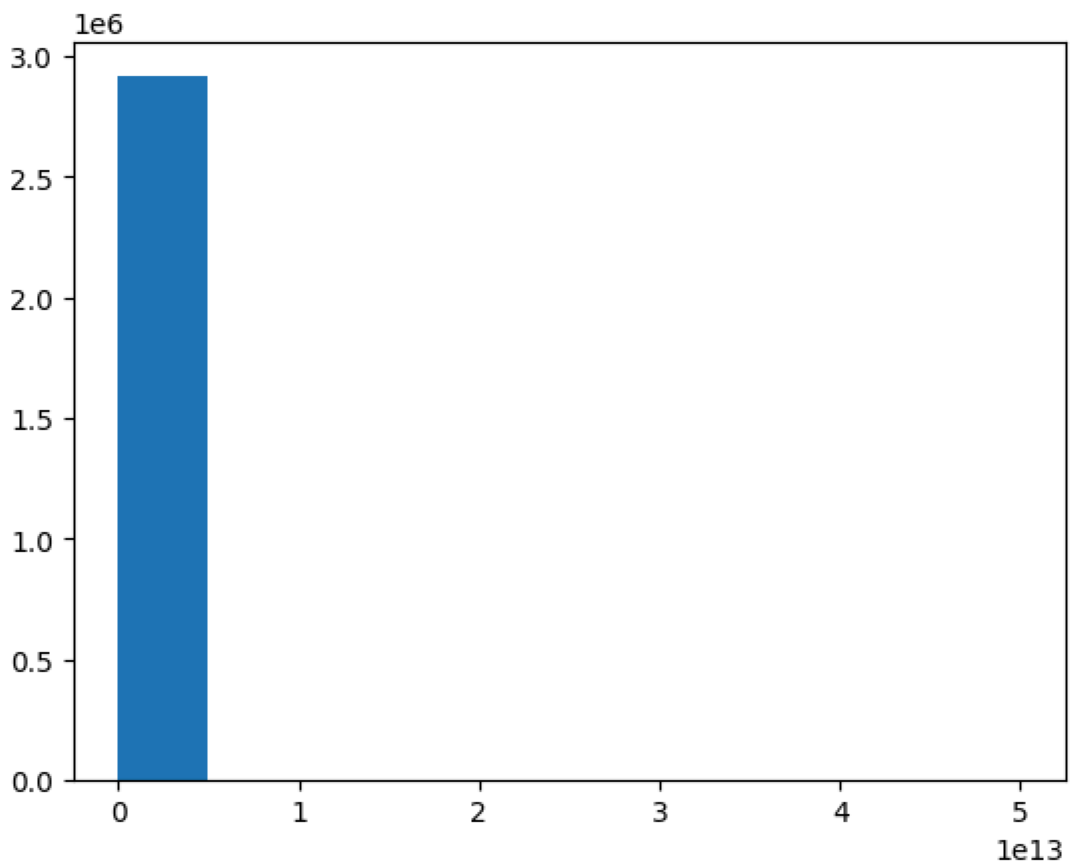
In [ ]:
```
plt.hist(data["weight"])
plt.show()
```



In [ ]:
```
plt.hist(data["neighbors"])
plt.show()
```



In [ ]:
```
plt.hist(data["income"])
plt.show()
```



In [ ]:
```
data['date'] = data['year'].astype(str) + '-' + data['day'].astype(str)

# Convert to datetime
data['date'] = pd.to_datetime(data['date'], format='%Y-%j')

print(data['date'])
```

```
0         2017-01-11
1         2016-05-11
2         2016-09-02
3         2016-11-17
4         2016-08-25
             ...
2916692   2018-11-26
2916693   2018-11-26
2916694   2018-11-26
2916695   2018-11-26
2916696   2018-11-26
Name: date, Length: 2916697, dtype: datetime64[ns]
```

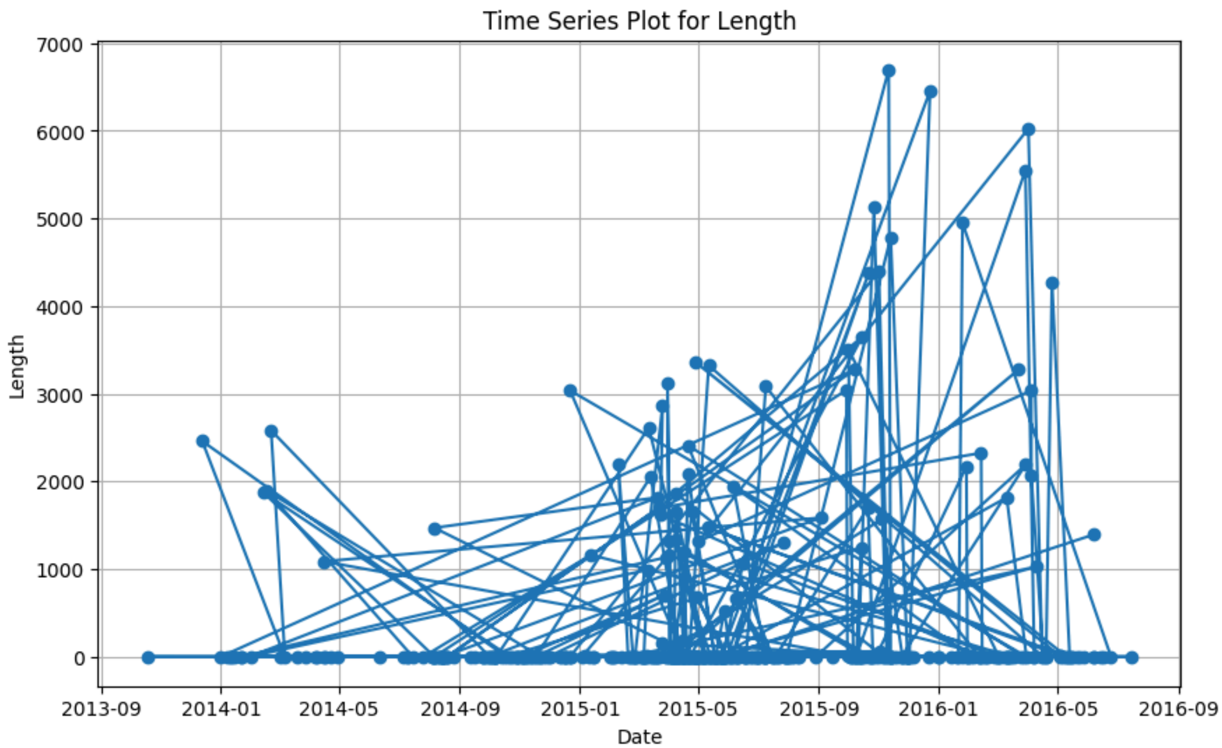
In [ ]:
```
data.set_index('date', inplace=True)
```

In [ ]:
```
print(data)
```

```
                                        address  year  day  length     weight  \
date
2017-01-11   111K8kZAEnJg245r2cM6y9zgJGHZtJPy6  2017   11      18   0.008333
2016-05-11  1123pJv8jzeFQaCV4w644pzQJzVWay2zcA  2016  132      44   0.000244
2016-09-02  112536im7hy6wtKbpH1qYDWtTyMRAcA2p7  2016  246       0   1.000000
2016-11-17  1126eDRw2wqSkWosjTCre8cjjQW8sSeWH7  2016  322      72   0.003906
2016-08-25  1129TSjKtx65E35GiUo4AYVeyo48twbrGX  2016  238     144   0.072848
...                                        ...   ...  ...     ...        ...
2018-11-26  12D3trgho1vJ4mGtWBRPyHdMJK96TRYSry  2018  330       0   0.111111
2018-11-26  1P7PputTcVkhXBmXBvSD9MJ3UYPsiou1u2  2018  330       0   1.000000
2018-11-26  1KYiKJEfdJtap9QX2v9BXJMpz2SfU4pgZw  2018  330       2  12.000000
2018-11-26  15iPUJsRNZQZHmZZVwmQ63srsmughCXV4a  2018  330       0   0.500000
2018-11-26  3LFFBxp15h9KSFtaw55np8eP5fv6kdK17e  2018  330     144   0.073972

            count  looped  neighbors        income            label
date
2017-01-11      1       0          2  1.000500e+08  princetonCerber
2016-05-11      1       0          1  1.000000e+08   princetonLocky
2016-09-02      1       0          2  2.000000e+08  princetonCerber
2016-11-17      1       0          2  7.120000e+07  princetonCerber
2016-08-25    456       0          1  2.000000e+08   princetonLocky
...           ...     ...        ...           ...              ...
2018-11-26      1       0          1  1.255809e+09            white
2018-11-26      1       0          1  4.409699e+07            white
2018-11-26      6       6         35  2.398267e+09            white
2018-11-26      1       0          1  1.780427e+08            white
2018-11-26   6800       0          2  1.123500e+08            white

[2916697 rows x 10 columns]
```

In [ ]:
```
white = data[data['label'] == 'montrealNoobCrypt']

plt.figure(figsize=(10,6))
plt.plot(white.index, white['count'], marker='o', linestyle='-')
plt.title('Time Series Plot for Length')
```

```
plt.xlabel('Date')
plt.ylabel('Length')
plt.grid(True)
plt.show()
```



In [ ]:
```
data.label.value_counts()
```

Out[ ]:
```
label
white                          2875284
paduaCryptoWall                  12390
montrealCryptoLocker              9315
princetonCerber                   9223
princetonLocky                    6625
montrealCryptXXX                  2419
montrealNoobCrypt                  483
montrealDMALockerv3                354
montrealDMALocker                  251
montrealSamSam                      62
montrealCryptoTorLocker2015         55
montrealGlobeImposter               55
montrealGlobev3                     34
montrealGlobe                       32
montrealWannaCry                    28
montrealRazy                        13
montrealAPT                         11
paduaKeRanger                       10
montrealFlyper                       9
montrealXTPLocker                    8
montrealXLockerv5.0                  7
montrealVenusLocker                  7
montrealCryptConsole                 7
montrealEDA2                         6
montrealJigSaw                       4
paduaJigsaw                          2
montrealXLocker                      1
montrealSam                          1
montrealComradeCircle                1
Name: count, dtype: int64
```

In [ ]:
```
print(count_val)
```

```
[   1    1    1 ...    6    1 6800]
```

In [ ]:
```
fig, ax = plt.subplots(7, 1,figsize=(16,16))
fig.tight_layout()

day_val = data.day.values
length_val = data.length.values
weight_val = data.weight.values
looped_val = data.looped.values
neighbors_val = data.neighbors.values
income_val = data.income.values
count_val = data['count'].values

sns.distplot(day_val, ax=ax[0])
ax[0].set_title('Day Distribution')
ax[0].set_xlim([min(day_val), max(day_val)])

sns.distplot(length_val, ax=ax[1])
ax[1].set_title('Length Distribution')
ax[1].set_xlim([min(length_val), max(length_val)])

sns.distplot(weight_val, ax=ax[2])
ax[2].set_title('Weight Distribution')
ax[2].set_xlim([min(weight_val), max(weight_val)])

sns.distplot(looped_val, ax=ax[3])
```

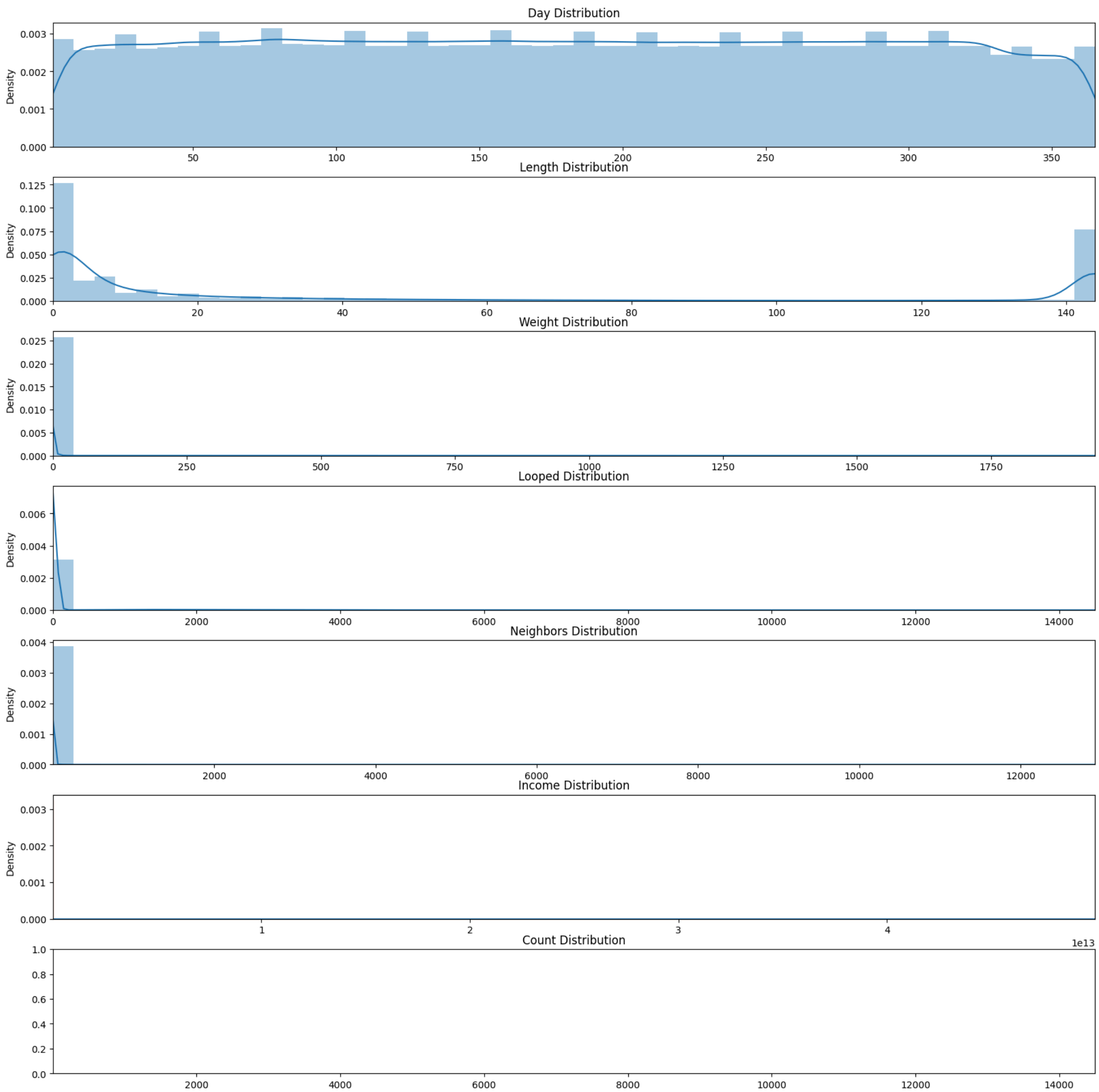
```
ax[3].set_title('Looped Distribution')
ax[3].set_xlim([min(looped_val), max(looped_val)])

sns.distplot(neighbors_val, ax=ax[4])
ax[4].set_title('Neighbors Distribution')
ax[4].set_xlim([min(neighbors_val), max(neighbors_val)])

sns.distplot(income_val, ax=ax[5])
ax[5].set_title('Income Distribution')
ax[5].set_xlim([min(income_val), max(income_val)])

sns.distplot(count_val, ax=ax[5])
ax[6].set_title('Count Distribution')
ax[6].set_xlim([min(count_val), max(count_val)])
```

```
<ipython-input-15-742b11ecaa82>:12: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(day_val, ax=ax[0])
<ipython-input-15-742b11ecaa82>:16: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(length_val, ax=ax[1])
<ipython-input-15-742b11ecaa82>:20: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(weight_val, ax=ax[2])
<ipython-input-15-742b11ecaa82>:24: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(looped_val, ax=ax[3])
<ipython-input-15-742b11ecaa82>:28: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(neighbors_val, ax=ax[4])
<ipython-input-15-742b11ecaa82>:32: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
```

```
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(income_val, ax=ax[5])
<ipython-input-15-742b11ecaa82>:36: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(count_val, ax=ax[5])
```

Out[ ]:

```
(1.0, 14497.0)
```



In [ ]:

```
print("Mean year = " + str(np.mean(data["year"])) + ", year standard deviation = " + s
```

```
Mean year = 2014.475011288454, year standard deviation = 2.2573971651095768
```

In [ ]:
```
print("Mean day = " + str(np.mean(data["day"])) + ", day standard deviation = " + str(
```

```
Mean day = 181.457211016434, day standard deviation = 104.0118179365244
```

In [ ]:
```
print("Mean length = " + str(np.mean(data["length"])) + ", length standard deviation =
```

```
Mean length = 45.00859293920486, length standard deviation = 58.98235218811939
```

In [ ]:
```
print("Mean weight = " + str(np.mean(data["weight"])) + ", weight standard deviation =
```

```
Mean weight = 0.5455192341640024, weight standard deviation = 3.6742546257308684
```

In [ ]:
```
print("Mean count = " + str(np.mean(data["count"])) + ", count standard deviation = "
```

```
Mean count = 721.6446428957139, count standard deviation = 1689.6755041726624
```

In [ ]:
```
print("Mean looped = " + str(np.mean(data["looped"])) + ", looped standard deviation =
```

```
Mean looped = 238.50669884461772, looped standard deviation = 966.3215201658936
```

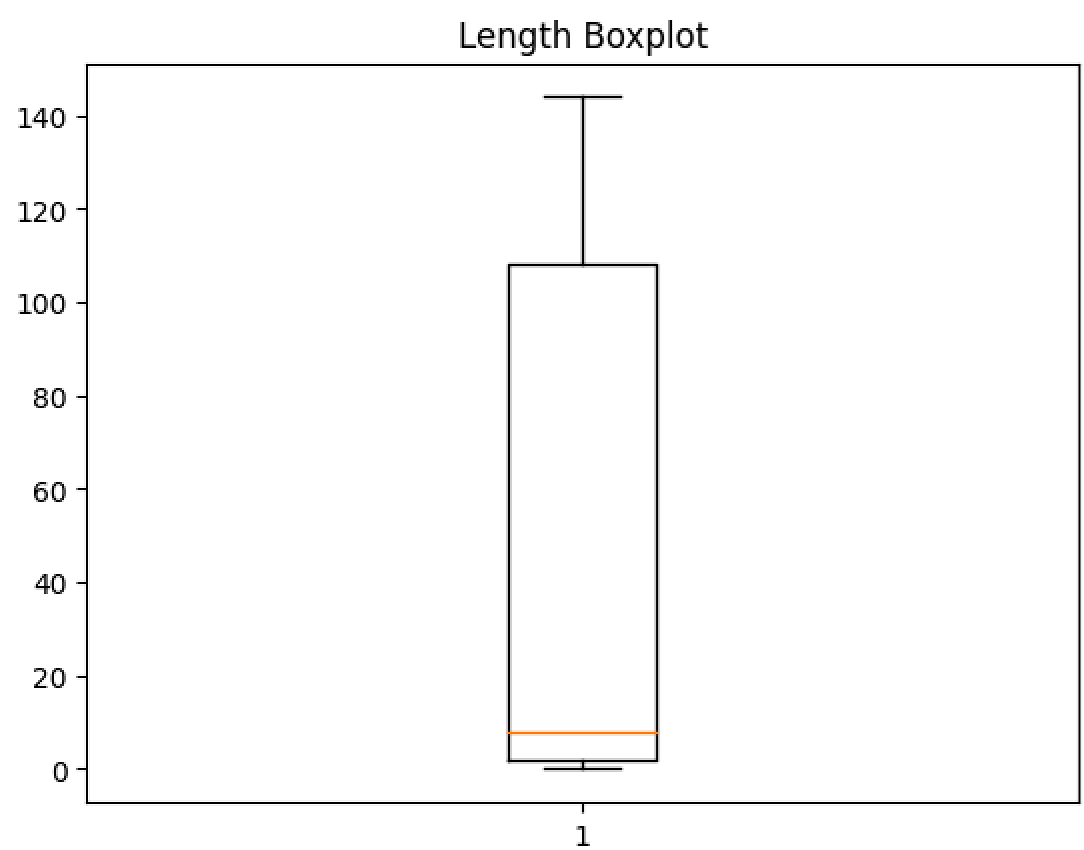
In [ ]:
```
print("Mean neighbors = " + str(np.mean(data["neighbors"])) + ", neighbors standard de
```

```
Mean neighbors = 2.206516137946451, neighbors standard deviation = 17.918762006490027
```

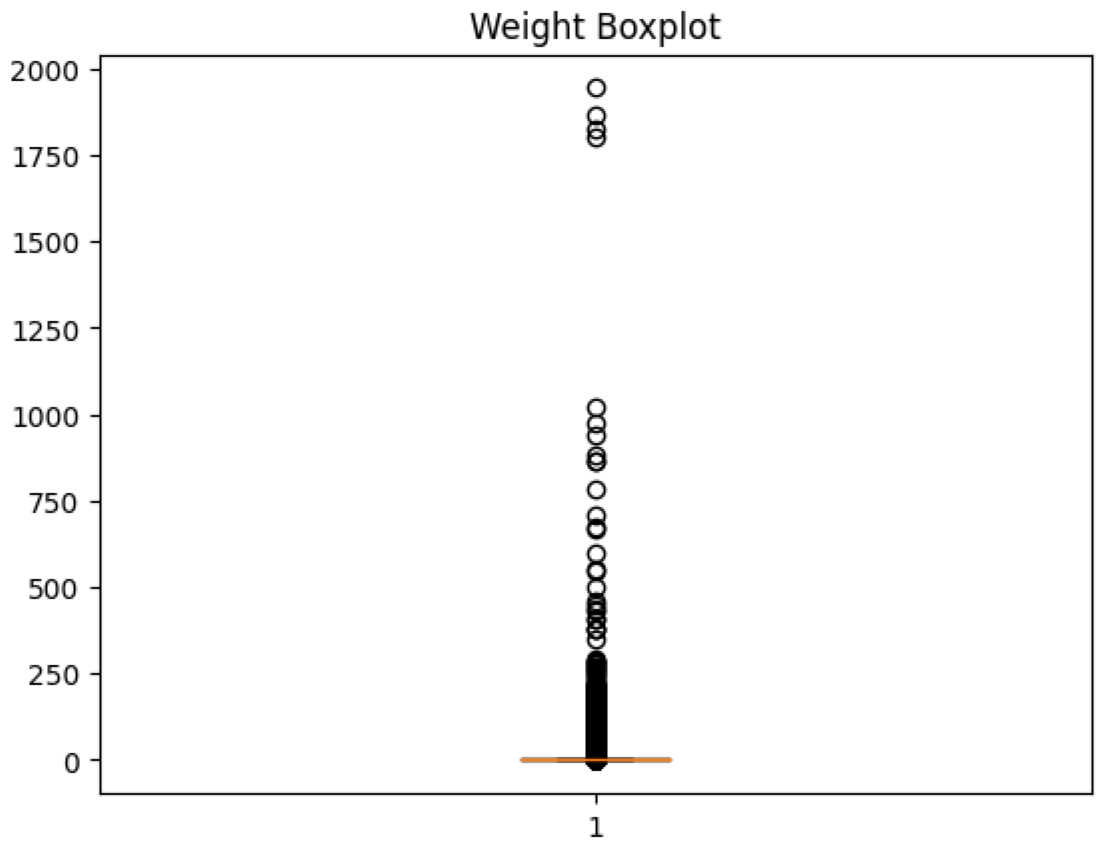
In [ ]:
```
print("Mean income = " + str(np.mean(data["income"])) + ", income standard deviation =
```

```
Mean income = 4464889007.186174, income standard deviation = 162685932780.61386
```

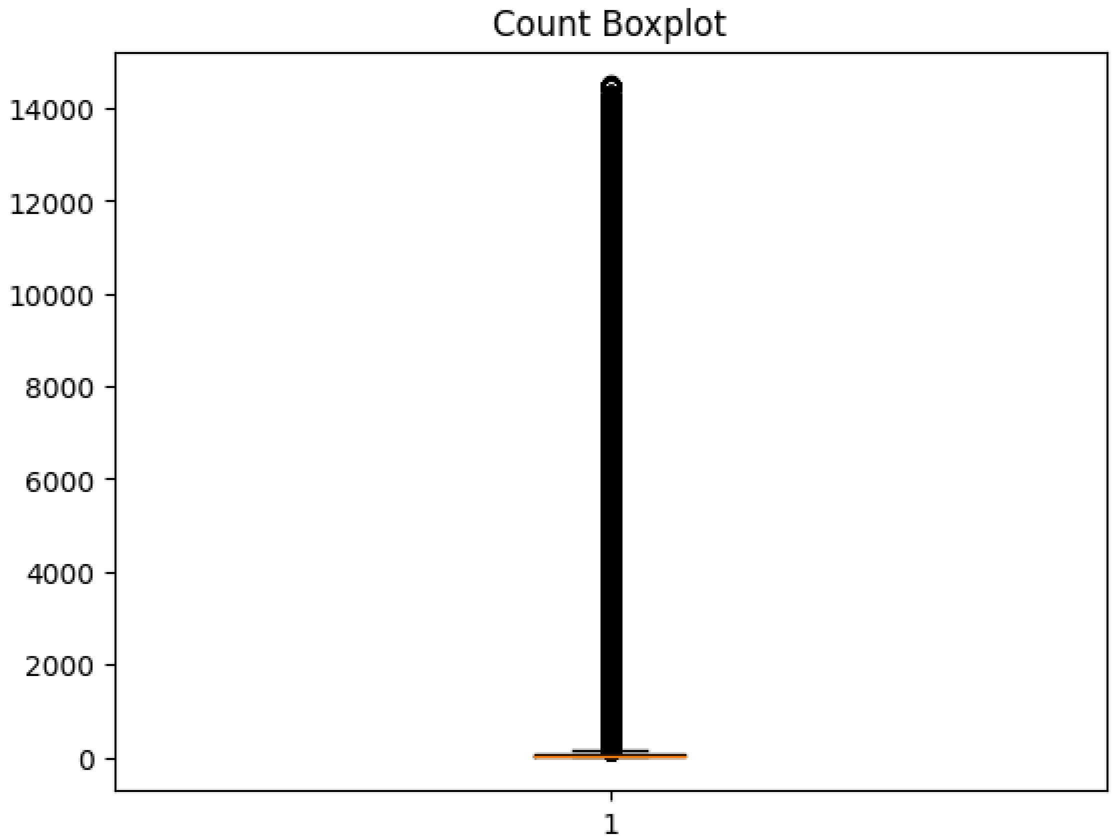
In [ ]:
```
plt.boxplot(data['length'])
plt.title("Length Boxplot")
plt.show()
```



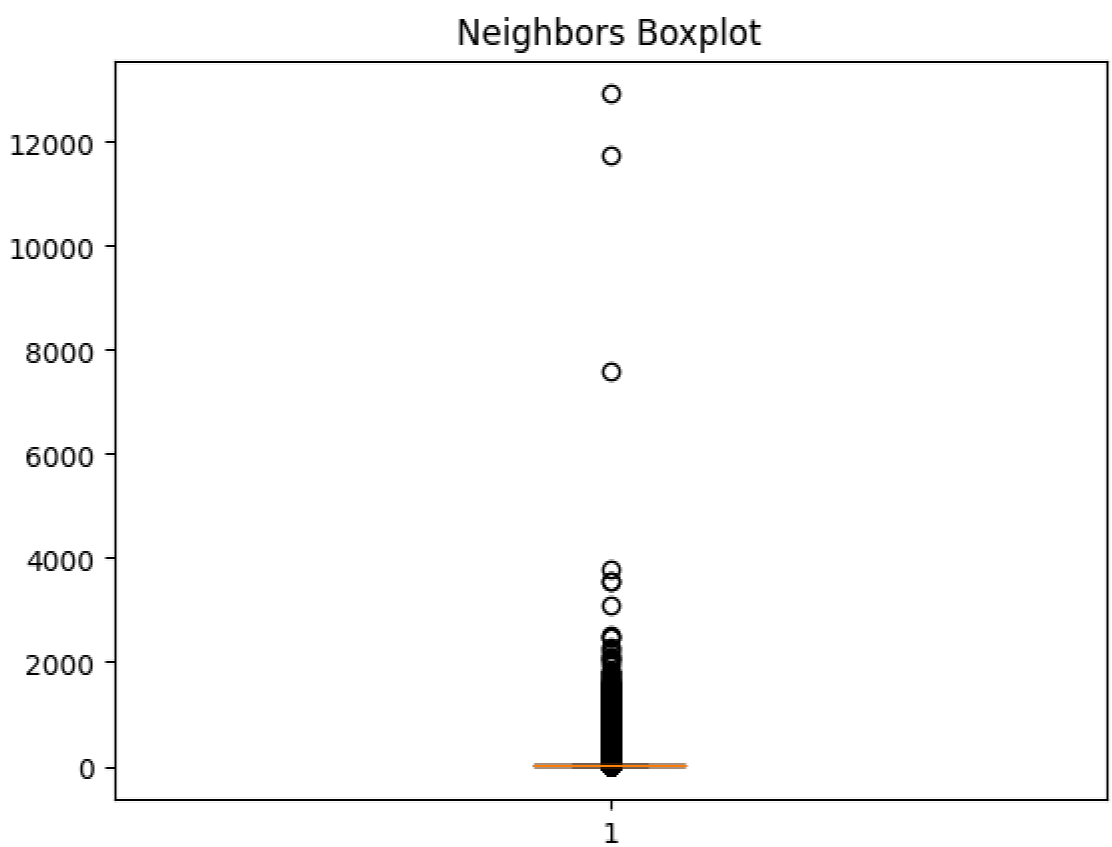
In [ ]:
```
plt.boxplot(data['weight'])
plt.title("Weight Boxplot")
plt.show()
```



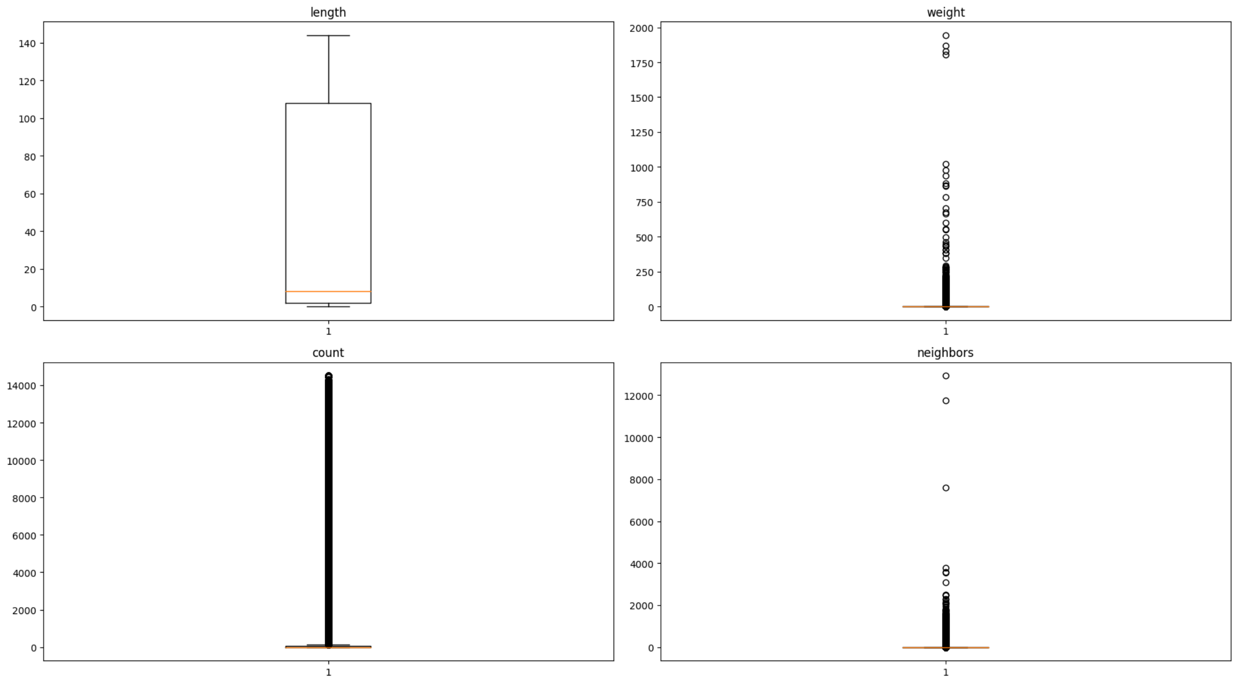
In [ ]:
```
plt.boxplot(data['count'])
plt.title("Count Boxplot")
plt.show()
```



In [ ]:
```
plt.boxplot(data['neighbors'])
plt.title("Neighbors Boxplot")
plt.show()
```



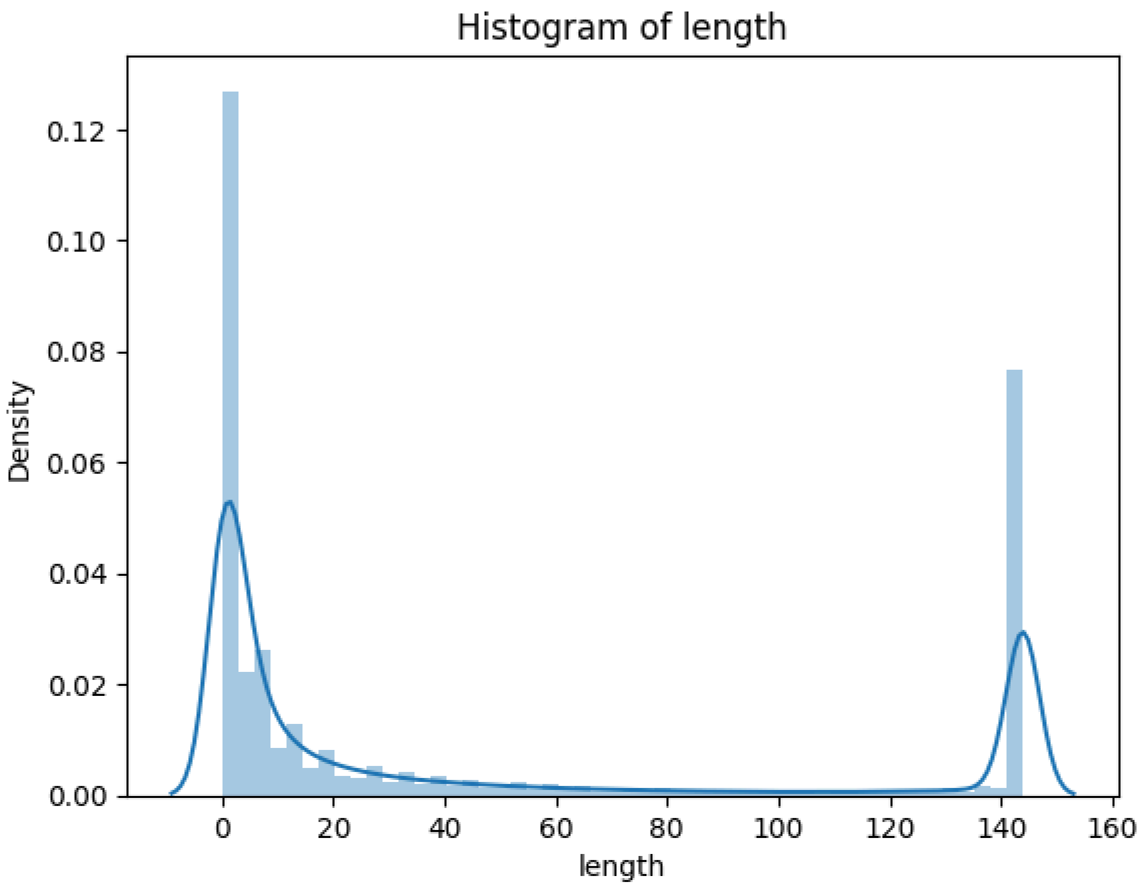
In [ ]:
```
# Create a figure and axis for each column
fig, axs = plt.subplots(2, 2, figsize=(18, 10))

# Plot histograms for each selected column
columns_to_plot = ['length', 'weight', 'count', 'neighbors']
for i in range(2):  # Iterate over rows
    for j in range(2):  # Iterate over columns
        col_idx = i * 2 + j
        axs[i, j].boxplot(data[columns_to_plot[col_idx]])
        axs[i, j].set_title(columns_to_plot[col_idx])

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

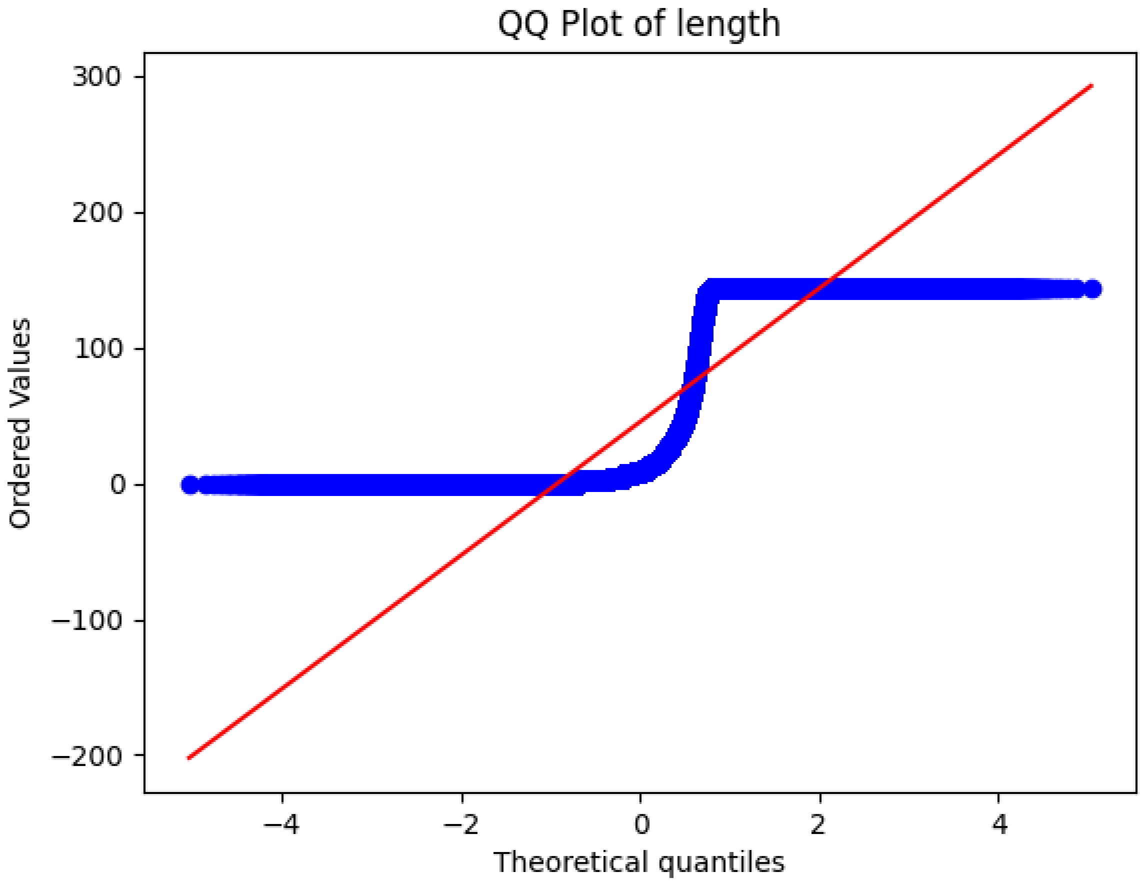


In [ ]:
```
from scipy.stats import norm, chi2_contingency, ttest_ind, f_oneway, pearsonr, probplo

# Univariate Analysis
for column in ['length', 'weight', 'count', 'neighbors']:
    # Histogram
    sns.distplot(data[column], kde=True)
    plt.title(f'Histogram of {column}')
    plt.show()

    # QQ Plot
    probplot(data[column], dist="norm", plot=plt)
    plt.title(f'QQ Plot of {column}')
    plt.show()

    # Estimate parameters
    mean = np.mean(data[column])
    std_dev = np.std(data[column])
    print(f"Mean of {column}: {mean}, Standard Deviation: {std_dev}")

    # Hypothesis testing (e.g., testing mean)
    # Example: Testing if mean is significantly different from 0
    t_stat, p_value = ttest_1samp(data[column], 0)
    print(f"T-statistic: {t_stat}, P-value: {p_value}")
```

```
    if p_value < 0.05:
        print("Reject null hypothesis")
```

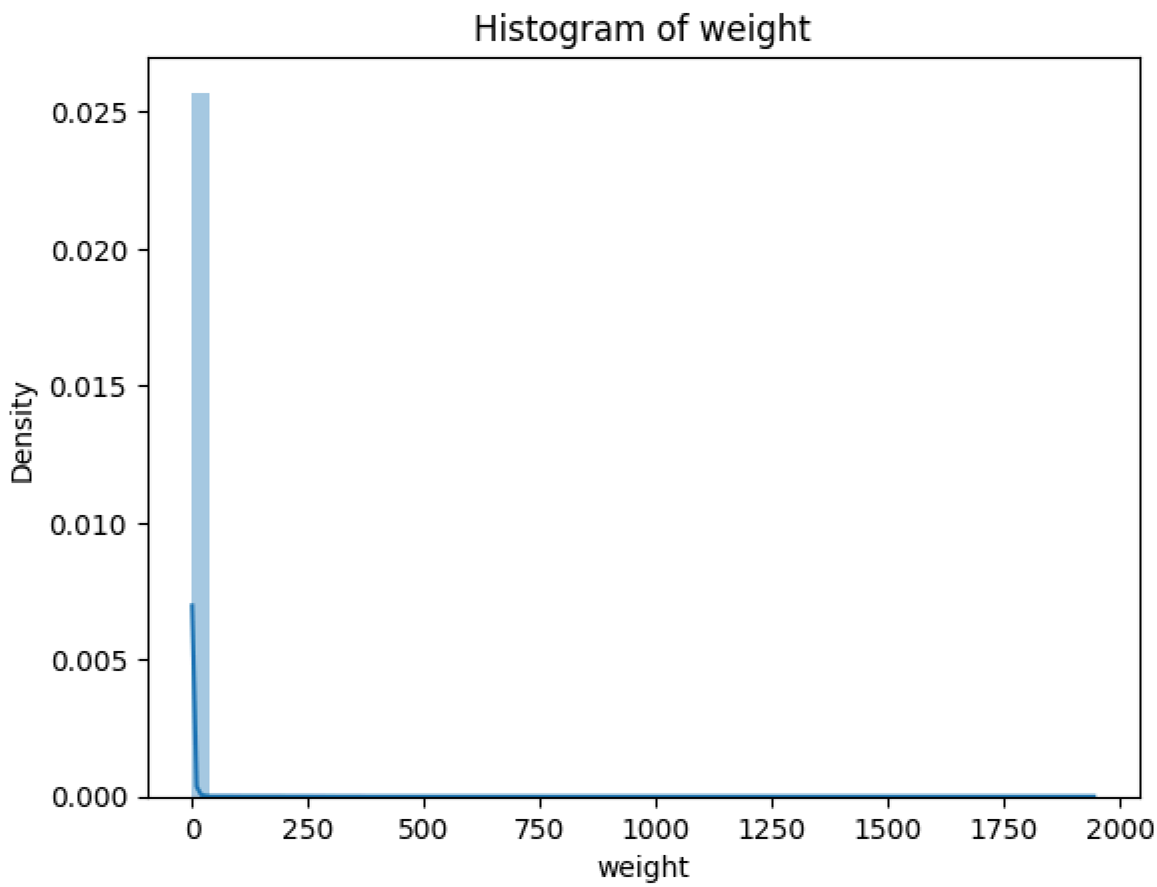
```
<ipython-input-30-d89811f479a1>:6: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(data[column], kde=True)
```

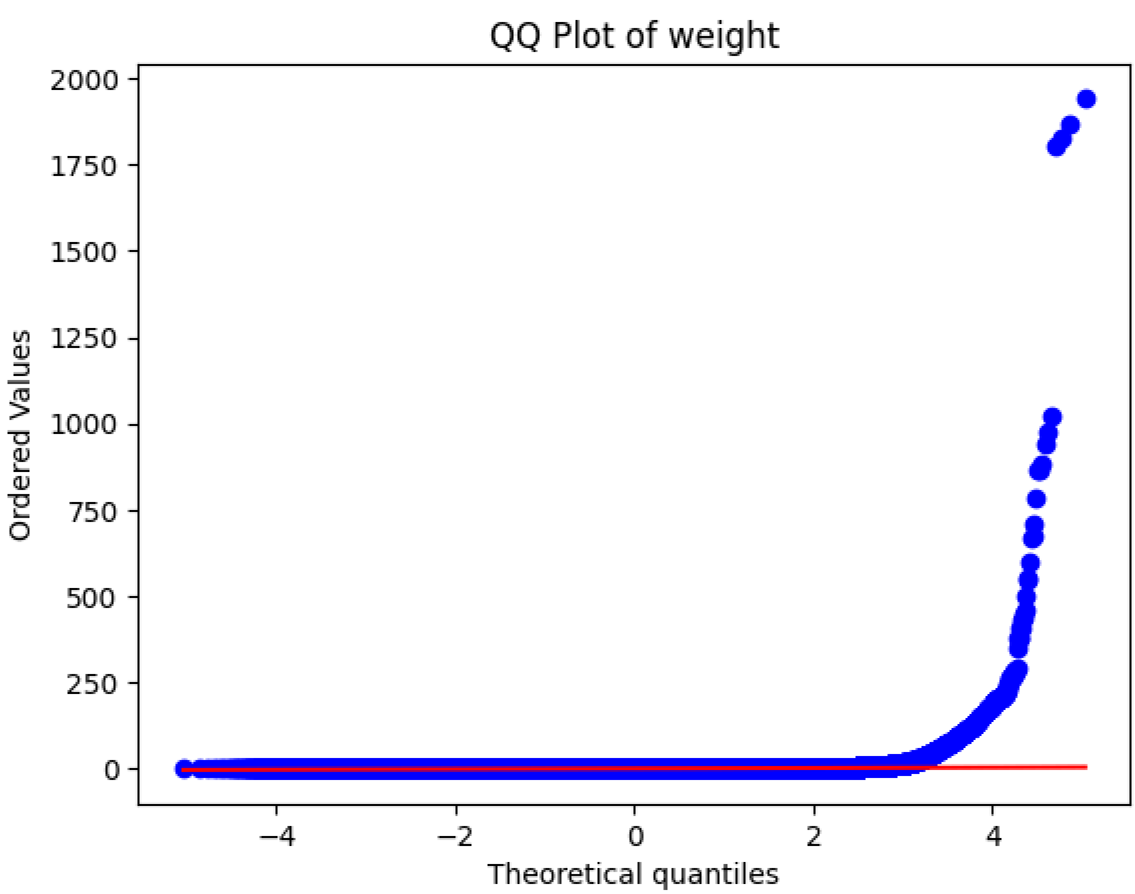





```
Mean of length: 45.00859293920486, Standard Deviation: 58.98235218811939
T-statistic: 1303.2235856969965, P-value: 0.0
Reject null hypothesis
```

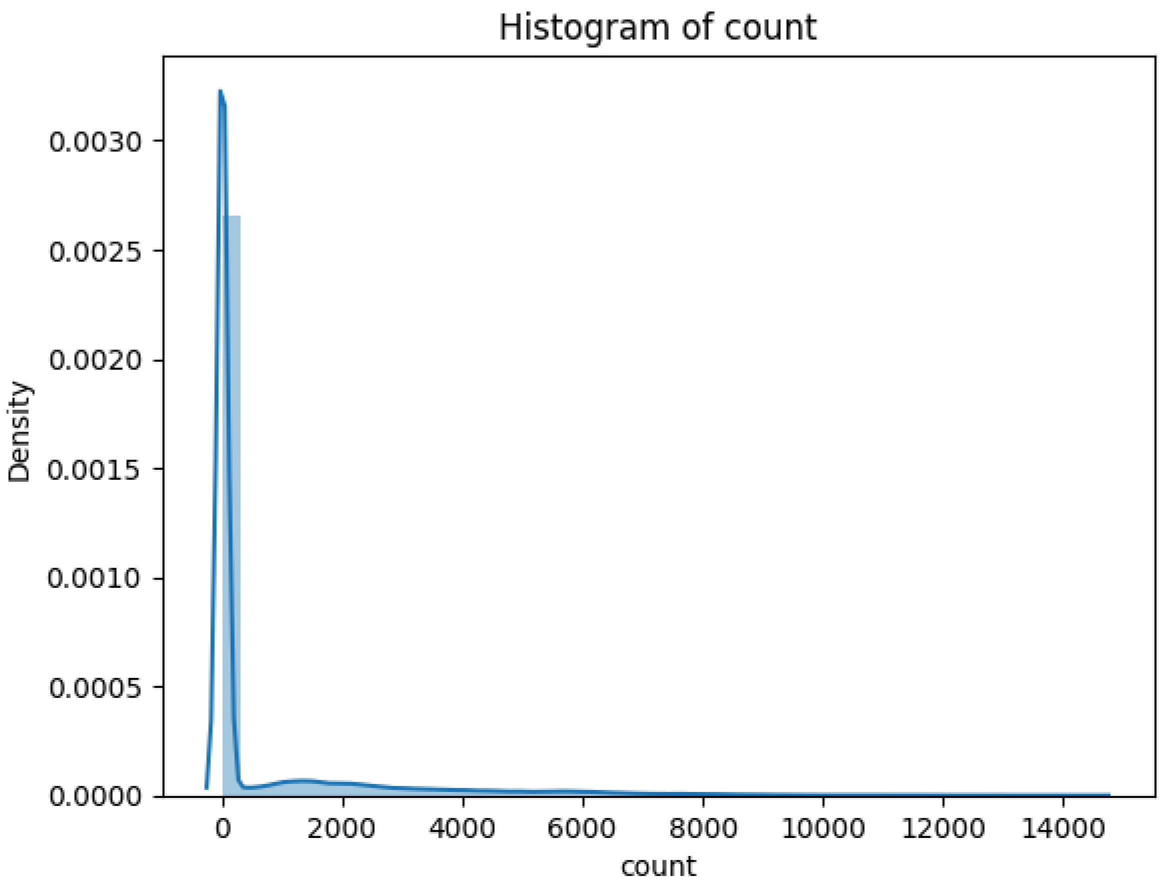
```
<ipython-input-30-d89811f479a1>:6: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(data[column], kde=True)
```

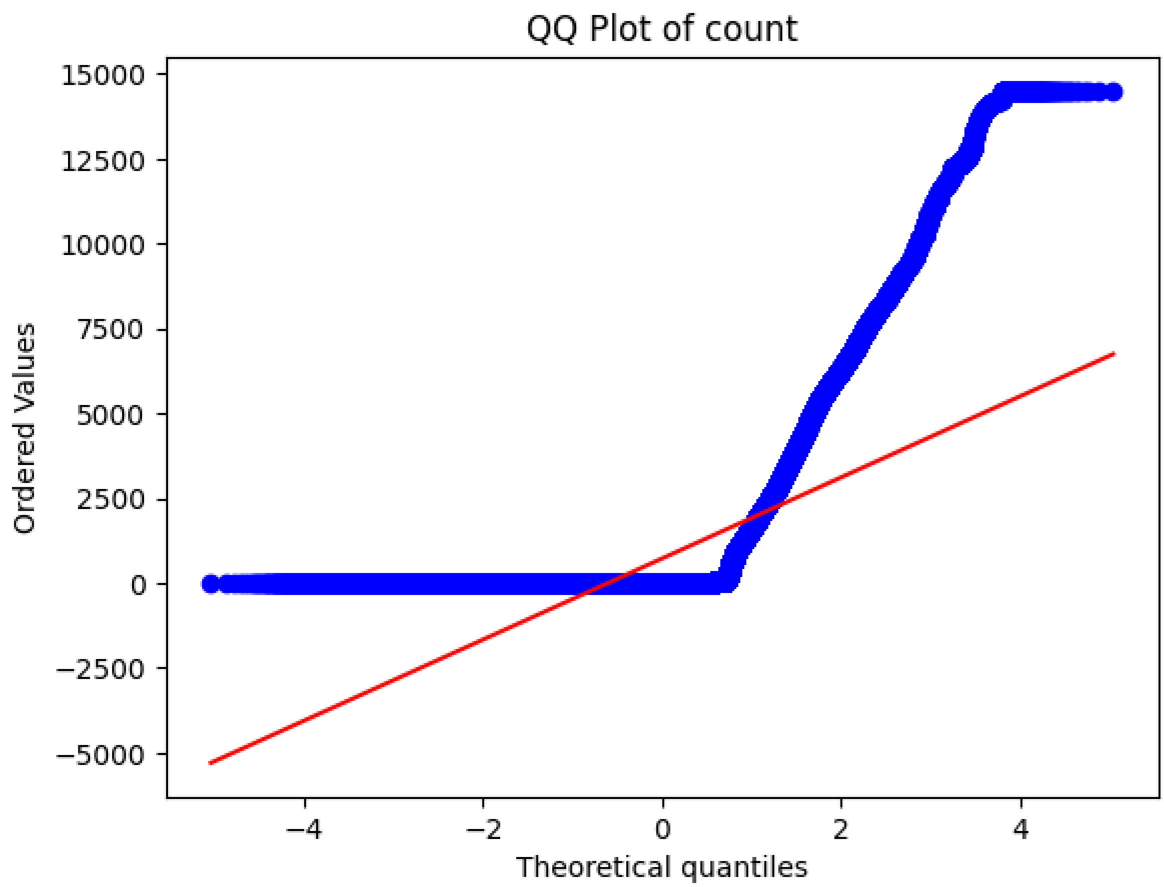





```
Mean of weight: 0.5455192341640024, Standard Deviation: 3.6742546257308684
T-statistic: 253.56330344304254, P-value: 0.0
Reject null hypothesis
```

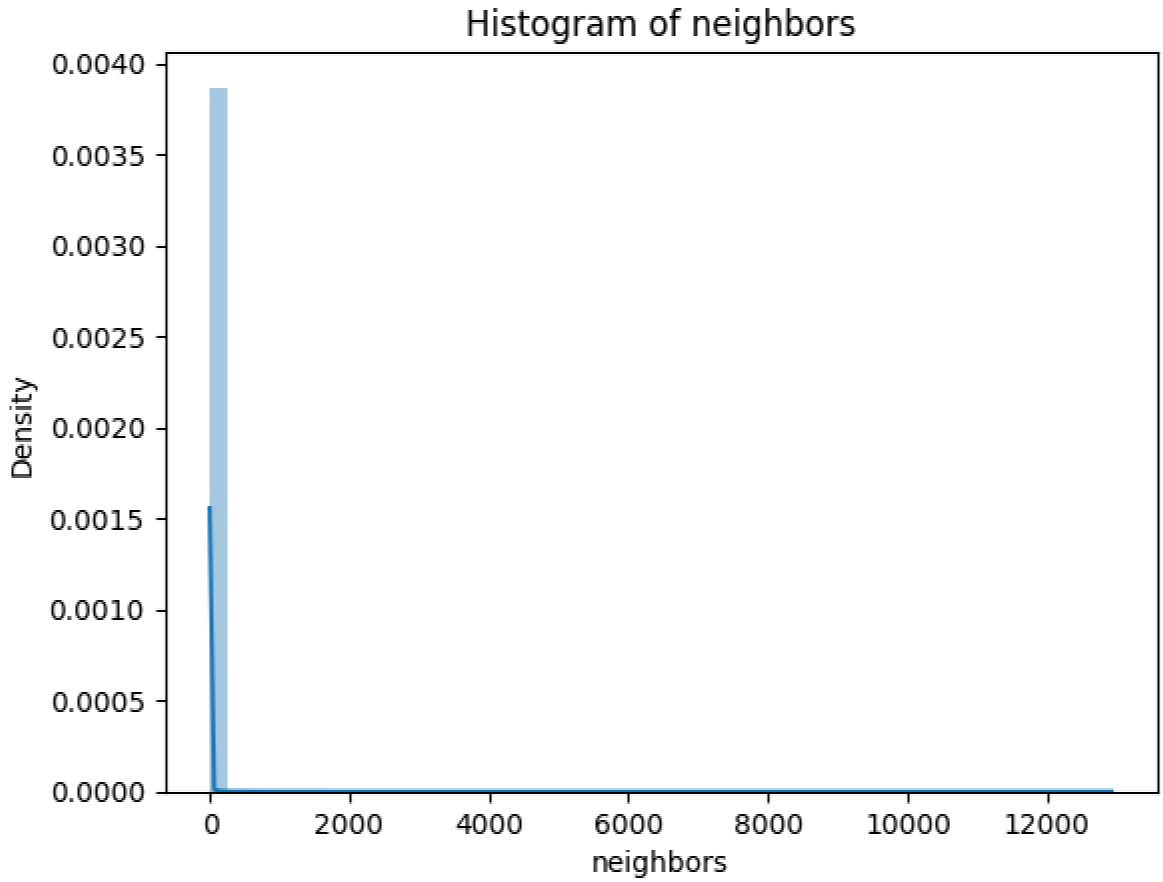
```
<ipython-input-30-d89811f479a1>:6: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(data[column], kde=True)
```

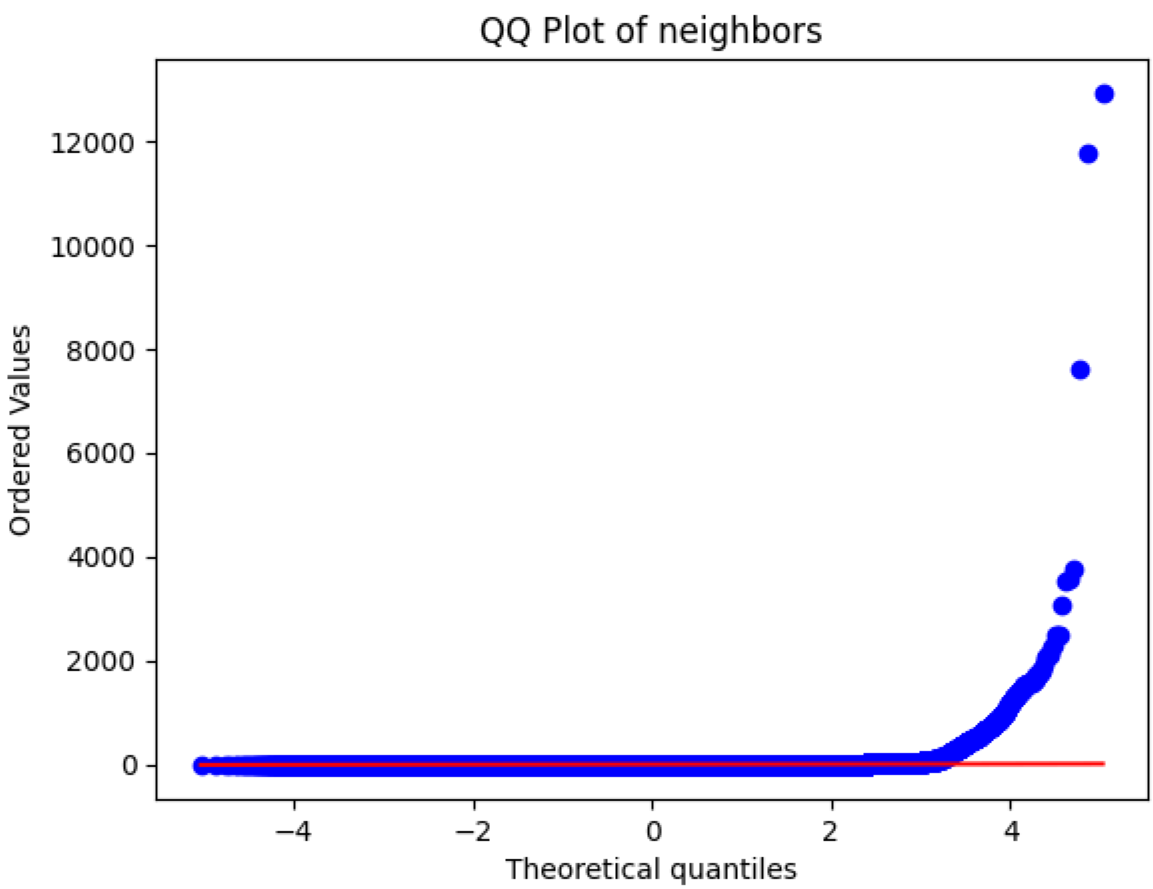





```
Mean of count: 721.6446428957139, Standard Deviation: 1689.6755041726624
T-statistic: 729.3998455598622, P-value: 0.0
Reject null hypothesis
```

```
<ipython-input-30-d89811f479a1>:6: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(data[column], kde=True)
```






```
Mean of neighbors: 2.206516137946451, Standard Deviation: 17.918762006490027
T-statistic: 210.30262319155054, P-value: 0.0
Reject null hypothesis
```

In [ ]:
```
data.label = pd.Categorical(data.label)

test = data.copy()
test_encode = pd.get_dummies(test.label)
test_encode.white
```

Out[ ]:
```
0          False
1          False
2          False
3          False
4          False
           ...
2916692     True
2916693     True
2916694     True
2916695     True
2916696     True
Name: white, Length: 2916697, dtype: bool
```

In [ ]:
```
test['label_bw'] = test_encode.white
test.drop('label', axis=1, inplace=True)
test.head()
```

Out[ ]:

	weight	length	count	neighbors	income	label_bw
0	0.008333	18	1	2	100050000.0	False
1	0.000244	44	1	1	100000000.0	False
2	1.000000	0	1	2	200000000.0	False
3	0.003906	72	1	2	71200000.0	False
4	0.072848	144	456	1	200000000.0	False

In [ ]:
```
test_corr = test.corr()
test_corr
```

Out[ ]:

	weight	length	count	neighbors	income	label_bw
weight	1.000000	0.000228	0.022313	0.691963	0.069774	-0.002676
length	0.000228	1.000000	0.703467	0.031523	0.000488	0.006860
count	0.022313	0.703467	1.000000	0.025441	-0.003635	0.008654
neighbors	0.691963	0.031523	0.025441	1.000000	0.138966	0.000872
income	0.069774	0.000488	-0.003635	0.138966	1.000000	0.002716
label_bw	-0.002676	0.006860	0.008654	0.000872	0.002716	1.000000

In [ ]:
```
data
```

Out[ ]:

	weight	length	count	neighbors	label
0	0.008333	18	1	2	princetonCerber
1	0.000244	44	1	1	princetonLocky
2	1.000000	0	1	2	princetonCerber
3	0.003906	72	1	2	princetonCerber
4	0.072848	144	456	1	princetonLocky
...	...	...	...	...	...
2916692	0.111111	0	1	1	white
2916693	1.000000	0	1	1	white
2916694	12.000000	2	6	35	white
2916695	0.500000	0	1	1	white
2916696	0.073972	144	6800	2	white

2916697 rows × 5 columns

In [ ]:
```
data['label'] = data.apply(
    lambda x: 1 if x['label'] == 'white' else 0,
    axis=1
)
data
```

Out[ ]:

	weight	length	count	neighbors	label
0	0.008333	18	1	2	0
1	0.000244	44	1	1	0
2	1.000000	0	1	2	0
3	0.003906	72	1	2	0
4	0.072848	144	456	1	0
...	...	...	...	...	...
2916692	0.111111	0	1	1	1
2916693	1.000000	0	1	1	1
2916694	12.000000	2	6	35	1
2916695	0.500000	0	1	1	1
2916696	0.073972	144	6800	2	1

2916697 rows × 5 columns

In [ ]:
```
corr_test = data.corr()
corr_test
```

Out[ ]:

	weight	length	count	neighbors	label
weight	1.000000	0.000228	0.022313	0.691963	-0.002676
length	0.000228	1.000000	0.703467	0.031523	0.006860
count	0.022313	0.703467	1.000000	0.025441	0.008654
neighbors	0.691963	0.031523	0.025441	1.000000	0.000872
label	-0.002676	0.006860	0.008654	0.000872	1.000000

In [ ]:

```
plt.figure(figsize = (12,8))
sns.heatmap(corr_test,alpha=0.95, annot=True)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-10-19c51a0a263f> in <cell line: 2>()
      1 plt.figure(figsize = (12,8))
----> 2 sns.heatmap(corr_test,alpha=0.95, annot=True)

NameError: name 'corr_test' is not defined
<Figure size 1200x800 with 0 Axes>
```

In [ ]:

```
print("Weight: " + str(stats.normaltest(data['weight'])))
print("Length: " + str(stats.normaltest(data['length'])))
print("Count: " + str(stats.normaltest(data['count'])))
print("Neighbors: " + str(stats.normaltest(data['neighbors'])))
```

```
Weight: NormaltestResult(statistic=15083010.11489625, pvalue=0.0)
Length: NormaltestResult(statistic=864404.313353176, pvalue=0.0)
Count: NormaltestResult(statistic=1771811.6898896296, pvalue=0.0)
Neighbors: NormaltestResult(statistic=15891400.930463219, pvalue=0.0)
```

In [ ]:

```
print("Weight: " + str(stats.shapiro(data['weight'])))
print("Length: " + str(stats.shapiro(data['length'])))
print("Count: " + str(stats.shapiro(data['count'])))
print("Neighbors: " + str(stats.shapiro(data['neighbors'])))
```

```
/usr/local/lib/python3.10/dist-packages/scipy/stats/_morestats.py:1882: UserWarning:
p-value may not be accurate for N > 5000.
  warnings.warn("p-value may not be accurate for N > 5000.")
Weight: ShapiroResult(statistic=0.045362114906311035, pvalue=0.0)
Length: ShapiroResult(statistic=0.7001427412033081, pvalue=0.0)
Count: ShapiroResult(statistic=0.5025076866149902, pvalue=0.0)
Neighbors: ShapiroResult(statistic=0.012168943881988525, pvalue=0.0)
```

In [ ]:

```
y = data[['label']]
x = data[['weight', 'length', 'count', 'neighbors']]
```

In [ ]:

```
sampled_df = data.sample(frac=0.1, random_state=42)
```

In [ ]:

```
len(sampled_df)
```

Out[ ]:

```
291670
```

In [ ]:

```
sampled_df = data.sample(frac=0.1, random_state=42)
#start with 100000
```

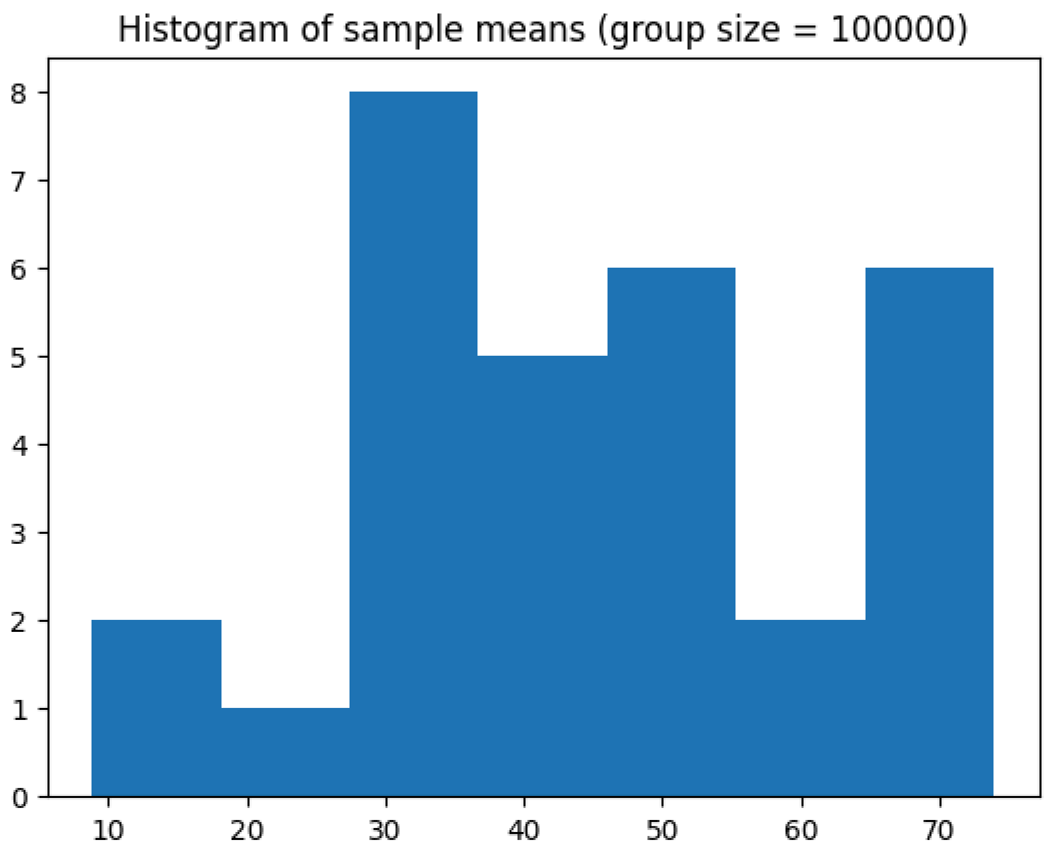
```
sample_means = []
i = 0
j = 300000
while j < len(x):
  subset = x.iloc[i:j]
  sample_mean = np.mean(subset['length'])
  sample_means.append(sample_mean)
  i+=300000
  j+=300000
sample_means.append(np.mean(x.iloc[i:]['length']))
```

In [ ]:
```
len(sample_means)
```

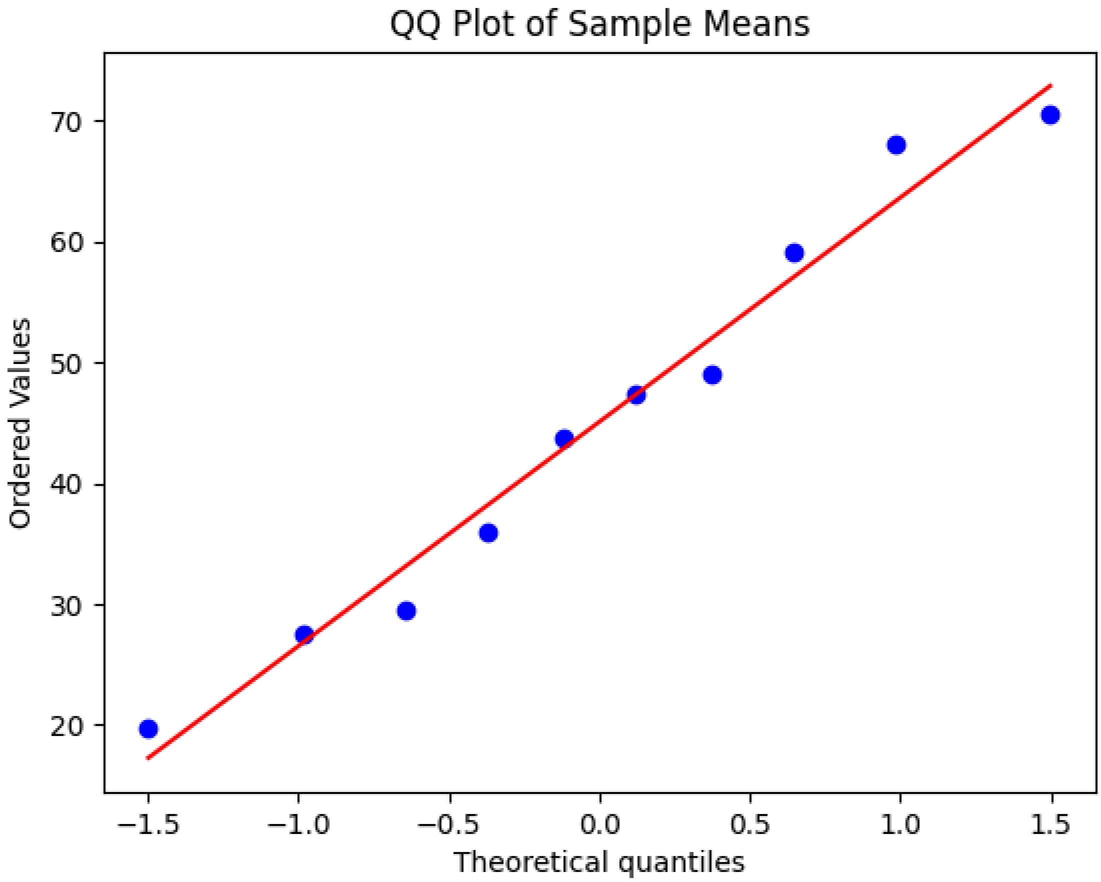
Out[ ]:
```
10
```

In [ ]:
```
plt.hist(sample_means, bins=7)
plt.title("Histogram of sample means (group size = 100000)")
```

Out[ ]:
```
Text(0.5, 1.0, 'Histogram of sample means (group size = 100000)')
```



In [ ]:
```
probplot(sample_means, dist="norm", plot=plt)
plt.title(f'QQ Plot of Sample Means')
plt.show()
```



In [ ]:
```
length = data['length']
print(np.std(length) / np.sqrt(300000))
```

```
0.10768654929382394
```

In [ ]:
```
np.sqrt(len(length))
```

Out[ ]:
```
1707.8340083275073
```

In [ ]:
```
pip install pingouin
```

```
Collecting pingouin
  Downloading pingouin-0.5.4-py2.py3-none-any.whl (198 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 198.9/198.9 kB 6.0 MB/s eta 0:00:00
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from
pingouin) (1.25.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from
pingouin) (1.11.4)
Requirement already satisfied: pandas>=1.5 in /usr/local/lib/python3.10/dist-packages
(from pingouin) (2.0.3)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages
(from pingouin) (3.7.1)
Requirement already satisfied: seaborn in /usr/local/lib/python3.10/dist-packages (fr
om pingouin) (0.13.1)
Requirement already satisfied: statsmodels in /usr/local/lib/python3.10/dist-packages
(from pingouin) (0.14.2)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-package
s (from pingouin) (1.2.2)
Collecting pandas-flavor (from pingouin)
  Downloading pandas_flavor-0.6.0-py3-none-any.whl (7.2 kB)
Requirement already satisfied: tabulate in /usr/local/lib/python3.10/dist-packages (f
rom pingouin) (0.9.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/di
st-packages (from pandas>=1.5->pingouin) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-package
s (from pandas>=1.5->pingouin) (2023.4)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packa
ges (from pandas>=1.5->pingouin) (2024.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-pac
kages (from matplotlib->pingouin) (1.2.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-package
s (from matplotlib->pingouin) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-pa
ckages (from matplotlib->pingouin) (4.51.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-pa
ckages (from matplotlib->pingouin) (1.4.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-pack
ages (from matplotlib->pingouin) (24.0)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packag
es (from matplotlib->pingouin) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-pac
kages (from matplotlib->pingouin) (3.1.2)
Requirement already satisfied: xarray in /usr/local/lib/python3.10/dist-packages (fro
m pandas-flavor->pingouin) (2023.7.0)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packag
es (from scikit-learn->pingouin) (1.4.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist
-packages (from scikit-learn->pingouin) (3.4.0)
Requirement already satisfied: patsy>=0.5.6 in /usr/local/lib/python3.10/dist-package
s (from statsmodels->pingouin) (0.5.6)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from p
atsy>=0.5.6->statsmodels->pingouin) (1.16.0)
Installing collected packages: pandas-flavor, pingouin
Successfully installed pandas-flavor-0.6.0 pingouin-0.5.4
```

In [ ]:

```
data['label'].unique()
```

Out[ ]:

```
array([0, 1])
```

In [ ]:

```
pip install pingouin
```

```
Collecting pingouin
  Downloading pingouin-0.5.4-py2.py3-none-any.whl (198 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 198.9/198.9 kB 1.6 MB/s eta 0:00:00
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from pingouin) (1.25.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from pingouin) (1.11.4)
Requirement already satisfied: pandas>=1.5 in /usr/local/lib/python3.10/dist-packages (from pingouin) (2.0.3)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (from pingouin) (3.7.1)
Requirement already satisfied: seaborn in /usr/local/lib/python3.10/dist-packages (from pingouin) (0.13.1)
Requirement already satisfied: statsmodels in /usr/local/lib/python3.10/dist-packages (from pingouin) (0.14.2)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (from pingouin) (1.2.2)
Collecting pandas-flavor (from pingouin)
  Downloading pandas_flavor-0.6.0-py3-none-any.whl (7.2 kB)
Requirement already satisfied: tabulate in /usr/local/lib/python3.10/dist-packages (from pingouin) (0.9.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.5->pingouin) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.5->pingouin) (2023.4)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.5->pingouin) (2024.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->pingouin) (1.2.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib->pingouin) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->pingouin) (4.51.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->pingouin) (1.4.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->pingouin) (24.0)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->pingouin) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->pingouin) (3.1.2)
Requirement already satisfied: xarray in /usr/local/lib/python3.10/dist-packages (from pandas-flavor->pingouin) (2023.7.0)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn->pingouin) (1.4.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn->pingouin) (3.4.0)
Requirement already satisfied: patsy>=0.5.6 in /usr/local/lib/python3.10/dist-packages (from statsmodels->pingouin) (0.5.6)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from patsy>=0.5.6->statsmodels->pingouin) (1.16.0)
Installing collected packages: pandas-flavor, pingouin
Successfully installed pandas-flavor-0.6.0 pingouin-0.5.4
```

In [ ]:

```python
import pingouin as pg
mod1 = pg.logistic_regression(x, data['label'])
mod1.round(6)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:1173: Futur
eWarning: `penalty='none'`has been deprecated in 1.2 and will be removed in 1.4. To k
eep the past behaviour, set `penalty=None`.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:314: LineSearch
Warning: The line search algorithm did not converge
  warn('The line search algorithm did not converge', LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/optimize.py:203: UserWarning: L
ine Search failed
  warnings.warn("Line Search failed")
```

Out[ ]:

	names	coef	se	z	pval	CI[2.5%]	CI[97.5%]
0	Intercept	4.203567	0.006257	671.868486	0.000000	4.191305	4.215830
1	weight	-0.006034	0.001202	-5.018040	0.000001	-0.008390	-0.003677
2	length	0.000127	0.000121	1.045769	0.295668	-0.000111	0.000365
3	count	0.000045	0.000005	9.817908	0.000000	0.000036	0.000054
4	neighbors	0.002328	0.000767	3.033492	0.002417	0.000824	0.003832

In [ ]:

```
import pingouin as pg
mod1 = pg.logistic_regression(data['weight'], data['label'])
mod1.round(6)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:1173: Futur
eWarning: `penalty='none'`has been deprecated in 1.2 and will be removed in 1.4. To k
eep the past behaviour, set `penalty=None`.
  warnings.warn(
```

Out[ ]:

	names	coef	se	z	pval	CI[2.5%]	CI[97.5%]
0	Intercept	-4.241393	0.004959	-855.326373	0.000000	-4.251112	-4.231674
1	weight	0.001897	0.000507	3.744542	0.000181	0.000904	0.002890

In [ ]:

```
import pingouin as pg
mod1 = pg.logistic_regression(data['length'], data['label'])
mod1.round(6)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:1173: Futur
eWarning: `penalty='none'`has been deprecated in 1.2 and will be removed in 1.4. To k
eep the past behaviour, set `penalty=None`.
  warnings.warn(
```

Out[ ]:

	names	coef	se	z	pval	CI[2.5%]	CI[97.5%]
0	Intercept	-4.196516	0.006119	-685.813976	0.0	-4.208509	-4.184523
1	length	-0.001011	0.000086	-11.708912	0.0	-0.001180	-0.000842

In [ ]:

```
import pingouin as pg
mod1 = pg.logistic_regression(data['count'], data['label'])
mod1.round(6)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:1173: Futur
eWarning: `penalty='none'`has been deprecated in 1.2 and will be removed in 1.4. To k
eep the past behaviour, set `penalty=None`.
  warnings.warn(
```

Out[ ]:

	names	coef	se	z	pval	CI[2.5%]	CI[97.5%]
0	Intercept	-4.208338	0.005330	-789.610341	0.0	-4.218784	-4.197892
1	count	-0.000049	0.000003	-14.751152	0.0	-0.000055	-0.000042

In [ ]:

```
print(len(data[data['label'] == 'white']))
```

```
2875284
```

In [ ]:

```
import pingouin as pg
mod1 = pg.logistic_regression(data['neighbors'], data['label'])
mod1.round(6)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:1173: Futur
eWarning: `penalty='none'`has been deprecated in 1.2 and will be removed in 1.4. To k
eep the past behaviour, set `penalty=None`.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:314: LineSearch
Warning: The line search algorithm did not converge
  warn('The line search algorithm did not converge', LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/optimize.py:203: UserWarning: L
ine Search failed
  warnings.warn("Line Search failed")
```

Out[ ]:

	names	coef	se	z	pval	CI[2.5%]	CI[97.5%]
0	Intercept	-0.007134	0.001472	-4.847135	0.000001	-0.010018	-0.004249
1	neighbors	-0.015768	0.000461	-34.212406	0.000000	-0.016671	-0.014865

In [ ]:

```
0          princetonCerber
1           princetonLocky
2          princetonCerber
3          princetonCerber
4           princetonLocky
                ...
2916692              white
2916693              white
2916694              white
2916695              white
2916696              white
Name: label, Length: 2916697, dtype: object
```

In [ ]:

```
y = data['label']
x = data[['weight', 'length', 'count', 'neighbors']]
```

In [ ]:

```
y
```

Out[ ]:
```
0          0
1          0
2          0
3          0
4          0
          ..
2916692    1
2916693    1
2916694    1
2916695    1
2916696    1
Name: label, Length: 2916697, dtype: int64
```

In [ ]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=
```

In [ ]:
```
y_test
```

Out[ ]:
```
2667698    1
2013400    1
2704440    1
1650833    1
1586811    1
          ..
2028843    1
1014855    1
2789801    1
1848639    1
1379081    1
Name: label, Length: 583340, dtype: int64
```

In [ ]:
```
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(random_state=0).fit(X_train, y_train)
```

In [ ]:
```
clf.predict(X_test)
```

Out[ ]:
```
array([1, 1, 1, ..., 1, 1, 1])
```

In [ ]:
```
clf.coef_
```

Out[ ]:
```
array([[-6.62593469e-03,  1.67360491e-04,  4.46227561e-05,
         2.63980118e-03]])
```

In [ ]:
```
clf.intercept_
```

Out[ ]:
```
array([4.19631637])
```

In [ ]:
```
# Importing basic libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

1) Meet the Data

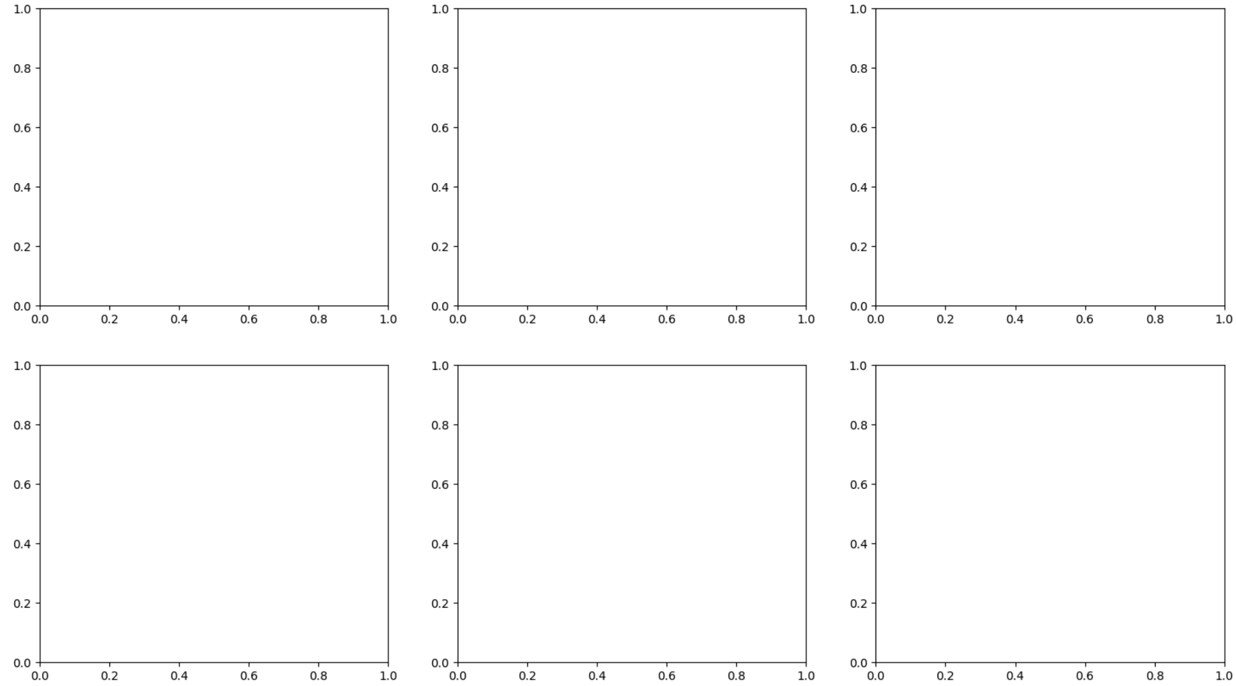
In [ ]:
```
# Importing the data

df = pd.read_csv("BitcoinHeistData.csv")
df.head()
```

Out[ ]:

	address	year	day	length	weight	count	looped	neighbors	
0	111K8kZAEnJg245r2cM6y9zgJGHZtJPy6	2017	11	18	0.008333	1	0	2	1
1	1123pJv8jzeFQaCV4w644pzQJzVWay2zcA	2016	132	44	0.000244	1	0	1	1
2	112536im7hy6wtKbpH1qYDWtTyMRAcA2p7	2016	246	0	1.000000	1	0	2	2
3	1126eDRw2wqSkWosjTCre8cjjQW8sSeWH7	2016	322	72	0.003906	1	0	2	
4	1129TSjKtx65E35GiUo4AYVeyo48twbrGX	2016	238	144	0.072848	456	0	1	2

In [ ]:
```
# Create a figure and axis for each column
fig, axs = plt.subplots(2, 3, figsize=(18, 10))

# Plot histograms for each selected column
columns_to_plot = ['length', 'weight', 'count', 'looped', 'neighbors', 'income']
for i in range(2):  # Iterate over rows
    for j in range(3):  # Iterate over columns
        col_idx = i * 3 + j
        axs[i, j].probplot(df[columns_to_plot[col_idx]], dist="norm", plot=plt)
        # axs[i, j].hist(df[columns_to_plot[col_idx]])
        axs[i, j].set_title(columns_to_plot[col_idx])

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

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-119-d9e3b2b7e692> in <cell line: 6>()
      7     for j in range(3):  # Iterate over columns
      8         col_idx = i * 3 + j
----> 9         axs[i, j].probplot(df[columns_to_plot[col_idx]], dist="norm", plot=pl
t)
     10         # axs[i, j].hist(df[columns_to_plot[col_idx]])
     11         axs[i, j].set_title(columns_to_plot[col_idx])

AttributeError: 'Axes' object has no attribute 'probplot'
```



In [ ]:
```
df = df[['length', 'weight', 'count', 'neighbors', 'label']]
```

Dataset Description

This dataset contains the entire Bitcoin transaction graph from January 2009 to December

1. The researchers extracted daily transitions on the network and formed the graph using a time interval of 24 hours. Any network edges that transfer less than B0.3 were filtered out, and there are no missing values.

Features

The variables of this dataset include: 1) address: String - Bitcoin address 2) year: Integer - Year 3) day: Integer - Day of the year (1 is the first day, 365 is the last day) 4) length: Integer 5) weight: Float 6) count: Integer 7) looped: Integer 8) neighbors: Integer 9) income: Integer - Satoshi amount (1 bitcoin = 100 million satoshis) 10) label: Category String - Name of the ransomware family (Cryptxxx, cryptolocker etc) or white (not known to be ransomware)

In [ ]:
```
df.shape
```

Out[ ]:
```
(2916697, 5)
```

In [ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2916697 entries, 0 to 2916696
Data columns (total 5 columns):
 #   Column     Dtype  
---  ------     -----  
 0   length     int64  
 1   weight     float64
 2   count      int64  
 3   neighbors  int64  
 4   label      object 
dtypes: float64(1), int64(3), object(1)
memory usage: 111.3+ MB
```

In [ ]:

```
# Dropping irrelevant columns

df.drop(columns={'address', 'year', 'day'}, inplace=True)
df.head(2)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-6-7447a6fa8cee> in <cell line: 3>()
      1 # Dropping irrelevant columns
      2 
----> 3 df.drop(columns={'address', 'year', 'day'}, inplace=True)
      4 df.head(2)

/usr/local/lib/python3.10/dist-packages/pandas/core/frame.py in drop(self, labels, ax
is, index, columns, level, inplace, errors)
   5256                 weight  1.0     0.8
   5257         """
-> 5258         return super().drop(
   5259             labels=labels,
   5260             axis=axis,

/usr/local/lib/python3.10/dist-packages/pandas/core/generic.py in drop(self, labels, 
axis, index, columns, level, inplace, errors)
   4547         for axis, labels in axes.items():
   4548             if labels is not None:
-> 4549                 obj = obj._drop_axis(labels, axis, level=level, errors=error
s)
   4550 
   4551         if inplace:

/usr/local/lib/python3.10/dist-packages/pandas/core/generic.py in _drop_axis(self, la
bels, axis, level, errors, only_slice)
   4589                 new_axis = axis.drop(labels, level=level, errors=errors)
   4590             else:
-> 4591                 new_axis = axis.drop(labels, errors=errors)
   4592             indexer = axis.get_indexer(new_axis)
   4593 

/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in drop(self, lab
els, errors)
   6697         if mask.any():
   6698             if errors != "ignore":
-> 6699                 raise KeyError(f"{list(labels[mask])} not found in axis")
   6700             indexer = indexer[~mask]
   6701         return self.delete(indexer)

KeyError: "['address', 'year', 'day'] not found in axis"
```

In [ ]:
```
df['label'].value_counts()
```

Out[ ]:
```
label
white                          2875284
paduaCryptoWall                  12390
montrealCryptoLocker              9315
princetonCerber                   9223
princetonLocky                    6625
montrealCryptXXX                  2419
montrealNoobCrypt                  483
montrealDMALockerv3                354
montrealDMALocker                  251
montrealSamSam                      62
montrealCryptoTorLocker2015         55
montrealGlobeImposter               55
montrealGlobev3                     34
montrealGlobe                       32
montrealWannaCry                    28
montrealRazy                        13
montrealAPT                         11
paduaKeRanger                       10
montrealFlyper                       9
montrealXTPLocker                    8
montrealXLockerv5.0                  7
montrealVenusLocker                  7
montrealCryptConsole                 7
montrealEDA2                         6
montrealJigSaw                       4
paduaJigsaw                          2
montrealXLocker                      1
montrealSam                          1
montrealComradeCircle                1
Name: count, dtype: int64
```

In [ ]:
```
# We are only concerned with a binary target, i.e. whether it is a Ransomware or not
df['label'] = df['label'].apply(lambda x: 1 if x == 'white' else 0)
df.head(2)
```

Out[ ]:

	length	weight	count	neighbors	label
0	18	0.008333	1	2	0
1	44	0.000244	1	1	0

In [ ]:
```
df.describe()
```

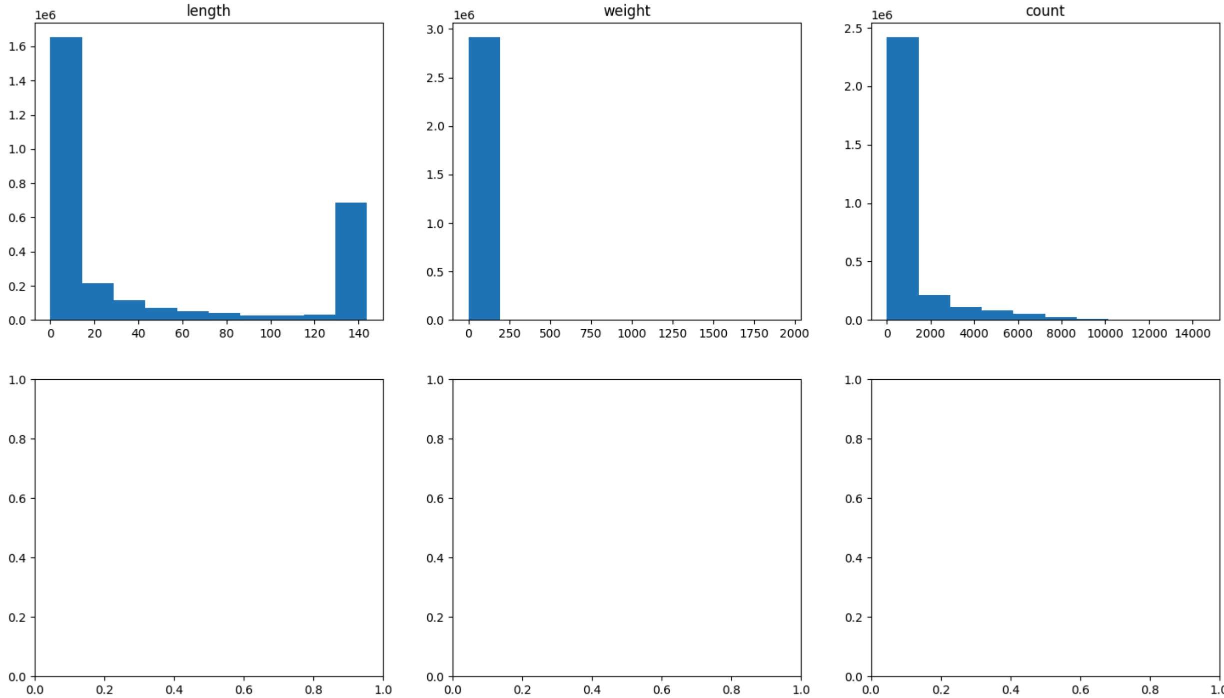
Out[ ]:

	length	weight	count	neighbors	label
count	2.916697e+06	2.916697e+06	2.916697e+06	2.916697e+06	2.916697e+06
mean	4.500859e+01	5.455192e-01	7.216446e+02	2.206516e+00	9.858014e-01
std	5.898236e+01	3.674255e+00	1.689676e+03	1.791877e+01	1.183089e-01
min	0.000000e+00	3.606469e-94	1.000000e+00	1.000000e+00	0.000000e+00
25%	2.000000e+00	2.148438e-02	1.000000e+00	1.000000e+00	1.000000e+00
50%	8.000000e+00	2.500000e-01	1.000000e+00	2.000000e+00	1.000000e+00
75%	1.080000e+02	8.819482e-01	5.600000e+01	2.000000e+00	1.000000e+00
max	1.440000e+02	1.943749e+03	1.449700e+04	1.292000e+04	1.000000e+00

In [ ]:

```
# Create a figure and axis for each column
fig, axs = plt.subplots(2, 3, figsize=(18, 10))

# Plot histograms for each selected column
columns_to_plot = ['length', 'weight', 'count', 'looped', 'neighbors', 'income']
for i in range(2):  # Iterate over rows
    for j in range(3):  # Iterate over columns
        col_idx = i * 3 + j
        axs[i, j].hist(df[columns_to_plot[col_idx]])
        axs[i, j].set_title(columns_to_plot[col_idx])

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

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in get_loc(self, key)
   3652         try:
-> 3653             return self._engine.get_loc(casted_key)
   3654         except KeyError as err:

/usr/local/lib/python3.10/dist-packages/pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()

/usr/local/lib/python3.10/dist-packages/pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'looped'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
<ipython-input-10-91eeea69df83> in <cell line: 6>()
      7     for j in range(3):  # Iterate over columns
      8         col_idx = i * 3 + j
----> 9         axs[i, j].hist(df[columns_to_plot[col_idx]])
     10         axs[i, j].set_title(columns_to_plot[col_idx])
     11 

/usr/local/lib/python3.10/dist-packages/pandas/core/frame.py in __getitem__(self, key)
   3759             if self.columns.nlevels > 1:
   3760                 return self._getitem_multilevel(key)
-> 3761             indexer = self.columns.get_loc(key)
   3762             if is_integer(indexer):
   3763                 indexer = [indexer]

/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in get_loc(self, key)
   3653             return self._engine.get_loc(casted_key)
   3654         except KeyError as err:
-> 3655             raise KeyError(key) from err
   3656         except TypeError:
   3657             # If we have a listlike key, _check_indexing_error will raise

KeyError: 'looped'
```



In [ ]:
```
df.columns
```

Out[ ]:
```
Index(['length', 'weight', 'count', 'looped', 'neighbors', 'income', 'label'], dtype
='object')
```

4) Statistical Inference

We perform univariate analysis on each column ('length', 'weight', 'count', 'looped', 'neighbors', 'income') in the dataset.

1. For each column, we first create a histogram to visualize the distribution of the data. The histogram provides insights into the shape and spread of the data.
2. We then create a QQ plot (Quantile-Quantile plot) to compare the distribution of the data to a normal distribution. This plot helps us assess if the data follows a normal distribution.
3. Next, we calculate the mean and standard deviation of each column to summarize the central tendency and spread of the data.
4. Finally, we conduct hypothesis testing to determine if the mean of each column is significantly different from 0. We use a one-sample t-test for this purpose. The t-test results in a t-statistic and a p-value. If the p-value is less than 0.05 (a commonly used significance level), we reject the null hypothesis, indicating that the mean is significantly different from 0.

In [ ]:
```
# Taking 1/10th data as the ssample dataset
# Assuming df is your original DataFrame
sampled_df = df.sample(frac=0.1, random_state=42)
```

In [ ]:
```
print(len(df))
```

```
2916697
```

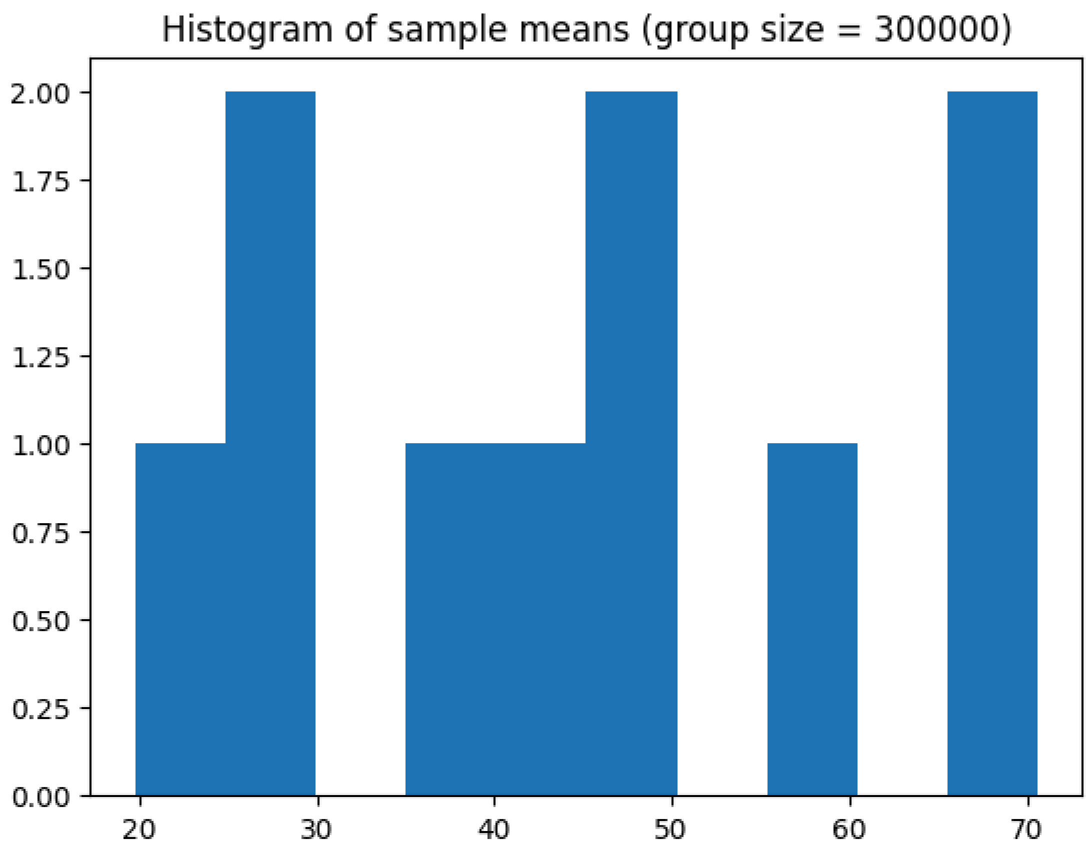
In [ ]:
```
#start with 100000
sample_means = []
i = 0
j = 300000
while j < len(df):
  subset = df.iloc[i:j]
  sample_mean = np.mean(subset['length'])
  sample_means.append(sample_mean)
  i+=300000
  j+=300000
sample_means.append(np.mean(df.iloc[i:]['length']))
```

In [ ]:
```
print(len(sample_means))
```

```
10
```

In [ ]:
```
plt.hist(sample_means)
plt.title("Histogram of sample means (group size = 300000)")
```

Out[ ]:
```
Text(0.5, 1.0, 'Histogram of sample means (group size = 300000)')
```



In [ ]:
```
probplot(df['income'], dist="norm", plot=plt)
plt.title(f'QQ Plot of Sample Means')
plt.show()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-6d349ec542cc> in <cell line: 1>()
----> 1 probplot(df['income'], dist="norm", plot=plt)
      2 plt.title(f'QQ Plot of Sample Means')
      3 plt.show()

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

In [ ]:

```
print("Population mean: " + str(np.mean(df['length'])))
print("Mean of sample means: " + str(np.mean(sample_means)))
print("Population standard deviation: " + str(np.std(df['length'])))
print("Standard deviation of sample means: " + str(np.std(sample_means)))
print("Population standard deviation divided by sqrt(n): " + str(np.std(df['length'])
```

```
Population mean: 45.00859293920486
Mean of sample means: 45.07368511257962
Population standard deviation: 58.98235218811939
Standard deviation of sample means: 16.34651349643786
Population standard deviation divided by sqrt(n): 0.034536349493286635
```

In [ ]:

```
ten_sample = df[2100000:2400000]
thirty_sample = df[2800000:2900000]
```

In [ ]:

```
print(ten_sample['length'])
```

```
2400000      2
2400001    144
2400002    144
2400003     88
2400004      0
          ...
2699995      0
2699996    144
2699997    144
2699998      2
2699999     14
Name: length, Length: 300000, dtype: int64
```

In [ ]:

```
import scipy.stats as st
print("95% Confidence Interval for 10 sample case: "+ str(st.t.interval(0.95, len(ten_
print("95% Confidence Interval for 30 sample case: "+ str(st.t.interval(0.95, len(thir
# import statsmodels.stats.api as sms

# sms.DescrStatsW(ten_sample['length']).tconfint_mean()
```

```
95% Confidence Interval for 10 sample case: (43.58853432700981, 44.013385672990196)
95% Confidence Interval for 30 sample case: (44.61371891914963, 45.352561080850364)
```

In [ ]:

```
# Estimate parameters
mean = np.mean(ten_sample['length'])
std_dev = np.std(ten_sample['length'])
print(f"Mean of 'length': {mean}, Standard Deviation: {std_dev}")

# Hypothesis testing (e.g., testing mean)
# Example: Testing if mean is significantly different from population mean
t_stat, p_value = ttest_1samp(ten_sample['length'], 45.0737)
print(f"Mean T-statistic: {t_stat}, P-value: {p_value}")
if p_value < 0.05:
    print("Reject null hypothesis\n")
```

```
else:
  print("Fail to reject null hypothesis\n")
```

```
Mean of 'length': 43.80096, Standard Deviation: 59.36316711349781
Mean T-statistic: -11.74309390632738, P-value: 7.787233742908089e-32
Reject null hypothesis

Mean T-statistic: -139.61211312440756, P-value: 0.0
Reject null hypothesis
```

In [ ]:

```
# Estimate parameters
mean = np.mean(thirty_sample['length'])
std_dev = np.std(thirty_sample['length'])
print(f"Mean of 'length': {mean}, Standard Deviation: {std_dev}")

# Hypothesis testing (e.g., testing mean)
# Example: Testing if mean is significantly different from population mean
t_stat, p_value = ttest_1samp(thirty_sample['length'], 45.0737)
print(f"Mean T-statistic: {t_stat}, P-value: {p_value}")
if p_value < 0.05:
    print("Reject null hypothesis\n")
else:
  print("Fail to reject null hypothesis\n")
```

```
Mean of 'length': 44.98314, Standard Deviation: 59.60273043863343
Mean T-statistic: -0.48047200340394564, P-value: 0.6308928569981553
Fail to reject null hypothesis

Mean T-statistic: -74.00871133173877, P-value: 0.0
Reject null hypothesis
```

In [ ]:

```
contingency_table = pd.crosstab(df['length'], thirty_sample['length'])
chi2, p, _, _ = chi2_contingency(contingency_table)
print(f"Chi-square test for independence between population standard deviation and 30
print(f"Chi-square statistic: {chi2}, P-value: {p}")
if p_value < 0.05:
    print("Reject null hypothesis\n")
else:
  print("Fail to reject null hypothesis\n")
```

```
Chi-square test for independence between population standard deviation and sample sta
ndard deviation:
Chi-square statistic: 7200000.0, P-value: 0.0
Reject null hypothesis
```

In [ ]:

```
contingency_table = pd.crosstab(df['length'], ten_sample['length'])
chi2, p, _, _ = chi2_contingency(contingency_table)
print(f"Chi-square test for independence between population standard deviation and 10
print(f"Chi-square statistic: {chi2}, P-value: {p}")
if p_value < 0.05:
    print("Reject null hypothesis\n")
else:
  print("Fail to reject null hypothesis\n")
```

```
Chi-square test for independence between population standard deviation and 10 sample
standard deviation:
Chi-square statistic: 21600000.0, P-value: 0.0
Reject null hypothesis
```

In [ ]:
```python
import scipy.stats as stats


sample_mean = np.mean(df['length'])
population_mean = np.mean(df['weight'])
population_std = np.std(df['length'])
sample_size = len(df['length'])
alpha = 0.05

# compute the z-score
z_score = (sample_mean-population_mean)/(population_std/np.sqrt(sample_size))
print('Z-Score :',z_score)

# Approach 1: Using Critical Z-Score

# Critical Z-Score
z_critical = stats.norm.ppf(1-alpha)
print('Critical Z-Score :',z_critical)

# Hypothesis
if abs(z_score) >  z_critical:
    print("Reject Null Hypothesis")
else:
  print("Fail to Reject Null Hypothesis")
```

```
Z-Score : 1287.4282996726054
Critical Z-Score : 1.6448536269514722
Reject Null Hypothesis
```

In [ ]:
```python
from sklearn.linear_model import LinearRegression
X = df[['length', 'weight', 'count', 'neighbors']]
y = df[['label']]

reg = LinearRegression().fit(X,y)
```

In [ ]:
```python
reg.coef_
```

Out[ ]:
```
array([[ 2.41061393e-06, -2.03308570e-04,  5.47664344e-07,
         3.30431853e-05]])
```

In [ ]:
```python
reg.intercept_
```

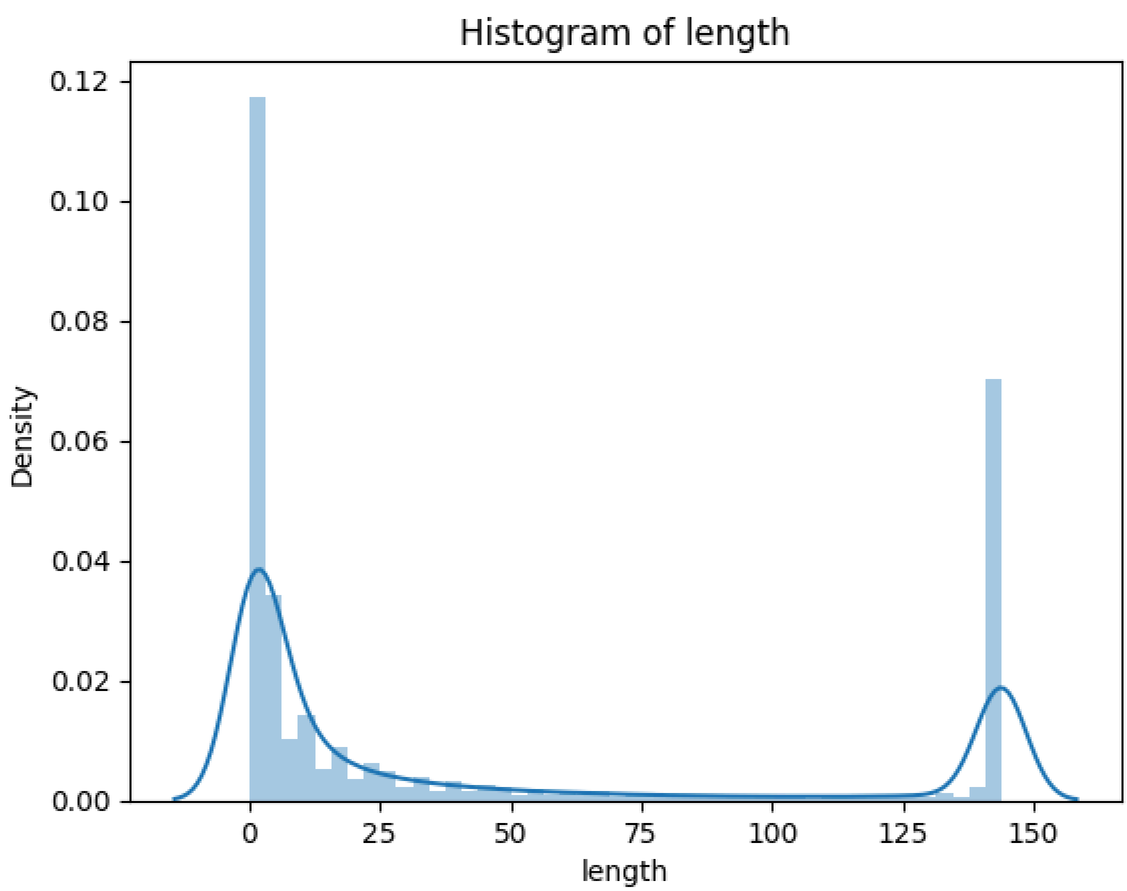
Out[ ]:
```
array([0.98533569])
```

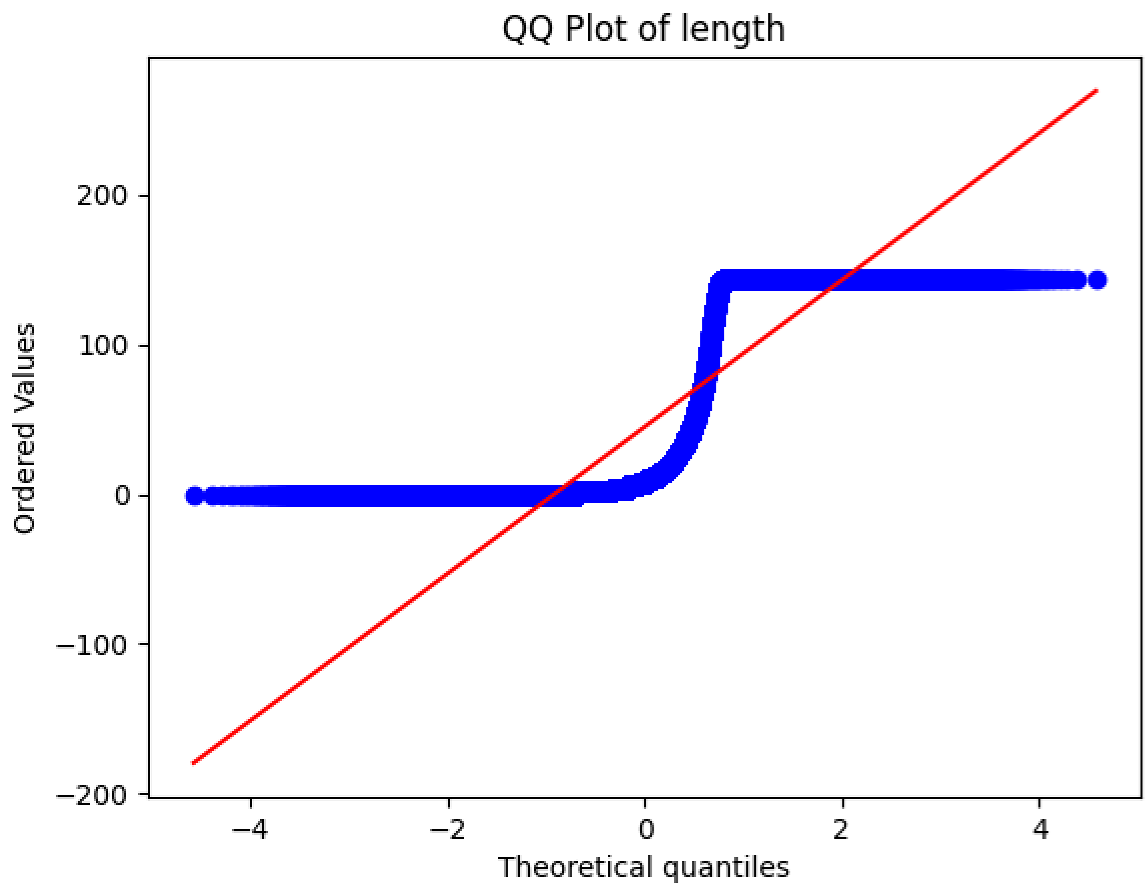
In [ ]:
```python
from scipy.stats import norm, chi2_contingency, ttest_ind, f_oneway, pearsonr, probplo
```

In [ ]:
```python
# Univariate Analysis
for column in ['length', 'weight', 'count', 'neighbors']:
    # Histogram
    sns.distplot(sampled_df[column], kde=True)
    plt.title(f'Histogram of {column}')
    plt.show()
```

```
    # QQ Plot
    probplot(sampled_df[column], dist="norm", plot=plt)
    plt.title(f'QQ Plot of {column}')
    plt.show()

    # Estimate parameters
    mean = np.mean(sampled_df[column])
    std_dev = np.std(sampled_df[column])
    print(f"Mean of {column}: {mean}, Standard Deviation: {std_dev}")

    # Hypothesis testing (e.g., testing mean)
    # Example: Testing if mean is significantly different from 0
    t_stat, p_value = ttest_1samp(sampled_df[column], 0)
    print(f"T-statistic: {t_stat}, P-value: {p_value}")
    if p_value < 0.05:
        print("Reject null hypothesis")
```

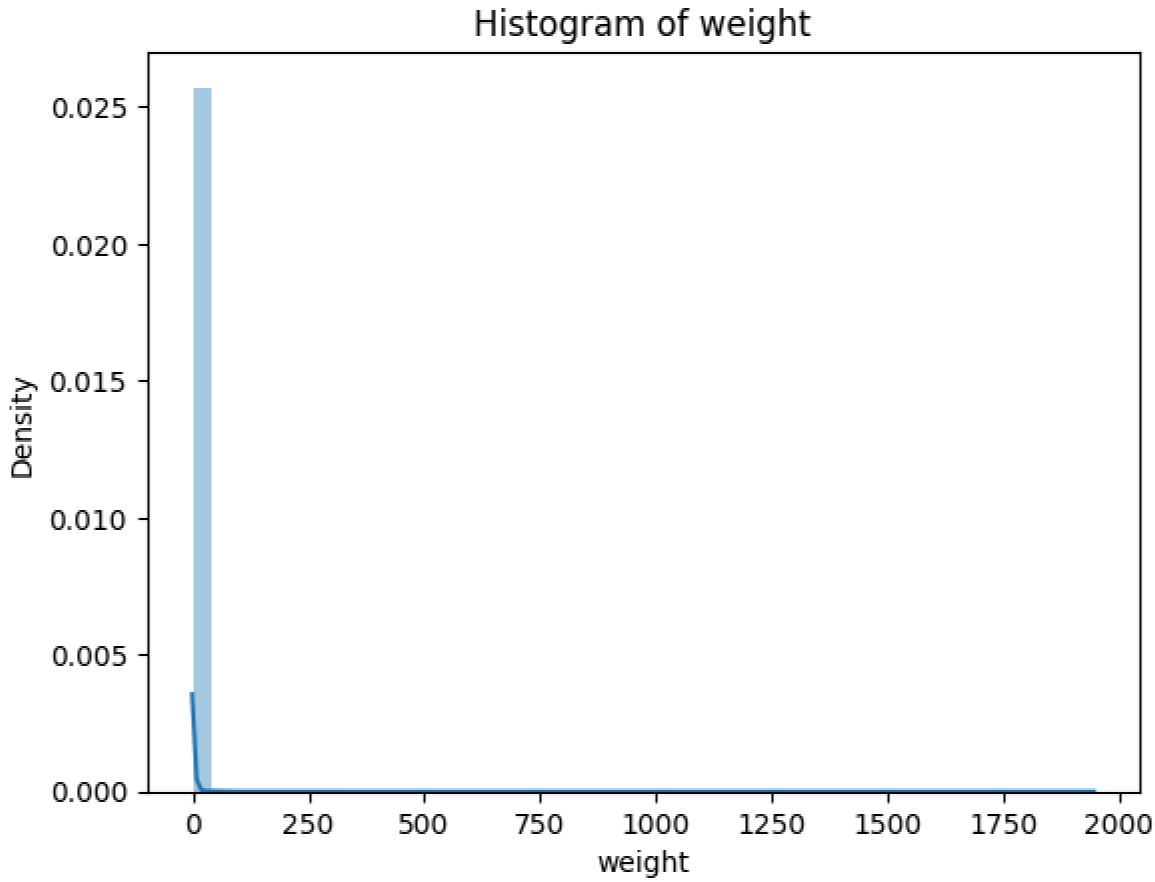
```
<ipython-input-14-a0ef6a1b089c>:4: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(sampled_df[column], kde=True)
```

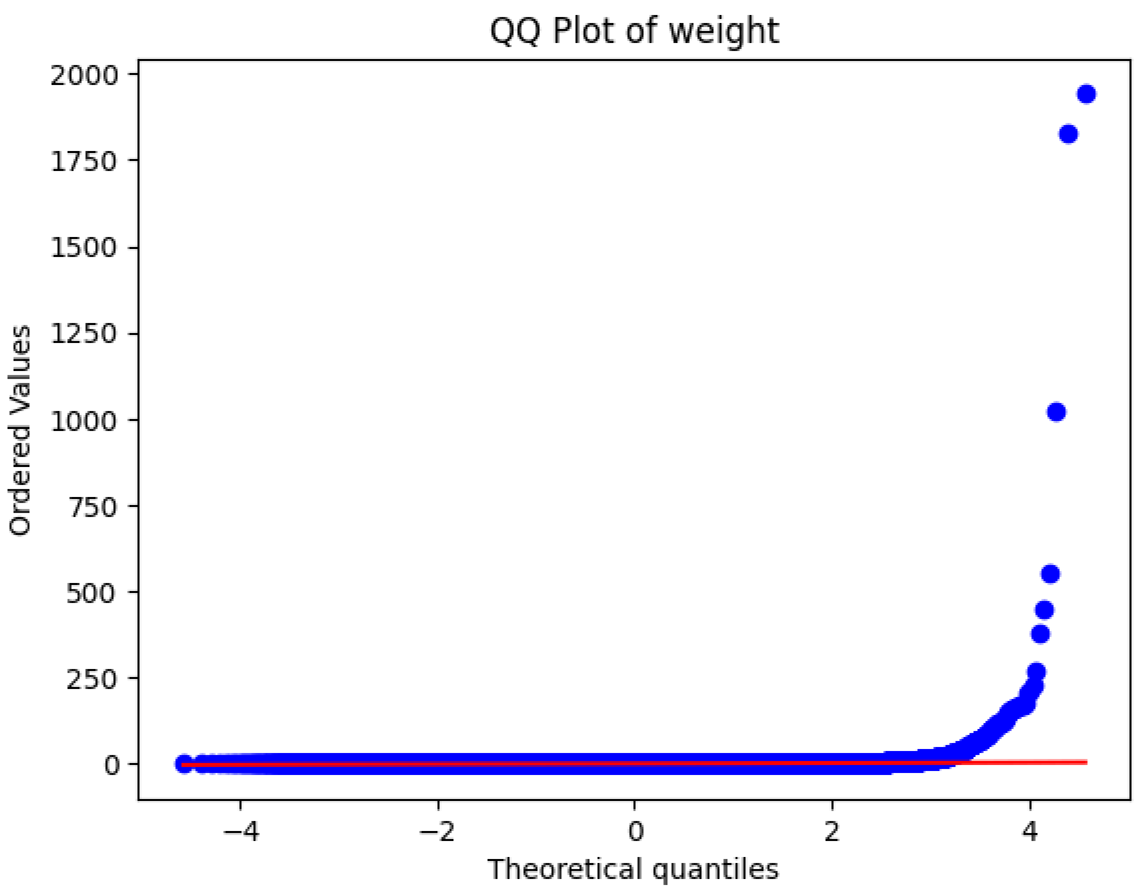





```
Mean of length: 44.83773442589228, Standard Deviation: 58.89026698383816
T-statistic: 411.1925840006627, P-value: 0.0
Reject null hypothesis
```

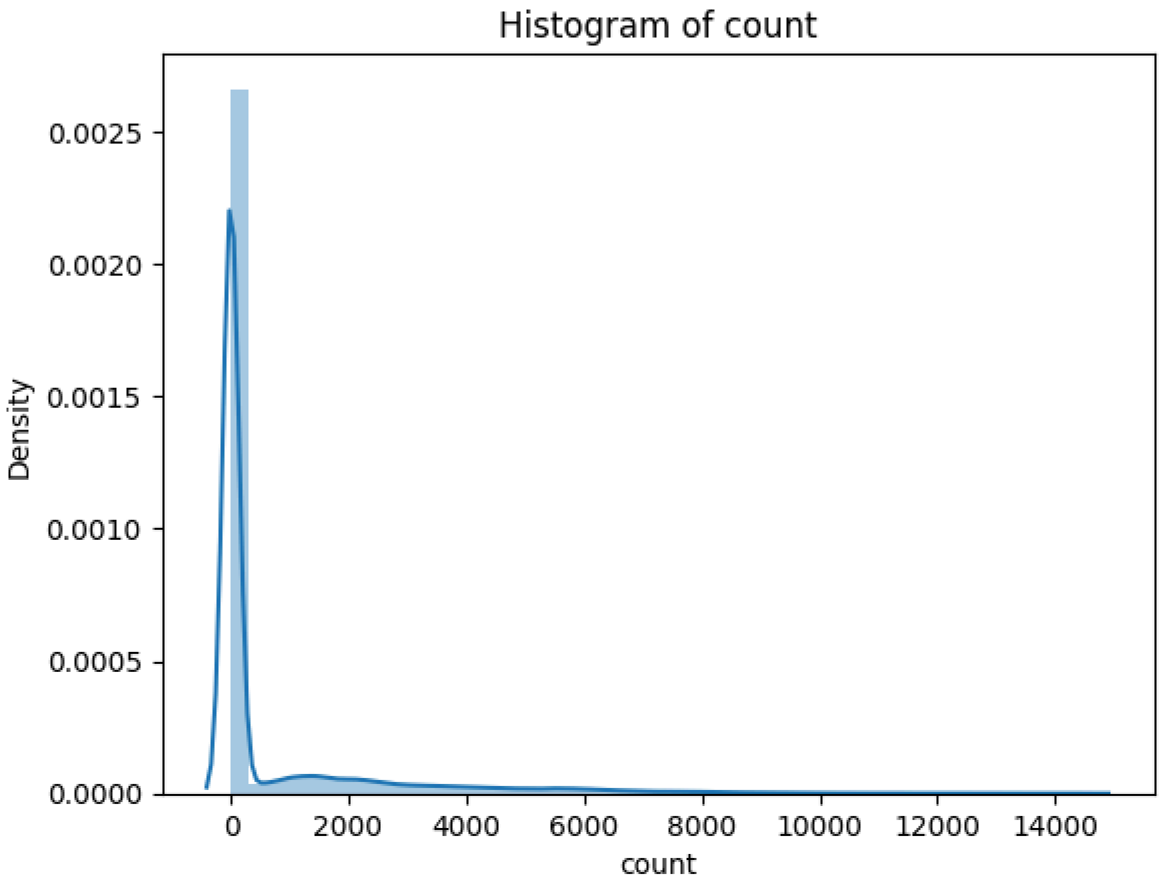
```
<ipython-input-14-a0ef6a1b089c>:4: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(sampled_df[column], kde=True)
```

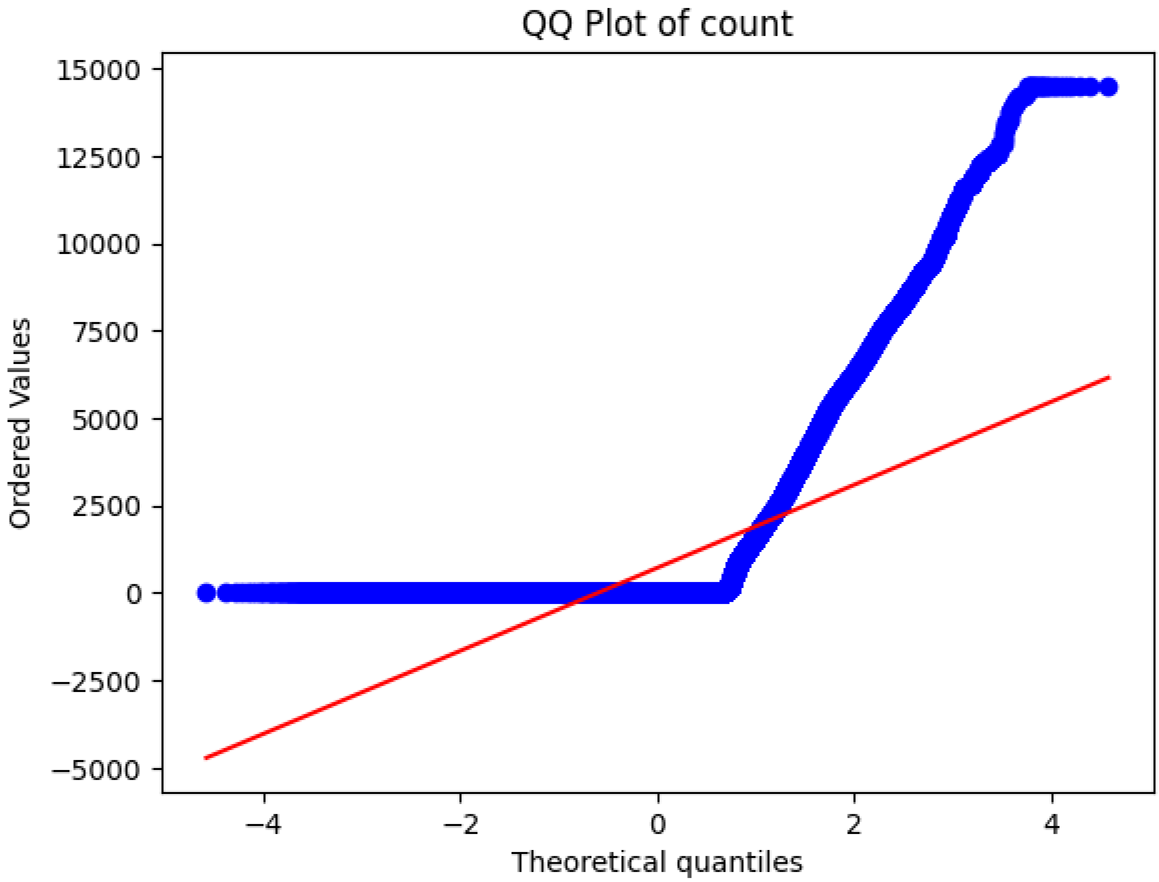





```
Mean of weight: 0.5505137372835565, Standard Deviation: 5.921078321079643
T-statistic: 50.21257474424304, P-value: 0.0
Reject null hypothesis
```

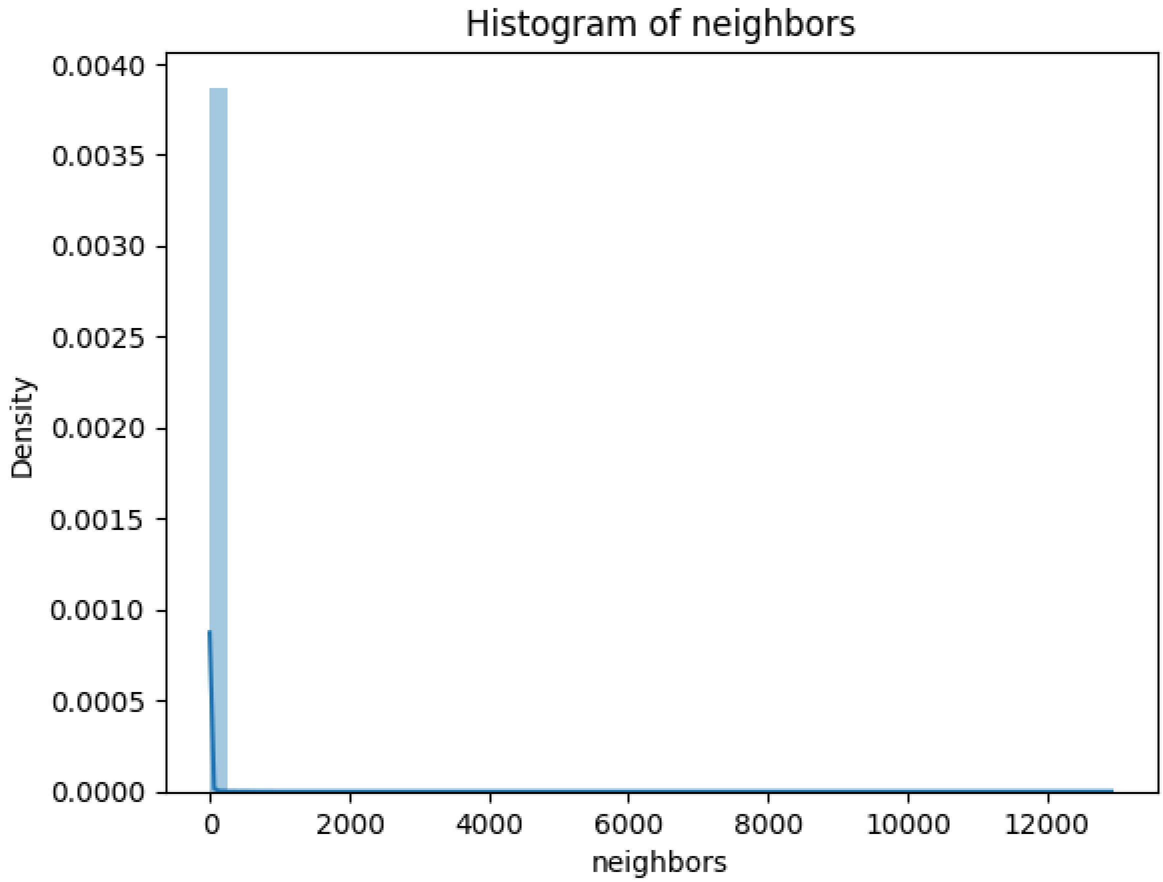
```
<ipython-input-14-a0ef6a1b089c>:4: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(sampled_df[column], kde=True)
```

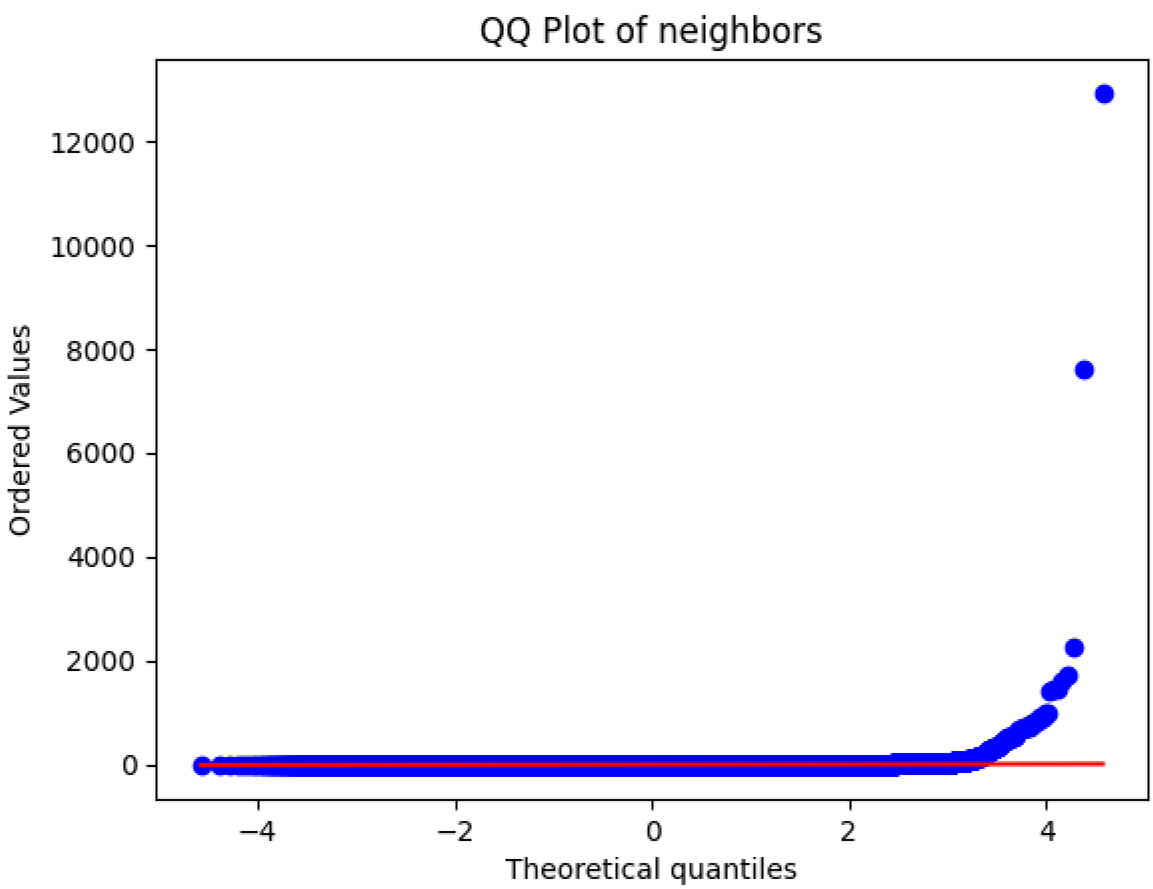





```
Mean of count: 715.637604141667, Standard Deviation: 1681.5877664602278
T-statistic: 229.83636806365342, P-value: 0.0
Reject null hypothesis
```

```
<ipython-input-14-a0ef6a1b089c>:4: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(sampled_df[column], kde=True)
```






```
Mean of neighbors: 2.215658106764494, Standard Deviation: 30.36086081688695
T-statistic: 39.412483474412, P-value: 0.0
Reject null hypothesis
```

Interpreting the results:

- Histogram: The shape of the histogram helps understand the data distribution. A bell-shaped curve suggests a normal distribution, while skewed shapes indicate non-normality.
- QQ Plot: Points close to the diagonal line suggest the data follows a normal distribution. Deviations from the line indicate non-normality.
- Mean and Standard Deviation: These values provide a summary of the central tendency and spread of the data.
- Hypothesis Testing: The t-statistic measures the difference between the sample mean and the hypothesized mean (0 in this case). The p-value indicates the probability of observing such a difference by random chance. If the p-value is less than 0.05, we reject the null hypothesis and conclude that the mean is significantly different from 0.

Multivariate Analysis

We conduct a chi-square test of independence between two categorical variables ('income' and 'label').

1. We create a contingency table using `pd.crosstab()` to show the frequency counts of each combination of categories between the two variables.

2. We then perform the chi-square test using `chi2_contingency()` from the `scipy.stats` module. This test helps us determine if there is a significant association between the two categorical variables. The test results in a chi-square statistic and a p-value.

3. The chi-square statistic measures the discrepancy between the observed and expected frequencies in the contingency table. The p-value indicates the probability of observing such a discrepancy by random chance.

4. If the p-value is less than 0.05 (commonly chosen significance level), we reject the null hypothesis, which suggests that there is a significant association between the two categorical variables.

In [ ]:
```
# Categorical vs Categorical: Chi-square test of independence
categorical_columns = ['label']  # Assuming 'label' is the categorical variable

contingency_table = pd.crosstab(df['income'], df[column])
chi2, p, _, _ = chi2_contingency(contingency_table)
print(f"Chi-square test for independence between 'income' and '{column}':")
print(f"Chi-square statistic: {chi2}, P-value: {p}")
if p < 0.05:
    print("Reject null hypothesis")
```

Interpreting the results:

- Chi-square statistic: A larger value indicates a greater discrepancy between the observed and expected frequencies, suggesting a stronger association between the variables.

- P-value: If the p-value is less than 0.05, we reject the null hypothesis and conclude that there is a significant association between the variables.
- Conclusion: Based on the p-value, we can determine if there is a significant relationship between the two categorical variables. If the null hypothesis is rejected, we can conclude that there is evidence of an association between 'income' and 'label'.

We perform independent t-tests to compare the means of numerical variables ('length', 'weight', 'count', 'looped', 'neighbors', 'income') between two groups defined by a categorical variable ('label').

1. For each numerical column, we split the data into two groups based on the categories in the 'label' column (assumed to be binary, with 0 and 1).

2. We then calculate the t-statistic and p-value using the `ttest_ind` function from `scipy.stats` to determine if there is a significant difference in the means of the two groups for each numerical variable.

3. The t-statistic measures the difference between the means of the two groups relative to the variance in the data. The p-value indicates the probability of observing such a difference by random chance.

4. If the p-value is less than 0.05 (commonly chosen significance level), we reject the null hypothesis and conclude that there is a significant difference in the means of the two groups for that numerical variable.

In [ ]:
```
# Categorical vs Numerical: T-test
categorical_columns = ['label']  # Assuming 'label' is the categorical variable
numerical_columns = ['length', 'weight', 'count', 'neighbors']
for cat_col in categorical_columns:
    for num_col in numerical_columns:
        group1 = sampled_df[sampled_df[cat_col] == 0][num_col]
        group2 = sampled_df[sampled_df[cat_col] == 1][num_col]
        t_stat, p_value = ttest_ind(group1, group2)
        print(f"T-test for '{num_col}' and '{cat_col}':")
        print(f"T-statistic: {t_stat}, P-value: {p_value}")
        if p_value < 0.05:
            print("Reject null hypothesis\n")
        else:
          print("Fail to reject null hypothesis\n")
```

```
T-test for 'length' and 'label':
T-statistic: -2.3758356634709794, P-value: 0.017509901031010345
Reject null hypothesis

T-test for 'weight' and 'label':
T-statistic: 0.3943609569198237, P-value: 0.6933148531783684
Fail to reject null hypothesis

T-test for 'count' and 'label':
T-statistic: -4.5447109981816265, P-value: 5.503271072713147e-06
Reject null hypothesis

T-test for 'neighbors' and 'label':
T-statistic: -0.31529196433179285, P-value: 0.75254021120546
Fail to reject null hypothesis
```

Interpreting the results:

- T-statistic: A larger absolute value of the t-statistic indicates a greater difference between the means of the two groups.
- P-value: If the p-value is less than 0.05, we reject the null hypothesis and conclude that there is a significant difference in the means of the two groups for that numerical variable.
- Conclusion: Based on the p-value, we can determine if there is a significant difference in the means of the two groups for each numerical variable.

We calculate the Pearson correlation coefficient between pairs of numerical variables ('length', 'weight', 'count', 'looped', 'neighbors', 'income'). The Pearson correlation coefficient measures the strength and direction of a linear relationship between two variables.

1. For each pair of numerical columns, we calculate the Pearson correlation coefficient using the `pearsonr` function from `scipy.stats`.

2. The correlation coefficient ranges from -1 to 1, where:

   - 1 indicates a perfect positive linear relationship,
   - -1 indicates a perfect negative linear relationship,
   - 0 indicates no linear relationship.
3. The p-value associated with the correlation coefficient indicates the probability of observing such a correlation by random chance when the true correlation is zero.

4. If the p-value is less than 0.05 (commonly chosen significance level), we reject the null hypothesis and conclude that there is a significant linear relationship between the two variables.

In [ ]:
```
# Numerical vs Numerical: Pearson correlation coefficient
numerical_columns = ['length', 'weight', 'count','neighbors']
for i, col1 in enumerate(numerical_columns):
    for col2 in numerical_columns[i+1:]:
        corr_coef, p_value = pearsonr(sampled_df[col1], sampled_df[col2])
        print(f"Pearson correlation coefficient between '{col1}' and '{col2}': {corr_c
        if p_value < 0.05:
```

```
            print("Reject null hypothesis\n")
        else:
          print("Fail to reject null hypothesis\n")
```

```
Pearson correlation coefficient between 'length' and 'weight': 0.0022689148668866973,
P-value: 0.2204405886089042
Fail to reject null hypothesis

Pearson correlation coefficient between 'length' and 'count': 0.7029921281288343, P-v
alue: 0.0
Reject null hypothesis

Pearson correlation coefficient between 'length' and 'neighbors': 0.01933057429855051
4, P-value: 1.6162532410767028e-25
Reject null hypothesis

Pearson correlation coefficient between 'weight' and 'count': 0.018776964888342396, P
-value: 3.610298988030419e-24
Reject null hypothesis

Pearson correlation coefficient between 'weight' and 'neighbors': 0.86021122364104, P
-value: 0.0
Reject null hypothesis

Pearson correlation coefficient between 'count' and 'neighbors': 0.01705740179857954,
P-value: 3.1792456346556725e-20
Reject null hypothesis
```

Interpreting the results:

- Correlation coefficient:
  - A coefficient close to 1 or -1 indicates a strong linear relationship.
  - A coefficient close to 0 indicates no linear relationship.
  - The sign indicates the direction of the relationship (positive or negative).
- P-value:
  - If the p-value is less than 0.05, we reject the null hypothesis and conclude that there is a significant linear relationship between the two variables.
  - If the p-value is greater than or equal to 0.05, we fail to reject the null hypothesis, suggesting no significant linear relationship.
- Conclusion: Based on the correlation coefficient and p-value, we can determine if there is a significant linear relationship between each pair of numerical variables.

In [ ]: