basic-stats-1-assignment

April 1, 2024

[6]:
```
# Loading necessary libraries

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy.stats as stats
import scipy.stats as t
import seaborn as sn

from scipy.stats import skew, kurtosis
from scipy.stats import t
```

0.1 Question 6

[180]:
```
def expected_value(values, weights):
    values = np.asarray(values)
    weights = np.asarray(weights)
    return (values * weights).sum() / weights.sum()

c_count = [1,4,3,5,6,2]
ch_prob = [0.015,0.20,0.65,0.005,0.01,0.120]
expected_value(c_count, ch_prob)
```

[180]:
```
3.09
```

0.2 Question 7 Soln

[150]:
```
# Load the CSV file into a DataFrame
q7 = pd.read_csv('q7.csv')
q7.head()
```

[150]:

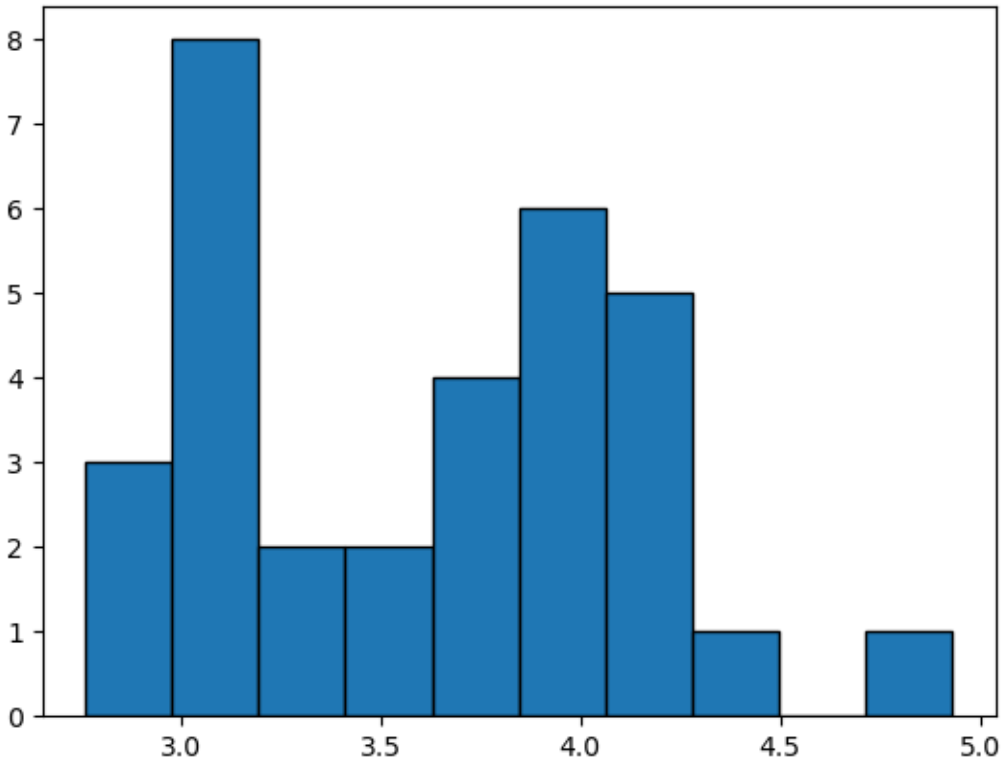
	Cars	Points	Score	Weigh
0	Mazda RX4	3.90	2.620	16.46
1	Mazda RX4 Wag	3.90	2.875	17.02
2	Datsun 710	3.85	2.320	18.61
3	Hornet 4 Drive	3.08	3.215	19.44
4	Hornet Sportabout	3.15	3.440	17.02

[151]:
```
# For Points series
points_mean = q7['Points'].mean()
points_median = q7['Points'].median()
points_mode = q7['Points'].mode()[0]
points_variance = q7['Points'].var()
points_std_dev = q7['Points'].std()
points_range = np.ptp(q7['Points'])

print("Points:")
print("Mean:", points_mean)
print("Median:", points_median)
print("Mode:", points_mode)
print("Variance:", points_variance)
print("Standard Deviation:", points_std_dev)
print("Range:", points_range)
print()

# For Score Series
score_mean = q7['Score'].mean()
score_median = q7['Score'].median()
score_mode = q7['Score'].mode()[0]
score_variance = q7['Score'].var()
score_std_dev = q7['Score'].std()
score_range = np.ptp(q7['Score'])

print("Score:")
print("Mean:", score_mean)
print("Median:", score_median)
print("Mode:", score_mode)
print("Variance:", score_variance)
print("Standard Deviation:", score_std_dev)
print("Range:", score_range)
print()

# For Weigh Series
weigh_mean = q7['Weigh'].mean()
weigh_median = q7['Weigh'].median()
weigh_mode = q7['Weigh'].mode()[0]
weigh_variance = q7['Weigh'].var()
weigh_std_dev = q7['Weigh'].std()
weigh_range = np.ptp(q7['Weigh'])

print("Weigh:")
print("Mean:", weigh_mean)
print("Median:", weigh_median)
print("Mode:", weigh_mode)
print("Variance:", weigh_variance)
```

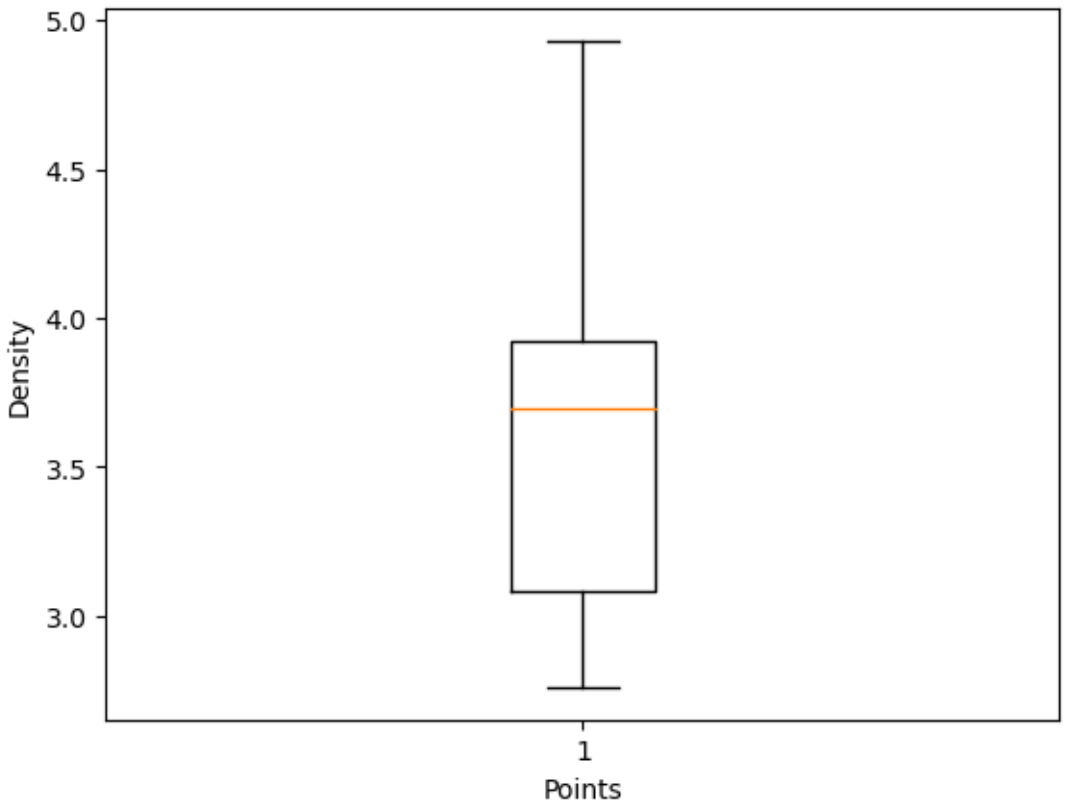
```
print("Standard Deviation:", weigh_std_dev)
print("Range:", weigh_range)
```

```
Points:
Mean: 3.5965625
Median: 3.6950000000000003
Mode: 3.07
Variance: 0.28588135080645166
Standard Deviation: 0.5346787360709716
Range: 2.17

Score:
Mean: 3.2172500000000004
Median: 3.325
Mode: 3.44
Variance: 0.9573789677419356
Standard Deviation: 0.9784574429896967
Range: 3.9110000000000005

Weigh:
Mean: 17.848750000000003
Median: 17.71
Mode: 17.02
Variance: 3.193166129032258
Standard Deviation: 1.7869432360968431
Range: 8.399999999999999
```

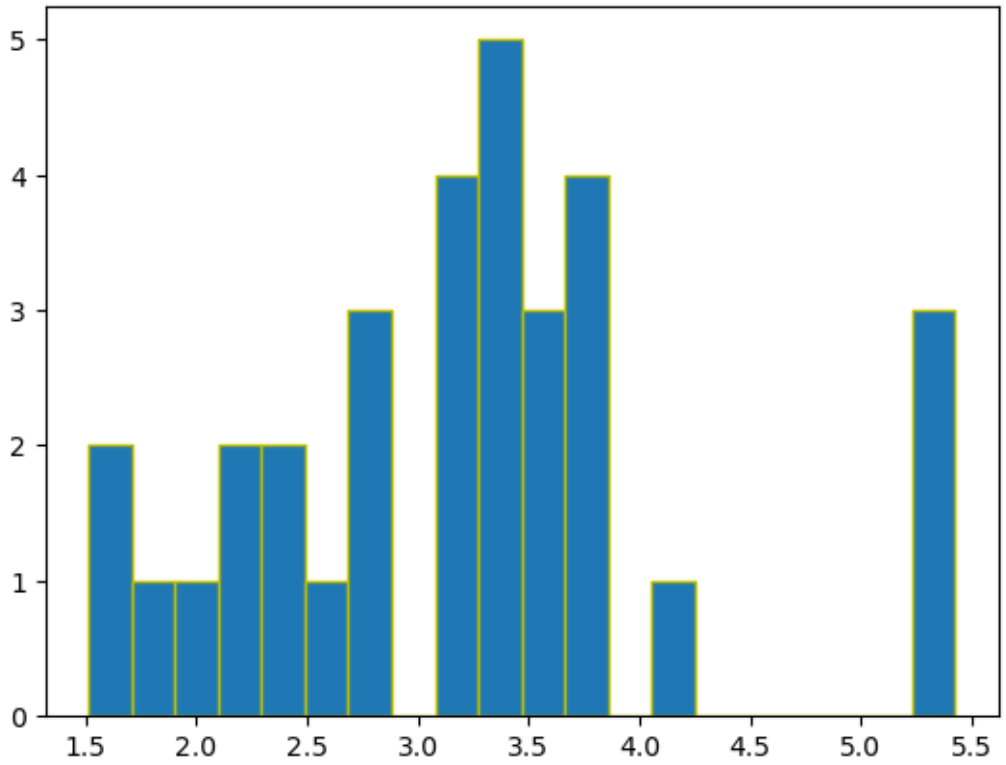
[181]:
```
# Points
plt.hist(q7["Points"], bins = 10, edgecolor= 'black')
plt.show()
```



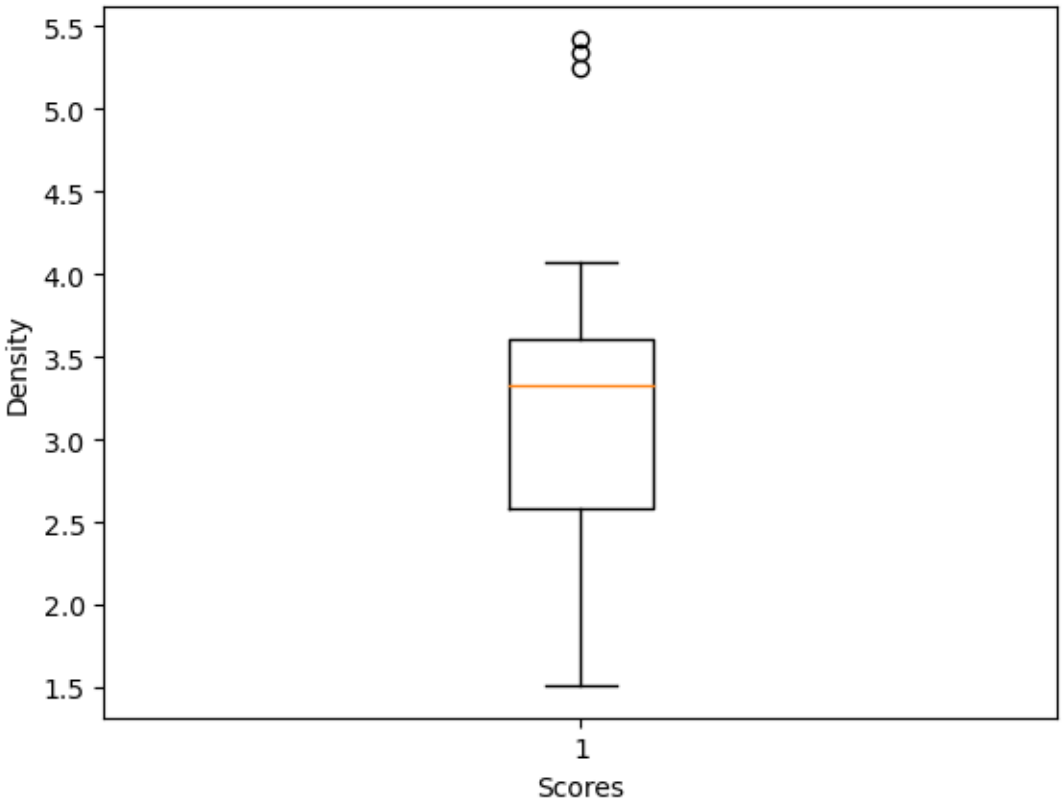
[153]:
```
plt.boxplot(x = 'Points', data =q7)
plt.xlabel('Points')
plt.ylabel('Density')
plt.savefig("PointsInferences.png")
plt.show()
```



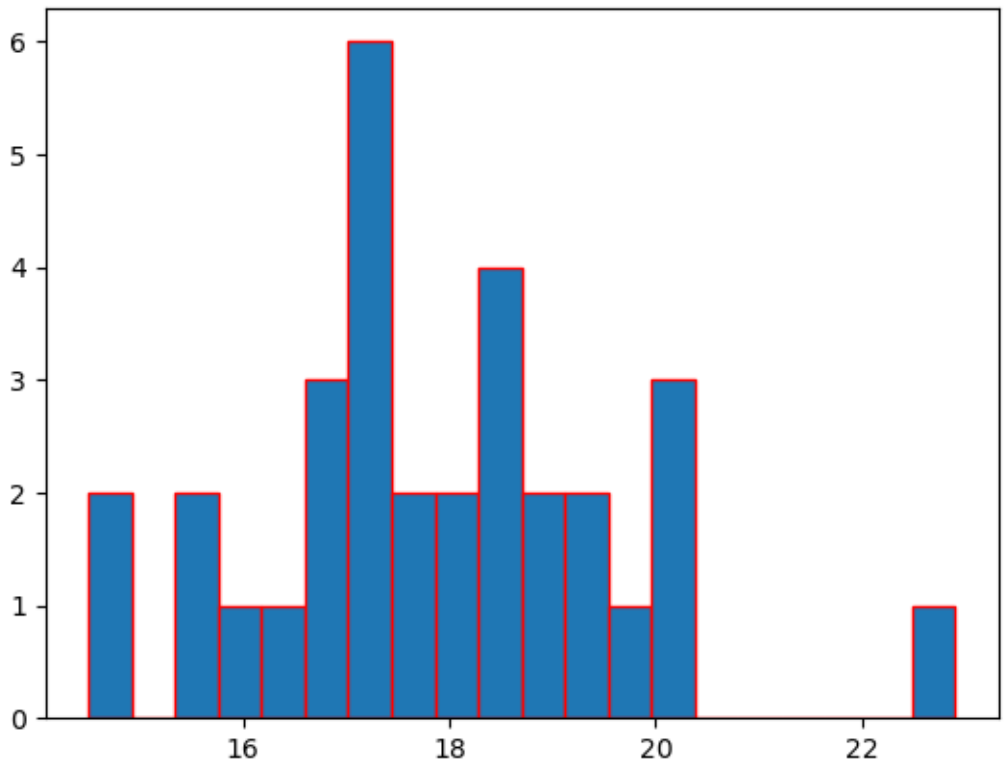
[182]:
```
# Score
plt.hist(q7["Score"], bins = 20, edgecolor = 'y')
plt.show()
```



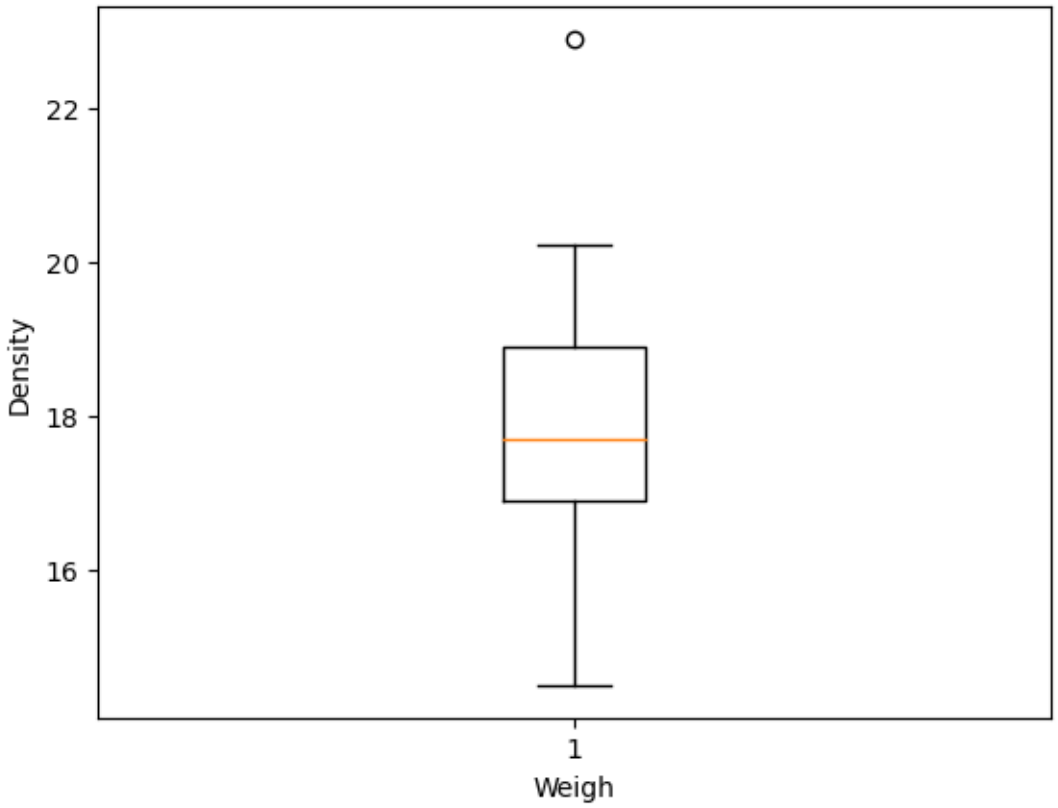
[155]:
```
plt.boxplot(x = 'Score', data= q7)
plt.xlabel('Scores')
plt.ylabel('Density')
plt.savefig("ScoresInferences.png")
plt.show()
```



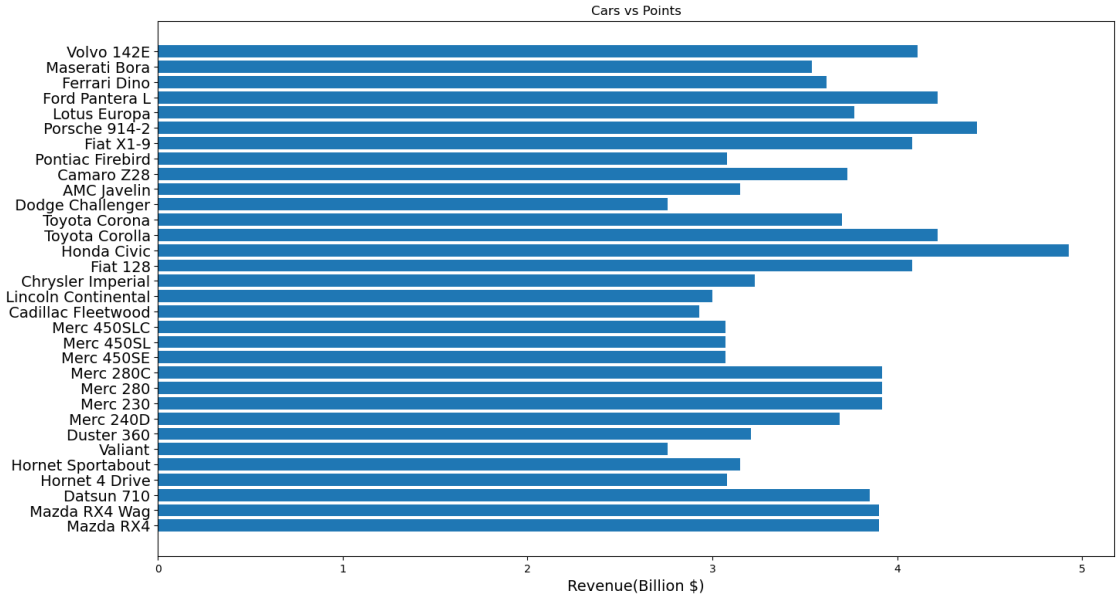
[183]:
```
# Weigh
plt.hist(q7["Weigh"], bins=20, edgecolor = 'red')
plt.show()
```



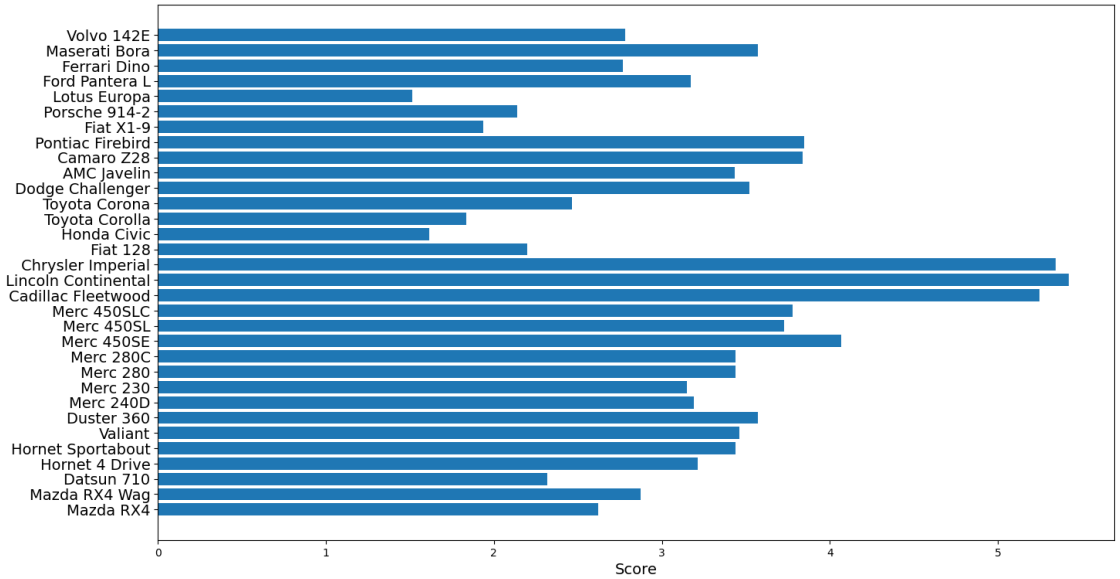
[157]:
```
plt.boxplot(x= "Weigh", data = q7)
plt.xlabel('Weigh')
plt.ylabel('Density')
plt.savefig("WeighInferences.png")
plt.show()
```



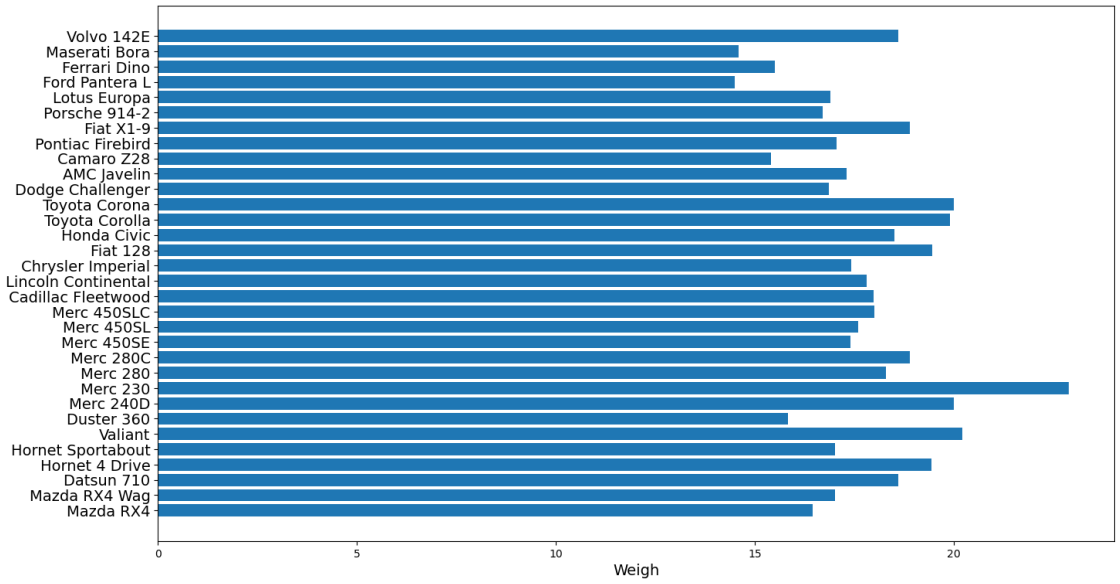
[187]:
```
# Cars vs Points
plt.figure(figsize=(16,9))
plt.barh(q7["Cars"], q7["Points"])
plt.xlabel('Revenue(Billion $)', fontsize=14)
plt.yticks(fontsize=14)
plt.show()
```



[188]:
```
# Cars vs Score
plt.figure(figsize=(16,9))
plt.barh(q7["Cars"], q7["Score"])
plt.xlabel('Score', fontsize=14)
plt.yticks(fontsize=14)
plt.show()
```



[189]:
```
# Cars vs Weigh
plt.figure(figsize=(16,9))
plt.barh(q7["Cars"], q7["Weigh"])
plt.xlabel('Weigh', fontsize=14)
plt.yticks(fontsize=14)
plt.show()
```



0.3 Question 8 Soln

[193]:
```
weigh = [108,110,123,134,135,145,167,187,199]
probs = [1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9]
round(expected_value(weigh, probs), 2)
```

[193]: 145.33

0.4 Question 9(a): Soln

[162]:
```
# Load the CSV file into a DataFrame
q9a = pd.read_csv('q9_a.csv')
q9a.head()
```

[162]:

	cars	speed	dist
0	1	4	2
1	2	4	10
2	3	7	4
3	4	7	22
4	5	8	16

[163]:
```
# Calculate skewness and kurtosis for speed and distance series
speed_skewness = skew(q9a['speed'])
speed_kurtosis = kurtosis(q9a['speed'])

distance_skewness = skew(q9a['dist'])
distance_kurtosis = kurtosis(q9a['dist'])

print("Speed Skewness:", speed_skewness)
print("Speed Kurtosis:", speed_kurtosis)
print("Distance Skewness:", distance_skewness)
print("Distance Kurtosis:", distance_kurtosis)
```

```
Speed Skewness: -0.11395477012828319
Speed Kurtosis: -0.5771474239437371
Distance Skewness: 0.7824835173114966
Distance Kurtosis: 0.24801865717051808
```

0.5 Question 9(b): Soln

[164]:
```
# Load the CSV file into a DataFrame
q9b = pd.read_csv('q9_b.csv')
q9b.head()
```

[164]:

	Unnamed: 0	SP	WT
0	1	104.185353	28.762059
1	2	105.461264	30.466833
2	3	105.461264	30.193597
3	4	113.461264	30.632114
4	5	104.461264	29.889149

[165]:
```
# Calculate skewness and kurtosis for SP and WT series
sp_skewness = skew(q9b['SP'])
sp_kurtosis = kurtosis(q9b['SP'])

wt_skewness = skew(q9b['WT'])
wt_kurtosis = kurtosis(q9b['WT'])

print("Speed Skewness:", sp_skewness)
print("Speed Kurtosis:", sp_kurtosis)
print("Distance Skewness:", wt_skewness)
print("Distance Kurtosis:", wt_kurtosis)
```

```
Speed Skewness: 1.5814536794423764
Speed Kurtosis: 2.7235214865269244
Distance Skewness: -0.6033099322115126
Distance Kurtosis: 0.8194658792266849
```

0.6 Question 11 Soln

[166]:
```
# Sample statistics
sample_mean = 200  # sample mean
sample_std = 30    # sample standard deviation
sample_size = 2000 # sample size

# Population statistics
population_size = 3000000 # population size

# Degrees of freedom
df = sample_size - 1

# Calculate the standard error of the mean
standard_error = sample_std / (sample_size ** 0.5)

# Calculate the margin of error for different confidence levels
confidence_levels = [0.94, 0.98, 0.96]
confidence_intervals = {}

for confidence_level in confidence_levels:
    # Calculate the critical value
    alpha = 1 - confidence_level
    z_critical = stats.t.ppf(1 - alpha / 2, df)

    # Calculate the margin of error
    margin_of_error = z_critical * standard_error

    # Calculate the confidence interval
    lower_bound = sample_mean - margin_of_error
    upper_bound = sample_mean + margin_of_error

    confidence_intervals[confidence_level] = (lower_bound, upper_bound)

# Print the confidence intervals
for confidence_level, interval in confidence_intervals.items():
    print(f"{confidence_level*100}% Confidence Interval: ({interval[0]},␣
 ↪{interval[1]})")
```

```
94.0% Confidence Interval: (198.7376089443071, 201.2623910556929)
98.0% Confidence Interval: (198.4381860483216, 201.5618139516784)
96.0% Confidence Interval: (198.6214037429732, 201.3785962570268)
```

0.7 Question 12 Soln

[167]:
```
scores = [34,36,36,38,38,39,39,40,40,41,41,41,41,42,42,45,49,56]
scores
```

[167]:
```
[34, 36, 36, 38, 38, 39, 39, 40, 40, 41, 41, 41, 41, 42, 42, 45, 49, 56]
```

[194]:
```
print("Mean: ", np.mean(scores))
print("Median: ", np.median(scores))
print("Variance: ", np.var(scores))
print("Standard Deviation: ", np.std(scores))
```

```
Mean:  41.0
Median:  40.5
Variance:  24.11111111111111
Standard Deviation:  4.910306620885412
```

0.8 Question 20 Soln

[7]:
```
# Load the CSV file into a DataFrame
q20 = pd.read_csv('Cars.csv')
q20.head()
```

[7]:

	HP	MPG	VOL	SP	WT
0	49	53.700681	89	104.185353	28.762059
1	55	50.013401	92	105.461264	30.466833
2	55	50.013401	92	105.461264	30.193597
3	70	45.696322	92	113.461264	30.632114
4	53	50.504232	92	104.461264	29.889149

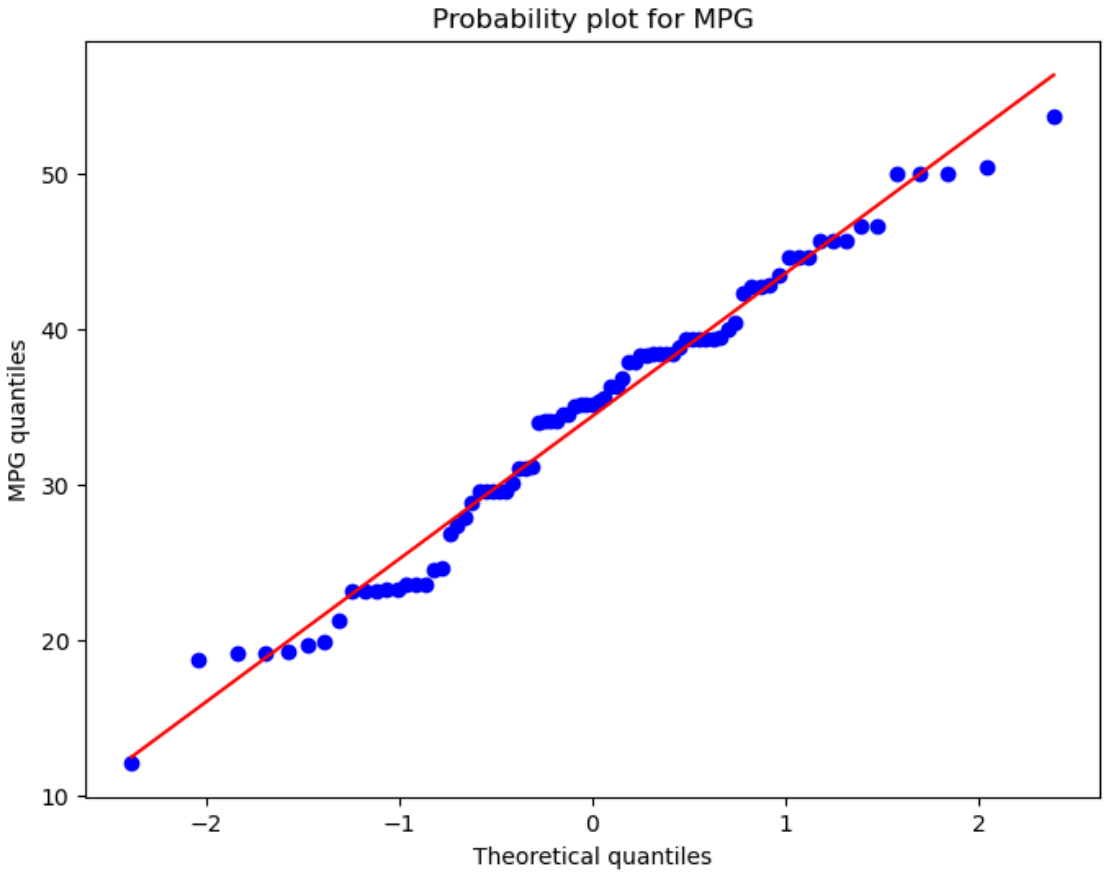
[30]:
```
import pandas as pd
from scipy.stats import norm

# Load the dataset
df = pd.read_csv('Cars.csv')

# Calculate the mean and standard deviation of MPG
mean_mpg = df['MPG'].mean()
std_dev_mpg = df['MPG'].std()

# Calculate the probabilities using the normal distribution
# a. P(MPG > 38)
p_a = 1 - norm.cdf(38, loc=mean_mpg, scale=std_dev_mpg)

# b. P(MPG < 40)
p_b = norm.cdf(40, loc=mean_mpg, scale=std_dev_mpg)

# c. P(20 < MPG < 50)
```

```
p_c = norm.cdf(50, loc=mean_mpg, scale=std_dev_mpg) - norm.cdf(20,␣
 ↪loc=mean_mpg, scale=std_dev_mpg)

print(f"Probability of MPG > 38: {round(p_a,3)}")
print(f"Probability of MPG < 40: {round(p_b,3)}")
print(f"Probability of 20 < MPG < 50: {round(p_c,3)}")
```

```
Probability of MPG > 38: 0.348
Probability of MPG < 40: 0.729
Probability of 20 < MPG < 50: 0.899
```

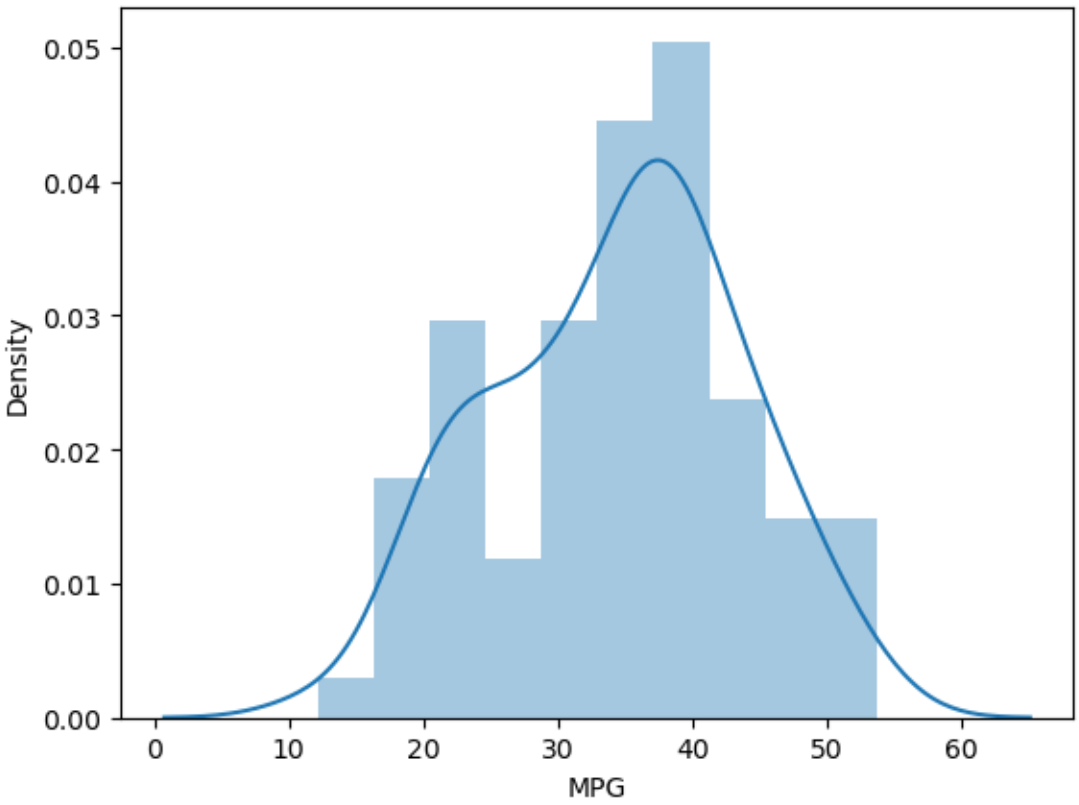
0.9 Question 21 Soln

[171]:
```
#Q 21 a)

plt.figure(figsize=(8, 6))
stats.probplot(q20['MPG'], dist="norm", plot=plt)
plt.title('Probability plot for MPG')
plt.xlabel('Theoretical quantiles')
plt.ylabel('MPG quantiles')
plt.savefig('21_a.jpg')
plt.show()
```



[197]:
```
sn.distplot(q20['MPG'],kde=True, bins =10)
plt.show()
```

```
C:\Users\LENOVO\AppData\Local\Temp\ipykernel_7028\2696060157.py:2: 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

  sn.distplot(q20['MPG'],kde=True, bins =10)
```



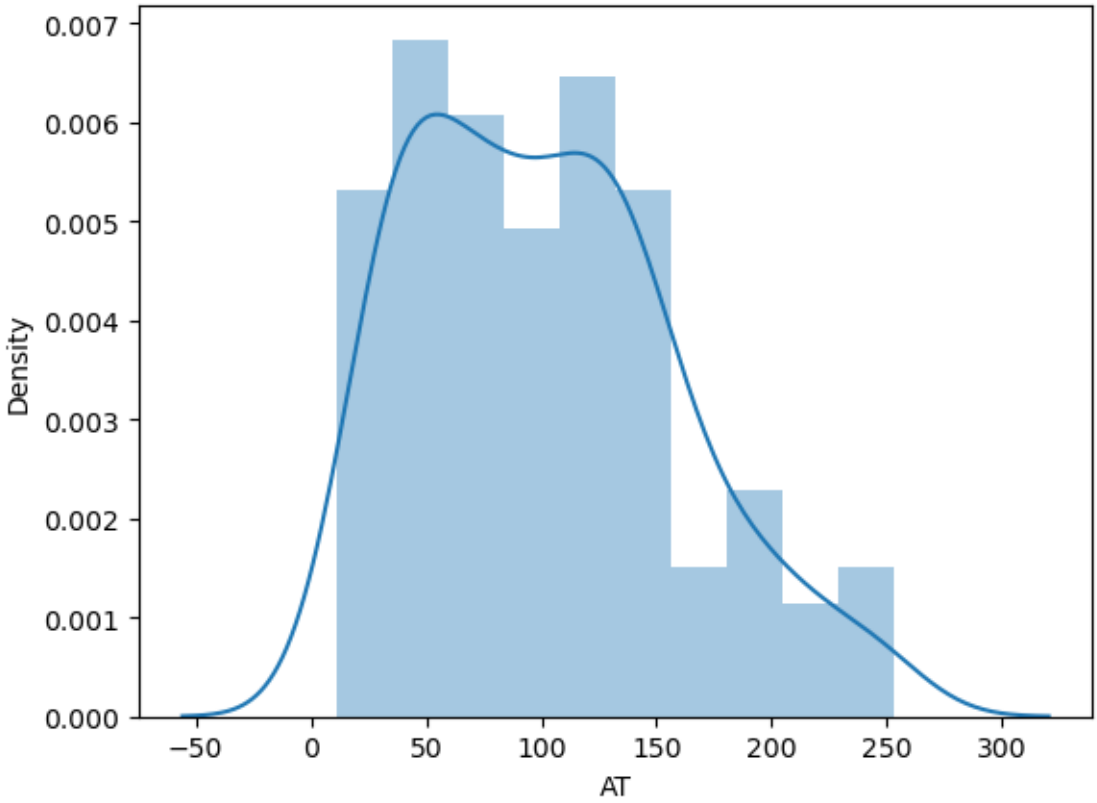
[173]:
```
#Q 21 b)
q21b = pd.read_csv('wc-at.csv')
q21b.head()
```

[173]:

	Waist	AT
0	74.75	25.72
1	72.60	25.89
2	81.80	42.60
3	83.95	42.80
4	74.65	29.84

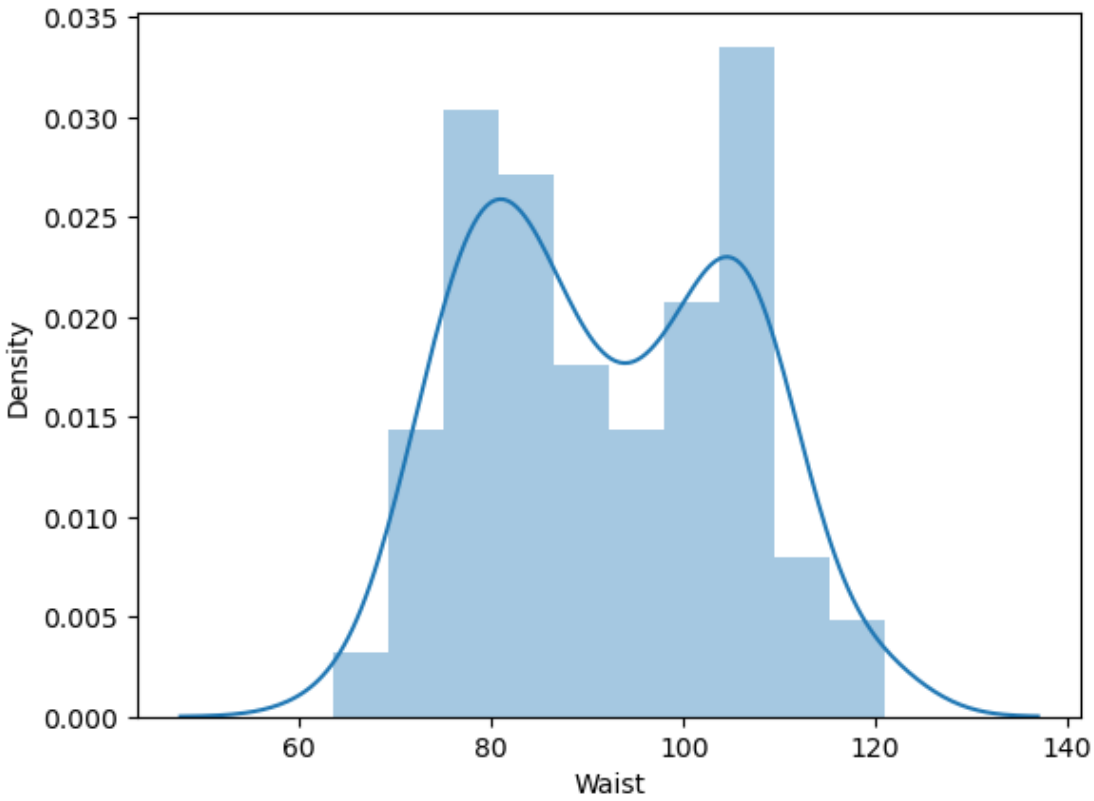
[201]:
```
#Q 21 b)
sn.distplot(q21b['AT'],kde=True, bins =10)
plt.show()
```

```
C:\Users\LENOVO\AppData\Local\Temp\ipykernel_7028\212691754.py:2: 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

  sn.distplot(q21b['AT'],kde=True, bins =10)
```



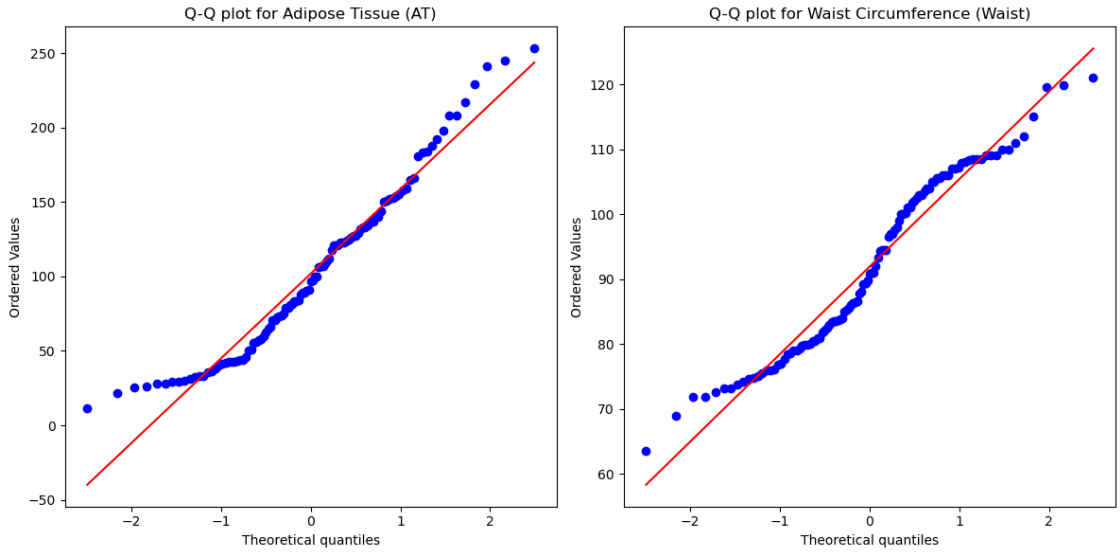
[203]:
```
sn.distplot(q21b['Waist'],kde=True, bins =10)
plt.show()
```

```
C:\Users\LENOVO\AppData\Local\Temp\ipykernel_7028\430685058.py:1: 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

  sn.distplot(q21b['Waist'],kde=True, bins =10)
```



[175]:
```
# Plot Q-Q plots
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
stats.probplot(q21b['AT'], dist="norm", plot=plt)
plt.title('Q-Q plot for Adipose Tissue (AT)')
```

```
plt.subplot(1, 2, 2)
stats.probplot(q21b['Waist'], dist="norm", plot=plt)
plt.title('Q-Q plot for Waist Circumference (Waist)')

plt.savefig('wc & at.jpg')
plt.tight_layout()
plt.show()
```



0.10 Question 22 Soln

[176]:
```
from scipy.stats import norm

# Z-score for 90% confidence interval
z_90 = norm.ppf(0.95)  # 90% confidence interval (0.95 on each side)

# Z-score for 94% confidence interval
z_94 = norm.ppf(0.97)  # 94% confidence interval (0.97 on each side)

# Z-score for 60% confidence interval
z_60 = norm.ppf(0.80)  # 60% confidence interval (0.80 on each side)

print("Z-score for 90% confidence interval:", round(z_90,3))
print("Z-score for 94% confidence interval:", round(z_94, 3))
print("Z-score for 60% confidence interval:", round(z_60,3))
```

```
Z-score for 90% confidence interval: 1.645
Z-score for 94% confidence interval: 1.881
```

```
Z-score for 60% confidence interval: 0.842
```

0.11 Question 23 Soln

[177]:
```python
# Sample size
n = 25

# Degrees of freedom
dof = n - 1

# t-score for 95% confidence interval
t_95 = t.ppf(0.975, dof)  # 95% confidence interval (0.025 on each side)

# t-score for 96% confidence interval
t_96 = t.ppf(0.98, dof)  # 96% confidence interval (0.02 on each side)

# t-score for 99% confidence interval
t_99 = t.ppf(0.995, dof)  # 99% confidence interval (0.005 on each side)

print("t-score for 95% confidence interval:", round(t_95,3))
print("t-score for 96% confidence interval:", round(t_96, 3))
print("t-score for 99% confidence interval:", round(t_99,3))
```

```
t-score for 95% confidence interval: 2.064
t-score for 96% confidence interval: 2.172
t-score for 99% confidence interval: 2.797
```

0.12 Question 24 Soln

[178]:
```python
# Given data
sample_mean = 260
population_mean = 270
sample_std = 90
sample_size = 18
confidence_level = 0.96

# Calculate the t-score
df = sample_size - 1
t_score = t.ppf(confidence_level + (1 - confidence_level) / 2, df)

# Calculate the standard error of the mean
standard_error = sample_std / (sample_size ** 0.5)

# Calculate the t-statistic
t_statistic = (sample_mean - population_mean) / standard_error

# Calculate the probability
```

```
probability = t.cdf(t_statistic, df)

print("t-score:", round(t_score,3))
print("t-statistic:", round(t_statistic,3))
print("Probability:", round(probability,3))
```

```
t-score: 2.224
t-statistic: -0.471
Probability: 0.322
```