Week 1 - Normalization and data types

This first week is going to be a warm-up week where we conider different statistical tools data types.

Lets first import the libraries that we are going to need.

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import scipy.stats
import scipy as sp
```

Exercise 1: Sample statistics and normalization

Let X and Y be two random variables, denoting age and weight, respectively.
Consider a random sample of size $n = 20$ from these two variables

```
X = np.array([69, 74, 68, 70, 72, 67, 66, 70, 76, 68, 72, 79, 74, 67, 66, 71,
74, 75, 75, 76])
Y = np.array([153, 175, 155, 135, 172, 150, 115, 137, 200, 130, 140, 265, 185,
112, 140, 150, 165, 185, 210, 220])
```

a) Find the mean, median and mode for X

```
def compute_mean(X):
    """
    Input: certain list of data
    Output: mean of this list
    Note: There also have an optional to build function prescind from Numpy
    """
    # TODO: the calculation process of mean
    mean = sum(X)/len(X)
    return mean

def compute_median(X):
    # TODO: the calculation process of median
    if len(X) % 2 == 1:
        median=sorted(X)[int(len(X)//2)]
    else:
        median=(sorted(X)[len(X)//2]+sorted(X)[len(X)//2-1])/2
    return median

def compute_mode(X):
    """
    Note: Could also build a counts dictionary for observing elements which may
reuse when calculating pmf
    """
    # TODO: the calculation process of mode
    count=count_dictionary(X)
    mode=max(count,key=count.get)
    return mode
```

```
def count_dictionary(X):
    # We need y counts also for later when we are computing the empirical PMF
    count={}
    for i in X:
        if i not in count.keys():
            count[i]=1
        else:
            count[i]+=1
    return count

x_mean = compute_mean(X)
x_median = compute_median(X)
x_mode = compute_mode(X)

print(f"Mean: {x_mean}\nMedian: {x_median}\nMode: {x_mode}" )
```

```
Mean: 71.45
Median: 71.5
Mode: 74
```

b) What is the variance for Y?

```
# In order to make our life a lot easier, we will compute the probability of
each element
# Because that will be useful for the next couple of tasks
# NOTE: We can reuse the dictionary from earlier (Task A)

def compute_pmf(observation_counts):
    unique_values = []
    probabilities = []
    for k, v in observation_counts.items():
        unique_values.append(k)
        probabilities.append(v/len(X))
    print(f"Values: {unique_values}\nProbabilities: {probabilities}")
    return {k: v for k, v in zip(unique_values, probabilities)}

def compute_ev(pmf):
    expected_value = 0
    for v, p in pmf.items():
        expected_value += v * p
    print(f"Expected value: {expected_value}")
    return expected_value

def compute_variance(pmf, ev):
    variance = 0
    for v, p in pmf.items():
        variance += p * ((v - ev)**2)
    print(f"Variance is {variance}")
    return variance

x_pmf = compute_pmf(x_counts)
x_ev = compute_ev(x_pmf)
x_variance = compute_variance(x_pmf, x_ev)
# Need these for later
print("\nAnd for y it is:")
```

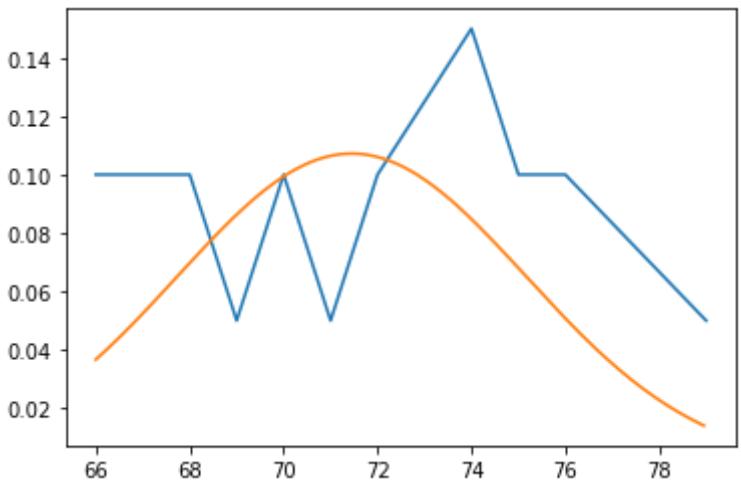
```
y_pmf = compute_pmf(y_counts)
y_ev = compute_ev(y_pmf)
y_variance = compute_variance(y_pmf, y_ev)
```

```
Values: [69, 74, 68, 70, 72, 67, 66, 76, 79, 71, 75]
Probabilities: [0.05, 0.15, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.05, 0.05, 0.1]
Expected value: 71.45000000000002
Variance is 13.847500000000002

And for y it is:
Values: [153, 175, 155, 135, 172, 150, 115, 137, 200, 130, 140, 265, 185, 112,
165, 210, 220]
Probabilities: [0.05, 0.05, 0.05, 0.05, 0.05, 0.1, 0.05, 0.05, 0.05, 0.05, 0.1,
0.05, 0.1, 0.05, 0.05, 0.05, 0.05]
Expected value: 164.7
Variance is 1369.21
```

c) Plot the normal distribution for X. Consider if the data seems to fit a normal distribution.

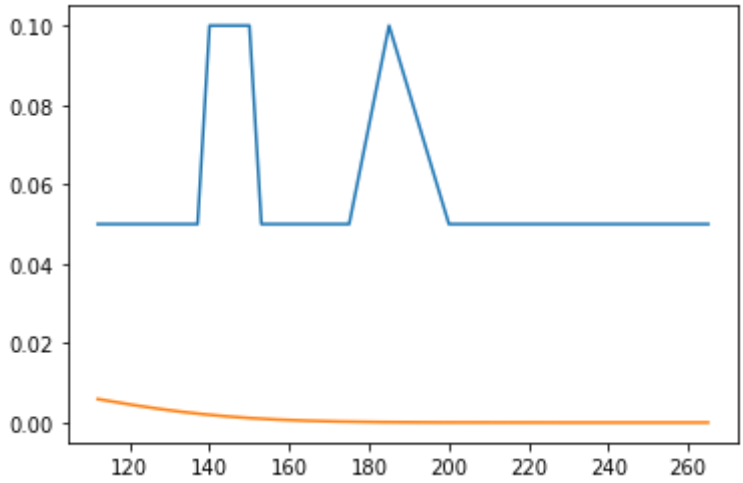
```
import math
# Okay, I could not resist Python magic on this one
# I wanted to sort the observations and probabilities based on the observations,
so we zip the two lists
# Sort based on the observations and zip using zip(* LIST)
unique_values, probabilities = zip(*(sorted(x_pmf.items(), key=lambda x: x[0])))
plt.plot(unique_values, probabilities)
x_normal = np.arange(min(X), max(X), 1/len(X))

#  scipy.stats.norm.pdf could help get probility density function of normal
distribution
plt.plot(x_normal, scipy.stats.norm.pdf(x_normal, x_mean,
math.sqrt(x_variance)))
```

```
[<matplotlib.lines.Line2D at 0x7fc0b8093ca0>]
```



```
import math
# Okay, I could not resist Python magic on this one
# I wanted to sort the observations and probabilities based on the observations,
so we zip the two lists
# Sort based on the observations and zip using zip(* LIST)
unique_values, probabilities = zip(*(sorted(y_pmf.items(), key=lambda x: x[0])))
plt.plot(unique_values, probabilities)
y_normal = np.arange(min(Y), max(Y), 1/len(Y))

plt.plot(y_normal, scipy.stats.norm.pdf(y_normal, y_mean,
math.sqrt(y_variance)))
```

```
[<matplotlib.lines.Line2D at 0x7fc10a5b1af0>]
```



d) What is the probability of observing an age of 80 or higher?

Answer

Using the probability mass function, the probability is 0 since there is no samples of the value 80. However, using the probability density function of normal distribution for X, we can evaluate the density in range $[80, \infty)$.

e) Find the 2-dimensional mean $\hat{\mu}$ and the covariance matrix $\widehat{\Sigma}$ for these two variables.

```
# 2-Dimensional mean is the vector (E[X_1], E[X_2])
Z = np.c_[X, Y]
n, d = Z.shape
mu = np.mean(Z, axis=0, keepdims=True)
cov = (Z - mu).T @ (Z - mu)

print("Mean:\n", mu)
print("Cov: \n", cov / (n-1))
```

```
Mean:
 [[ 71.45 164.7 ]]
Cov:
 [[  14.57631579  128.87894737]
 [ 128.87894737 1441.27368421]]
```

f) Normalize the data with *range normalization* to the range $[0, 1]$

range normalization:

$x_i^{'} = \frac{x_i - \min_i x_i}{\max_i x_i - \min_i x_i}$

```
# TODO
# Range normalization is (x_i - min)/(max - min)

# Ordinarily I would set min and max as variables to save computer time. Who
needs efficiency amirite?
min_z = Z.min(0, keepdims=True)
max_z = Z.max(0, keepdims=True)
Z_normalize = (Z - min_z) / (max_z - min_z)
print("Range normalized: \n", Z_normalize)
```

```
Range normalized:
 [[0.23076923 0.26797386]
 [0.61538462 0.41176471]
 [0.15384615 0.28104575]
 [0.30769231 0.1503268 ]
 [0.46153846 0.39215686]
 [0.07692308 0.24836601]
 [0.         0.01960784]
 [0.30769231 0.16339869]
 [0.76923077 0.5751634 ]
 [0.15384615 0.11764706]
 [0.46153846 0.18300654]
 [1.         1.        ]
 [0.61538462 0.47712418]
 [0.07692308 0.        ]
 [0.         0.18300654]
 [0.38461538 0.24836601]
 [0.61538462 0.34640523]
 [0.69230769 0.47712418]
 [0.69230769 0.64052288]
 [0.76923077 0.70588235]]
```

g) Normalize the data with *standard score normalization*, such that is has mean 0 and standard deviation 1.

z-score:

$x_i^{'} = \frac{x_i - \hat{\mu}}{\hat{\sigma}}$

```
# TODO
# With standard score normalization, x_i = (x_i - mu)/(sigma)
X_ssn = [(x - x_mean)/np.sqrt(x_variance) for x in X]
# Cheating and computing mean of Y with Numpy
Y_ssn = [(y - np.mean(Y))/np.sqrt(y_variance) for y in Y]
print(np.mean(Y_ssn), np.std(Y_ssn))
print(np.mean(X_ssn), np.std(X_ssn))
```

```
3.4416913763379854e-16 1.0
-7.216449660063518e-16 1.0
```

Exercise 2: Robustness

Determine whether the following statements are true or false and explain why.

- Mean is robust against outliers
- Median is robust against outliers
- Standard deviation is robust against outliers

Answer

- Mean is robust against outliers

  False: We say that a statistic is robust if it is not affected by extreme values (such as outliers) in the data. The sample mean is unfortunately not robust because a single large value (an outlier) can skew the average.

Data Mining and Analysis Page 47

- Median is robust against outliers

  True: It is not affected by extreme values

- Standard deviation is robust against outliers

  False: Standard deviation $\sigma = \sqrt{\frac{\sum(x_i-\mu)^2}{N}}$, it could influenced by each x_i

Exercise 3: More on robustness

Provide an informal definition of when a measure is robust.

answer
A measure is robust if it is insensitive to outliers. A robust measure would give you roughly the same result despite the presence of errors.

Exercise 4: Independence analysis

Please explain what the idea of independence analysis is using contingency tables; for which kind of data is this particularly relevant?

Answer

Data Mining and Analysis Page 78

Exercise 5: Contingency table and χ^2 statistics

In the table below, assuming that X_1 is discretized into three bins, as follows: $[-2, -0.5]$, $[-0.5, 0.5]$, and $[0.5, 2]$.

X_1	X_2
0.3	a
−0.3	b
0.44	a
−0.60	a
0.40	a
1.20	b
−0.12	a
−1.60	b
1.60	b
−1.32	a

Answer the following questions

a) Construct the contingency table between the discretized X_1 and X_2 attributes . Include the marginal counts.
b) Compute the χ^2 statistic between them.
c) Determine whether they are dependent or not at the 5% significance level. Use the χ^2 critical values from Table 3.10.

Table 3.10. χ^2 Critical values for different *p-values* for different degrees of freedom (q): For example, for $q = 5$ degrees of freedom, the critical value of $\chi^2 = 11.070$ has *p-value* $= 0.05$.

q	0.995	0.99	0.975	0.95	0.90	0.10	0.05	0.025	0.01	0.005
1	—	—	0.001	0.004	0.016	2.706	3.841	5.024	6.635	7.879
2	0.010	0.020	0.051	0.103	0.211	4.605	5.991	7.378	9.210	10.597
3	0.072	0.115	0.216	0.352	0.584	6.251	7.815	9.348	11.345	12.838
4	0.207	0.297	0.484	0.711	1.064	7.779	9.488	11.143	13.277	14.860
5	0.412	0.554	0.831	1.145	1.610	9.236	11.070	12.833	15.086	16.750
6	0.676	0.872	1.237	1.635	2.204	10.645	12.592	14.449	16.812	18.548

Answer:

a)
Table: observed contingency table

	[-2,-0.5]	**[-0.5,0.5]**	**[0.5, 2]**	**row counts**
a	2	4		6
b	1	1	2	4
column counts	3	5	2	10

column counts	3	5	2	10

```
X_1 = [0.3, -0.3, 0.44, -0.60, 0.40, 1.20, -0.12, -1.60, 1.60, -1.32]
X_2 = ["a", "b", "a", "a", "a", "b", "a", "b", "b", "a"]
# For sanity's sake, use code to bin X_1
# Converting everything to numbers allow us to index in them for easier
computation
X_tmp = []
for x in X_1:
    if x >= -2 and x <= -0.5:
        X_tmp.append(0)
    elif x >= -0.5 and x <= 0.5:
            X_tmp.append(1)
    elif x > 0.5:
            X_tmp.append(2)
X_1 = X_tmp
X_2 = [0 if x == "a" else 1 for x in X_2]


# Our table looks like
## Bins: a b
## bin1
## bin2
## bin3
b1_row = [0, 0]
b2_row = [0, 0]
b3_row = [0, 0]
count_row = [0, 0, 0] # THIS ONE is for column counts
c_table = [b1_row, b2_row, b3_row]
for i in range(len(X_1)):
    x1_index = X_1[i]
    x2_index = X_2[i]
    c_table[x1_index][x2_index] += 1
print(f"Our table before row counts: {c_table}")
```

```
Our table before row counts: [[2, 1], [4, 1], [0, 2]]
```

b)

Table: Expectation table

	[-2,-0.5]	[-0.5,0.5]	[0.5, 2]	row counts
a	1.8	3	1.2	6
b	1.2	2	0.8	4
column counts	3	5	2	10

```
# Now add row counts
# Remember these are reference variables, so modifying these
# will also modify our table
b1_row.append(sum(b1_row))
b2_row.append(sum(b2_row))
b3_row.append(sum(b3_row))
c_table.append([b1_row[i] + b2_row[i] + b3_row[i] for i in range(3)])

"""
print(f"Our contingency table:")
print(".......a, b, counts")
print(f"bin1: {b1_row}")
print(f"bin2: {b2_row}")
print(f"bin3: {b3_row}")
print(f"count: {c_table[3]}")
"""

# Construct matrix of e_{i,j} needed for Chi^2
E = [[0, 0] for _ in range(3)]
for i in range(3):
    for j in range(2):
        E[i][j] = (c_table[i][2] * c_table[3][j])/(len(X_1))
```

Table: X^2

	[-2,-0.5]	[-0.5,0.5]	[0.5, 2]	row counts
a	0.022	0.33	1.2	6
b	0.033	0.5	1.8	4
column counts	3	5	2	10

```
Chi_squared = 0
for i in range(3):
    for j in range(2):
        Chi_squared += ((c_table[i][j]-E[i][j])**2)/(E[i][j])
print(f"Chi_squared is: {Chi_squared}")
```

```
Chi_squared is: 3.8888888888888884
```

null hypothesis H_0, X_1 and X_2 are independent

Chi-square

$X^2 = \sum_{i_1}^{m_1} \sum_{j_1}^{m_2} \frac{(n_ij - eij)^2}{e_ij}$

$X^2 = 0.022 + 0.333 + 0 + 0.333 + 0.5 + 1.8 = 3.886$

degree of freedom

$q = (2-1) * (3-1) = 2$

cirtical value by table $= 5.991$

$X^2 \leq 5.991$ it accept the null hypothesis

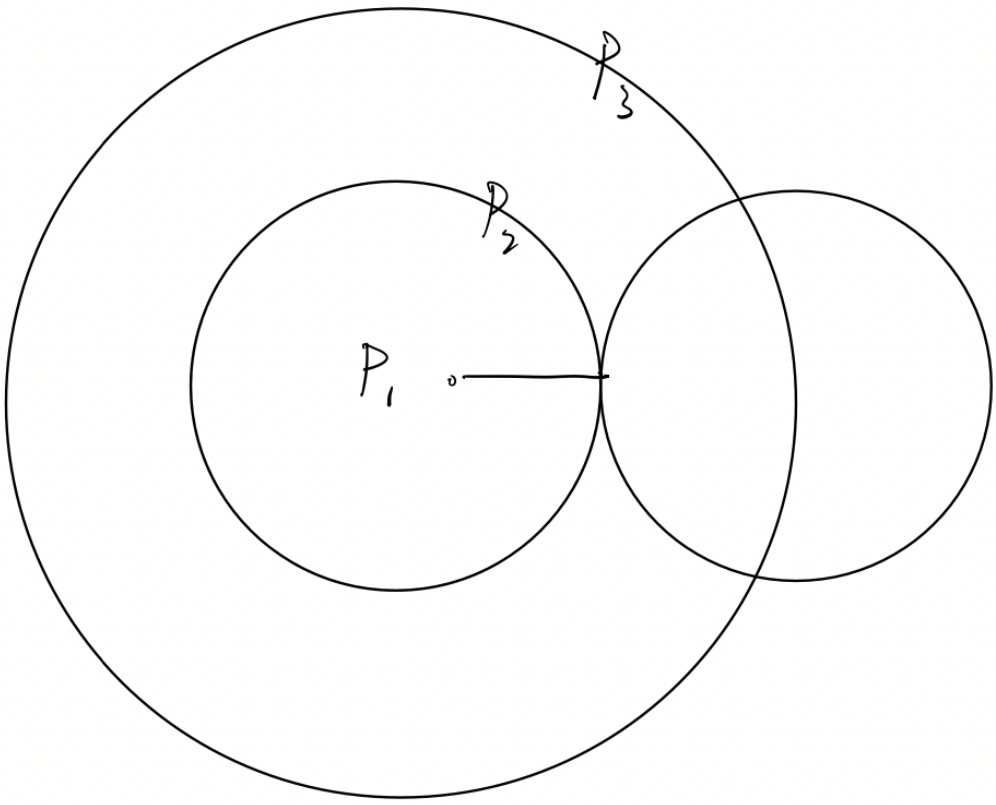
Exercise 6: Working with Metrics

Consider the following situation:

We know some distances between data points: $d(p_1, p_2) = 1$, $d(p_1, p_3) = 2$, $d(p_3, p_4) = 1$.
We also know that d is a metric.

1. What do we know about the remaining distances?
2. Do we need to compute further distances if we want to find the two points that are most similar to p_1?
3. Can p_4 be closer to p_2 than p_3 is to p_2?

Answers:



Optional Exercises

Exercise 7: Mean absolute deviation

Define a measure of deviation called *mean absolute deviation* for a random variable X as follows:

$$\frac{1}{n}\sum_{i=1}^{n}|x_i-\mu|$$

Is this measure robust? Why or why not?

Answer

It is not robust, as a single outlier can skew the mean absolute deviation. (Similar to mean and variance)

Exercise 8: Correlation

Consider the table below. Assume that both the attributes X and Y are numeric, and the table represents the entire population. And we know that the correlation between X and Y is zero.

1. What can you infer about the values of Y? (Find a relationship between a, b and c)
2. If we know there is a missing row, what can we infer about it?

X	Y
1	a
0	b
1	c
0	a
0	c

Answer

1. since it is independent, we have

$E[XY]=E[X]E[Y]$

and

$E[XY]=\frac{a+c}{5}$

$E[X]=\frac{2}{5}$

$E[Y]=\frac{2a+b+2c}{5}$

We have $\frac{a+c}{5}=\frac{2}{5}\cdot\frac{2a+b+2c}{5}$

solve this we got, $2b=a+c$

2.

	1	**0**
a	1	1
b	-	1
c	1	1

from the contigency table we could figure $[1,b]$ are now empty and could add one more line.

Exercise 9: 3-way contingency table

Consider the 3-way contingency table for attributes X,Y,Z shown in the table below. Compute the χ^2 metric for the correlation between Y and Z. Are they dependent or independent at the 95% confidence level? See Table 3.10 above for χ^2 values.

	$Z = f$		$Z = g$	
	$Y = d$	$Y = e$	$Y = d$	$Y = e$
$X = a$	5	10	10	5
$X = b$	15	5	5	20
$X = c$	20	10	25	10

```
import scipy.stats.contingency
from scipy.stats import chi2_contingency
xy = np.array([[40,25],[40,35]])
chi2_contingency(xy)
```

```
(0.6515384615384604,
 0.4195631759021463,
 1,
 array([[37.14285714, 27.85714286],
        [42.85714286, 32.14285714]]))
```

Exercise 10: Mixed data

Consider the "mixed" data given in the table below. Here X_1 is a numeric attribute and X_2 is a categorical one. Assume that the domain of X_2 is given as $dom(X_2) = \{a, b\}$. Answer the following questions.

a) What is the mean vector for this dataset?

b) What is the covariance matrix?

X_1	X_2
0.3	a
−0.3	b
0.44	a
−0.60	a
0.40	a
1.20	b
−0.12	a
−1.60	b
1.60	b
−1.32	a

```
fine = np.array([[0.3,1,0],[-0.3,0,1],[0.44,1,0],[-0.6,1,0],[0.4,1,0],[1.2,0,1],
[-0.12,1,0],[-0.16,0,1],[1.6,0,1],[-1.32,1,0]])
mu = np.mean(fine, axis=0)
print("mean vector:\n", mu)
```

```
mean vector:
 [0.144 0.6   0.4  ]
```

```
n, d = fine.shape
mu = np.mean(fine, axis=0, keepdims=True)
cov = (fine - mu).T @ (fine - mu)

print("Cov: \n", cov / (n-1))
```

```
Cov:
 [[ 0.71873778 -0.196       0.196     ]
 [-0.196       0.26666667 -0.26666667]
 [ 0.196      -0.26666667  0.26666667]]
```