201600779 김영민

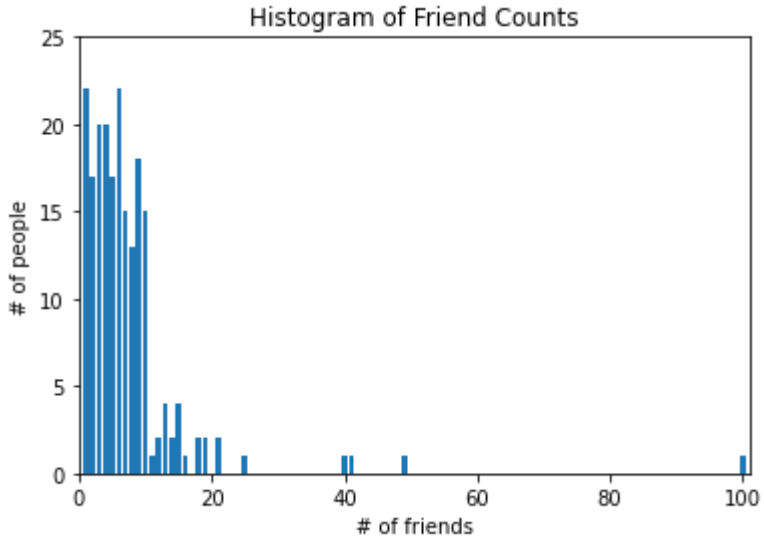
In [1]:

```
import warnings
warnings.filterwarnings('ignore')
```

In [2]:

```
def sum_of_squares(v):  # linear_algebra.py 코드에서 임포트 하는 대신 여기에 코딩

    return dot(v, v)

def dot(v, w):       # linear_algebra.py 코드에서 임포트 하는 대신 여기에 코딩

    return sum(v_i * w_i for v_i, w_i in zip(v, w))

from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

from collections import Counter
#from linear_algebra import sum_of_squares, dot # linear_algebra.py 코드에서 임포트
import math
import numpy as np

num_friends = [100,49,41,40,25,21,21,19,19,18,18,16,15,15,15,15,14,14,13,13,13,13,12,12,11,10,10,10,

def make_friend_counts_histogram(plt):
    friend_counts = Counter(num_friends)
    xs = range(101)
    ys = [friend_counts[x] for x in xs]
    plt.bar(xs, ys)
    plt.axis([0,101,0,25])
    plt.title("Histogram of Friend Counts")
    plt.xlabel("# of friends")
    plt.ylabel("# of people")
    plt.show()

import matplotlib as plt
%pylab inline

make_friend_counts_histogram(plt)
```

```
Populating the interactive namespace from numpy and matplotlib
```



In [3]:

```
num_points = len(num_friends)                    # 204
largest_value = max(num_friends)                 # 100
smallest_value = min(num_friends)                # 1
sorted_values = sorted(num_friends)
smallest_value = sorted_values[0]                  # 1
second_smallest_value = sorted_values[1]    # 1
second_largest_value = sorted_values[-2]    # 49

print(num_points)
print(largest_value)
print(smallest_value)
print(sorted_values)
print(smallest_value)
print(second_smallest_value)
print(second_largest_value)
```

```
204
100
1
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5,
5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
6, 6, 6, 6, 6, 6, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 8, 8, 8, 8, 8, 8, 8,
8, 8, 8, 8, 8, 8, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 10, 10, 10,
10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 11, 12, 12, 13, 13, 13, 13, 14, 14,
15, 15, 15, 15, 16, 18, 18, 19, 19, 21, 21, 25, 40, 41, 49, 100]
1
1
49
```

In [4]:

```
def mean(x):
    return sum(x) / len(x)

mean(num_friends)

# Numpy version
np.mean(num_friends)
```

Out[4]:

```
7.333333333333333
```

Out[4]:

```
7.333333333333333
```

In [5]:

```
# 데이터의 중앙에 있는 값(홀수) 또는 중앙에 있는 두 값의 평균(짝수)
def median(v):
    """finds the 'middle-most' value of v"""
    n = len(v)
    sorted_v = sorted(v)
    midpoint = n // 2

    if n % 2 == 1:
        # if odd, return the middle value
        return sorted_v[midpoint]
    else:
        # if even, return the average of the middle values
        lo = midpoint - 1
        hi = midpoint
        return (sorted_v[lo] + sorted_v[hi]) / 2

median(num_friends)

# Numpy version
np.median(num_friends)
```

Out[5]:

```
6.0
```

Out[5]:

```
6.0
```

In [6]:

```
def quantile(x, p):
    """returns the pth-percentile value in x"""
    p_index = int(p * len(x))
    return sorted(x)[p_index]

for i in range(0, 100, 25):
    print("%.2f Percentage value" % (i*0.01) , quantile(num_friends, i * 0.01))

# Numpy version
np.percentile(num_friends, [i for i in range(0,100,25)])
```

```
0.00 Percentage value 1
0.25 Percentage value 3
0.50 Percentage value 6
0.75 Percentage value 9
```

Out[6]:

```
array([1., 3., 6., 9.])
```

In [7]:

```
def mode(x):
    """returns a list, might be more than one mode"""
    counts = Counter(x)
    max_count = max(counts.values())
    return [x_i for x_i, count in counts.items()
            if count == max_count]

mode(num_friends)
```

Out[7]:

```
[6, 1]
```

In [8]:

```
# "range" already means something in Python, so we'll use a different name
def data_range(x):
    return max(x) - min(x)

data_range(num_friends)

np.max(num_friends) - np.min(num_friends)
```

Out[8]:

```
99
```

Out[8]:

```
99
```

In [9]:

```
# Mean - value

def de_mean(x):
    """translate x by subtracting its mean (so the result has mean 0)"""
    x_bar = mean(x)
    return [x_i - x_bar for x_i in x]

def variance(x):
    """assumes x has at least two elements"""
    n = len(x)
    deviations = de_mean(x)
    return sum_of_squares(deviations) / (n - 1)

variance(num_friends)

%timeit variance(num_friends)
%timeit np.var(num_friends) # 일반적인 분산 연산도 numpy가 빠름
```

Out[9]:

81.54351395730706

```
58.3 µs ± 2.01 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
The slowest run took 4.50 times longer than the fastest. This could mean that an int
ermediate result is being cached.
65.4 µs ± 38 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

In [10]:

```
def standard_deviation(x):
    return math.sqrt(variance(x))

standard_deviation(num_friends)

np.std(num_friends, dtype=np.float64)

def interquartile_range(x):
    return quantile(x, 0.75) - quantile(x, 0.25)

interquartile_range(num_friends)
```

Out[10]:

9.030144736232474

Out[10]:

9.007984838446012

Out[10]:

6

In [11]:

```
daily_minutes = [1,68.77,51.25,52.08,38.36,44.54,57.13,51.4,41.42,31.22,34.76,54.01,38.79,47.59,49.1

def covariance(x, y):
    n = len(x)
    return dot(de_mean(x), de_mean(y)) / (n - 1)

covariance(num_friends, daily_minutes)

np.cov(num_friends,daily_minutes)
```

Out[11]:

```
22.42543513957307
```

Out[11]:

```
array([[ 81.54351396,  22.42543514],
       [ 22.42543514, 100.78589895]])
```

In [12]:

```
def correlation(x, y):
    stdev_x = standard_deviation(x)
    stdev_y = standard_deviation(y)
    if stdev_x > 0 and stdev_y > 0:
        return covariance(x, y) / stdev_x / stdev_y
    else:
        return 0 # if no variation, correlation is zero

correlation(num_friends, daily_minutes)

np.corrcoef(num_friends, daily_minutes)

plt.plot(num_friends, daily_minutes, 'ro')
plt.axis([0,max(num_friends)+10,0,max(daily_minutes) +10 ])
plt.show()
```

Out[12]:

```
0.2473695736647823
```

Out[12]:

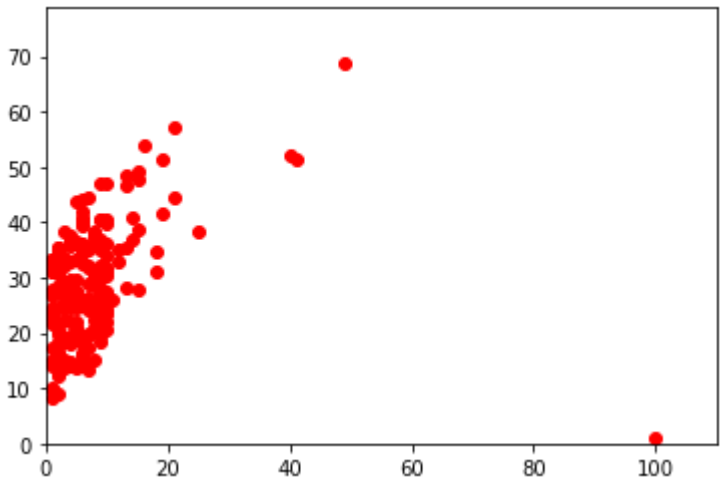
```
array([[1.        , 0.24736957],
       [0.24736957, 1.        ]])
```

Out[12]:

```
[<matplotlib.lines.Line2D at 0x1fde0ffb790>]
```

Out[12]:

```
(0.0, 110.0, 0.0, 78.77)
```



In [13]:

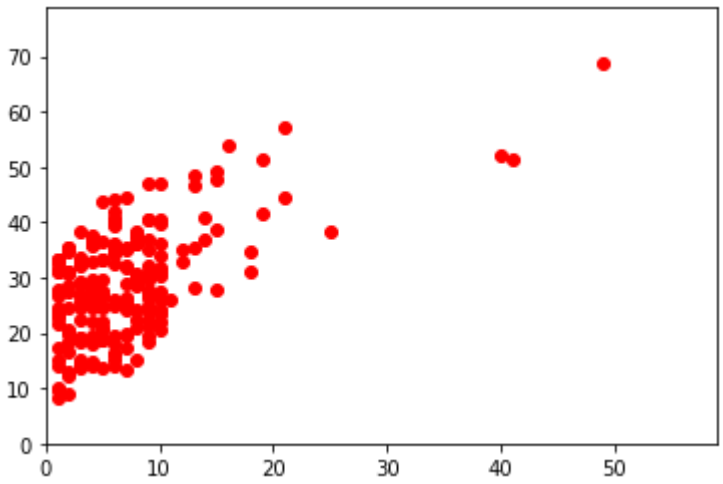
```
outlier = num_friends.index(100) # index of outlier

num_friends_good = [x
                    for i, x in enumerate(num_friends)
                    if i != outlier]

daily_minutes_good = [x
                      for i, x in enumerate(daily_minutes)
                      if i != outlier]

plt.plot(num_friends_good, daily_minutes_good, 'ro')
plt.axis([0,max(num_friends_good)+10,0,max(daily_minutes_good) +10 ])
plt.show()
```

Out[13]:

```
[<matplotlib.lines.Line2D at 0x1fde14c7b80>]
```

Out[13]:

```
(0.0, 59.0, 0.0, 78.77)
```



Apply Data

In [14]:

```
import pandas as pd
data = pd.read_csv('height-weight.csv')
data.head()
```

Out[14]:

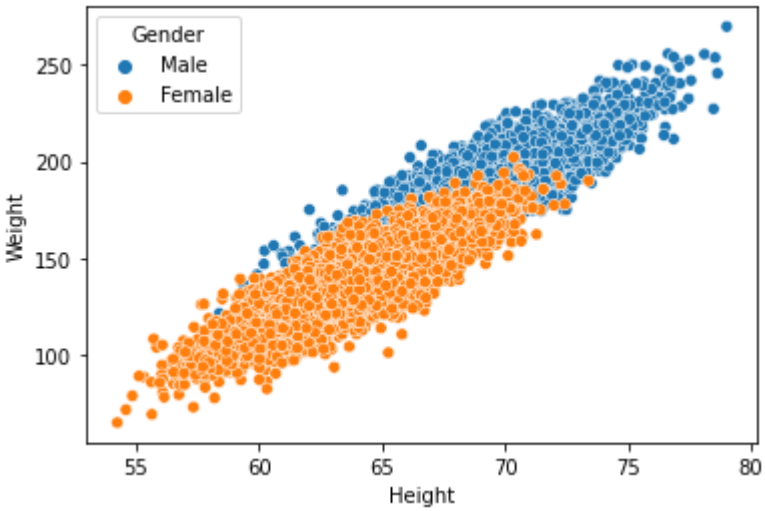
	Gender	Height	Weight
0	Male	73.847017	241.893563
1	Male	68.781904	162.310473
2	Male	74.110105	212.740856
3	Male	71.730978	220.042470
4	Male	69.881796	206.349801

In [15]:

```
import seaborn as sns
sns.scatterplot(data=data,x = 'Height',y='Weight',hue='Gender')
```

Out[15]:

<AxesSubplot:xlabel='Height', ylabel='Weight'>



In [16]:

```
man = data[data['Gender'] == 'Male']
woman = data[data['Gender'] == 'Female']
```

In [17]:

```
man.describe()
```

Out[17]:

	Height	Weight
count	5000.000000	5000.000000
mean	69.026346	187.020621
std	2.863362	19.781155
min	58.406905	112.902939
25%	67.174679	173.887767
50%	69.027709	187.033546
75%	70.988744	200.357802
max	78.998742	269.989699

In [18]:

```
woman.describe()
```

Out[18]:

	Height	Weight
count	5000.000000	5000.000000
mean	63.708774	135.860093
std	2.696284	19.022468
min	54.263133	64.700127
25%	61.894441	122.934096
50%	63.730924	136.117583
75%	65.563565	148.810926
max	73.389586	202.237214

In [19]:

```
print('Mean')
print()
print('Man Average Height= {0:0.2f} Woman Average Height= {0:0.2f}'.format(np.mean(man['Height']),np
print('Man Average Weight= {0:0.2f} , Woman Average Weight= {0:0.2f}'.format(np.mean(man['Weight']),
```

```
Mean

Man Average Height= 69.03 Woman Average Height= 69.03
Man Average Weight= 187.02 , Woman Average Weight= 187.02
```

In [20]:

```
print('Median')
print()
print('Man Median Height= {0:0.2f}, Woman Median Height= {0:0.2f}'.format(np.median(man['Height']),n
print('Man Median Weight= {0:0.2f}, Woman Median Weight= {0:0.2f}'.format(np.median(man['Weight']),n
```

```
Median

Man Median Height= 69.03, Woman Median Height= 69.03
Man Median Weight= 187.03, Woman Median Weight= 187.03
```

In [21]:

```
print('Quantile')
print()
print('Man Quantile Height= {0:0.2f}, Woman Median Height= {0:0.2f}'.format(np.quantile(man['Height'
print('Man Quantile Weight= {0:0.2f}, Woman Median Weight= {0:0.2f}'.format(np.quantile(man['Weight'
```

```
Quantile

Man Quantile Height= 67.17, Woman Median Height= 67.17
Man Quantile Weight= 173.89, Woman Median Weight= 173.89
```

In [22]:

```
print('Mode')
print()
print('Man Mode Height= {}, Woman Mode Height = {}'.format(man['Height'].astype(int).mode()[0],woman
print('Man Mode Weight= {}, Woman Mode Weight = {}'.format(man['Weight'].astype(int).mode()[0],woman
```

```
Mode

Man Mode Height= 69, Woman Mode Height = 63
Man Mode Weight= 192, Woman Mode Weight = 137
```

In [24]:

```
print('Range')
print()
man_h_range = np.max(man['Height']) - np.min(man['Height'])
man_w_range = np.max(man['Weight']) - np.min(man['Weight'])
woman_h_range = np.max(woman['Height']) - np.min(woman['Height'])
woman_w_range = np.max(woman['Weight']) - np.min(woman['Weight'])
print('Man Range Height = {0:0.2f}, Woman Range Height = {0:0.2f}'.format(man_h_range,woman_h_range)
print('Man Range Weight = {0:0.2f}, Woman Range Weight = {0:0.2f}'.format(man_w_range,woman_w_range)
```

```
Range

Man Range Height = 20.59, Woman Range Height = 20.59
Man Range Weight = 157.09, Woman Range Weight = 157.09
```

In [25]:

```
print('Variance')
print()
print('Man Variance Height = {0:0.2f}, Woman Variance Height = {0:0.2f}'.format(man['Height'].var(),
print('Man Variance Weight = {0:0.2f}, Woman Variance Weight = {0:0.2f}'.format(man['Weight'].var(),
```

```
Variance

Man Variance Height = 8.20, Woman Variance Height = 8.20
Man Variance Weight = 391.29, Woman Variance Weight = 391.29
```

In [26]:

```
print('Std')
print()
print('Man Std Height = {0:0.2f}, Woman Std Height = {0:0.2f}'.format(man['Height'].std(),woman['Hei
print('Man Std Weight = {0:0.2f}, Woman Std Weight = {0:0.2f}'.format(man['Weight'].std(),woman['Wei
```

```
Std

Man Std Height = 2.86, Woman Std Height = 2.86
Man Std Weight = 19.78, Woman Std Weight = 19.78
```

In [27]:

```
print('Covariance')
print()
man_cov = np.cov(man['Height'],man['Weight'])
woman_cov = np.cov(woman['Height'],woman['Weight'])

print('Man Covariance Height-Weight = ₩n{}'.format(man_cov))
print('Woman Covariance Height-Weight = ₩n{}'.format(woman_cov))
```

```
Covariance

Man Covariance Height-Weight =
[[  8.19884325  48.87964899]
 [ 48.87964899 391.29407402]]
Woman Covariance Height-Weight =
[[  7.26994749  43.57640416]
 [ 43.57640416 361.8542814 ]]
```

In [28]:

```
print('Correlation')
print()
man_corr = np.corrcoef(man['Height'],man['Weight'])
woman_corr = np.corrcoef(woman['Height'],woman['Weight'])

print('Man Correlation Height-Weight = ₩n{}'.format(man_corr))
print('Woman Correlation Height-Weight = ₩n{}'.format(woman_corr))
```

```
Correlation

Man Correlation Height-Weight =
[[1.         0.86297885]
 [0.86297885 1.        ]]
Woman Correlation Height-Weight =
[[1.         0.84960859]
 [0.84960859 1.        ]]
```

201600779 김영민