Text Mining Tutorial 4:

Textual Data Representation 2

Prof. Hsing-Kuo Pao
Teaching Assistant: Ghaluh Indah Permata Sari

TAIWAN TECH
NATIONAL TAIWAN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Outline

- Embedding Model
  - Word2Vec (word embedding)
- Textual Correlation
  - Similarity
  - Distance
  - Correlation

Index term: Vector Space Model

- Word embedding is a technique where individual words are transformed into a numerical representation of the word (a vector).

```
For example: have = [1, 0, 0, 0, 0, 0, ... 0]
a    = [0, 1, 0, 0, 0, 0, ... 0]
good = [0, 0, 1, 0, 0, 0, ... 0]
day  = [0, 0, 0, 1, 0, 0, ... 0] ...
```

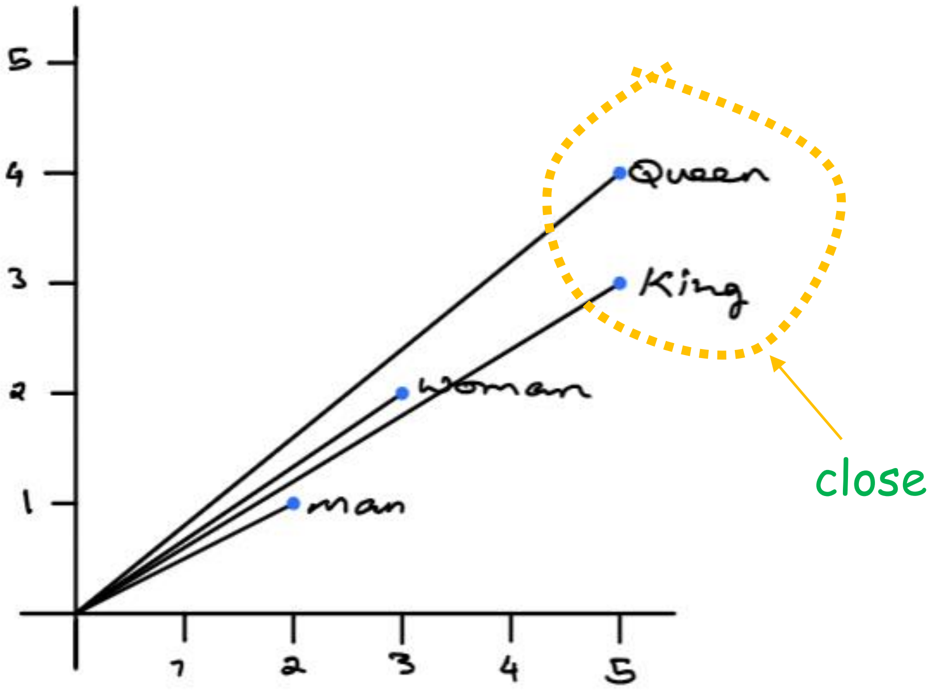
One-hot vector

good → [0, 0, 1, 0, 0, 0, ⋮, 0]

Embedding size

768 positions

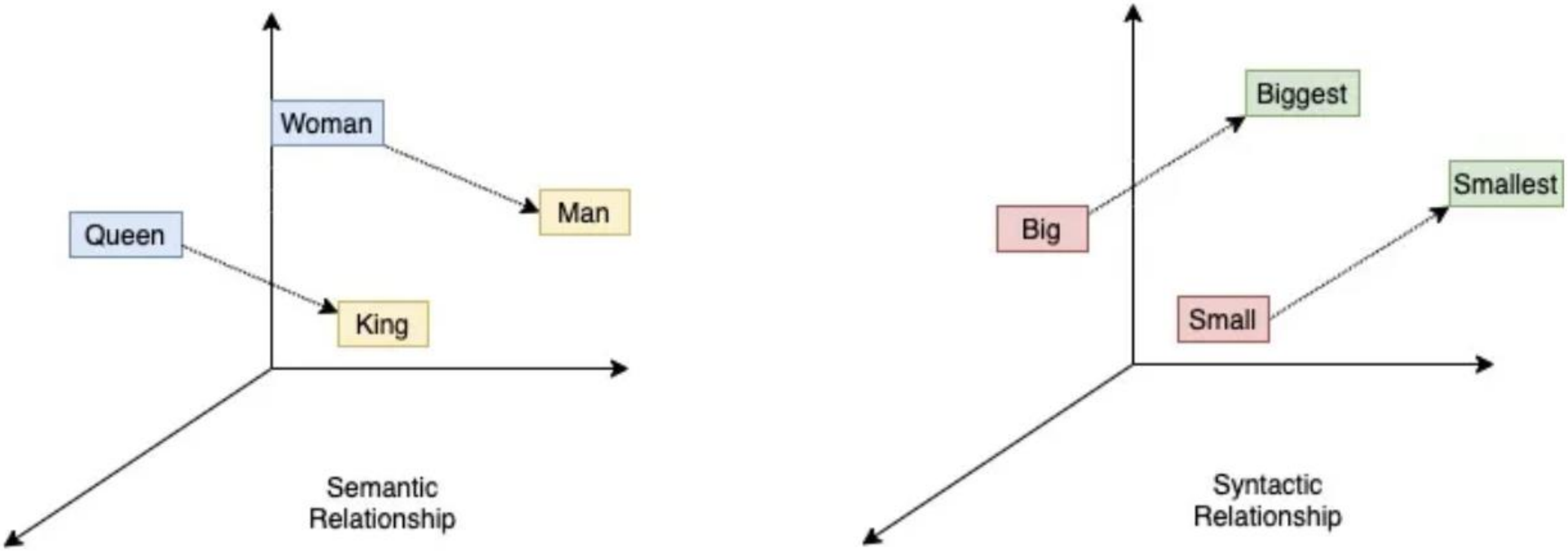
King	−	Man	+	Woman	=	Queen
[5,3]	−	[2,1]	+	[3, 2]	=	[5,4]



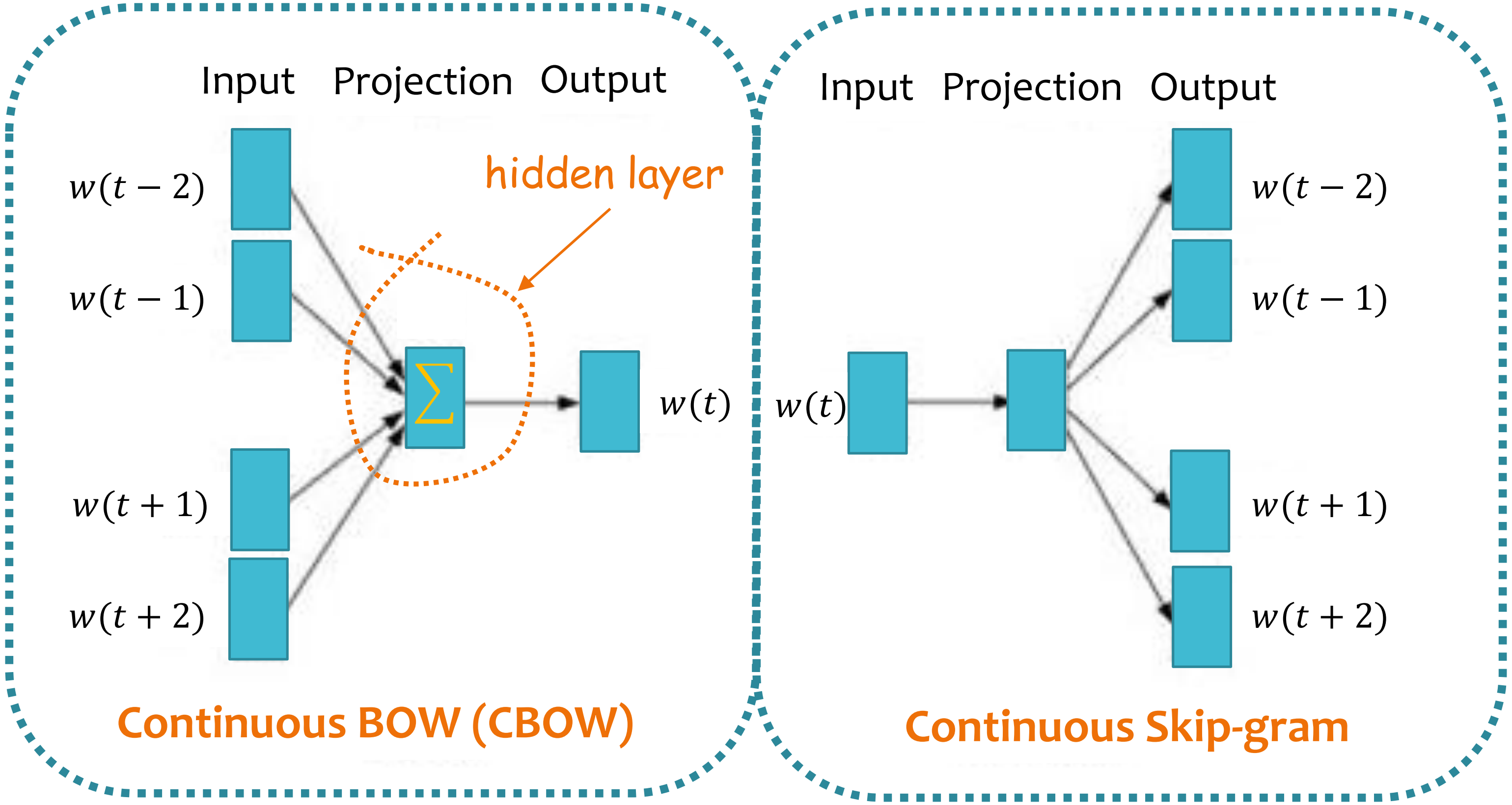
Source: https://towardsdatascience.com/word2vec-explained-49c52b4ccb71

Index term: Vector Space Model

Type of relationship in textual data



Word Embedding (Word2Vec)



- Predict target words from surrounding context

Small datasets ✓

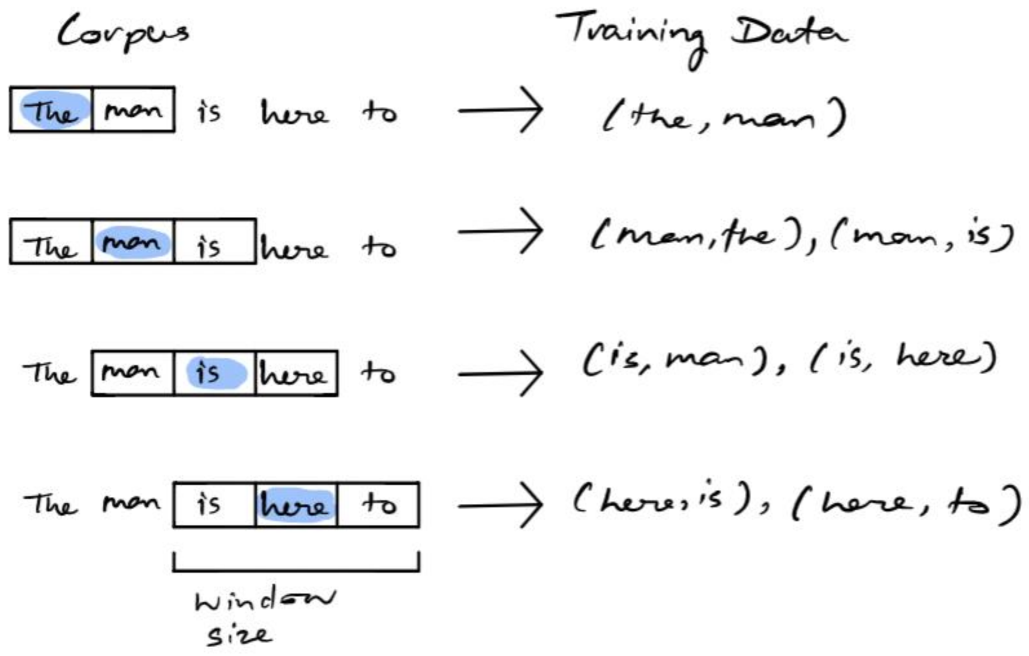
- CBOW inverse

Large datasets ✓

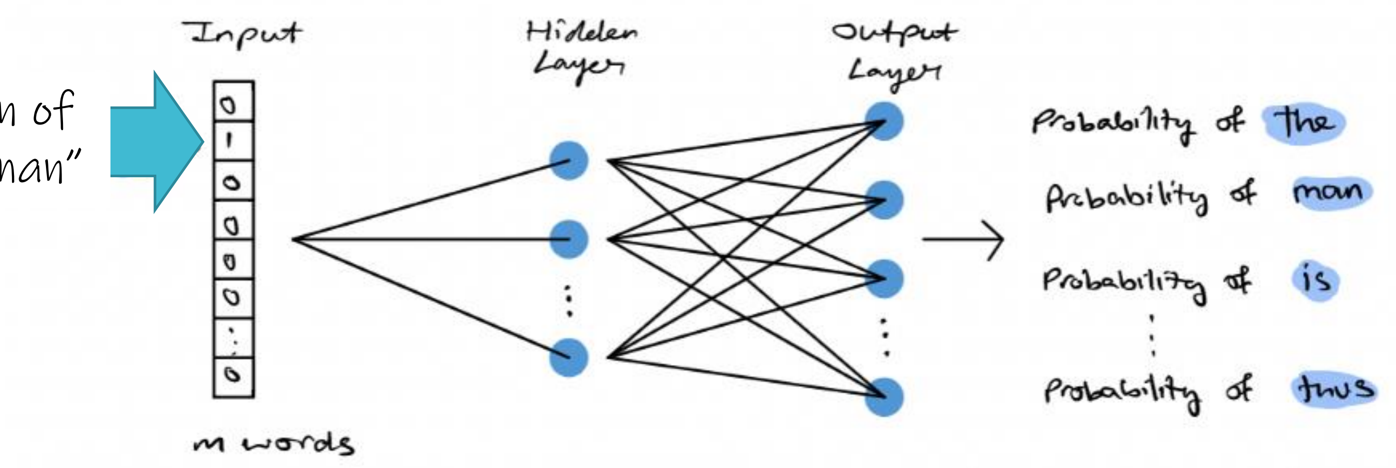
Example ➡ C_t = "the cat sits in the", T = "mat"

Continuous Skip-gram model

Generating training data



Position of word "man"



Source: https://towardsdatascience.com/word2vec-explained-49c52b4ccb71

Index term: Term frequency

- **Term frequency:** Term frequency tells you how much a term occurs in a document.

TF*IDF = TF(t,d) · IDF(t)

(Term Frequency/ tf)

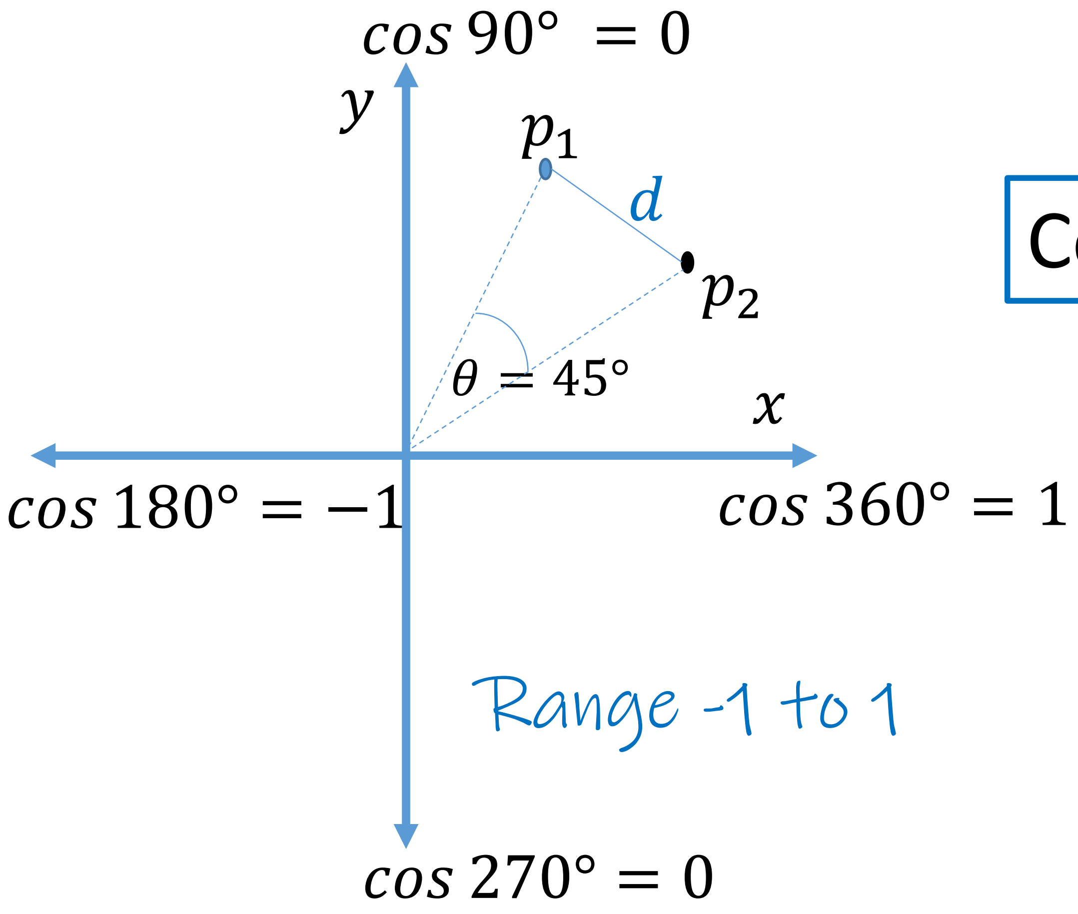
$$tf = \frac{count\ of\ term\ (t)\ in\ doc.}{num.of\ words\ in\ doc.}$$

(Inverse Document Frequency/ idf)

$$idf = log\left(\frac{docs.in\ corpus}{num.of\ docs.where\ t\ appears}\right)$$

Source paper: Using TF-IDF to Determine Word Relevance in Document Queries (2003)

Cosine similarity & cosine distance



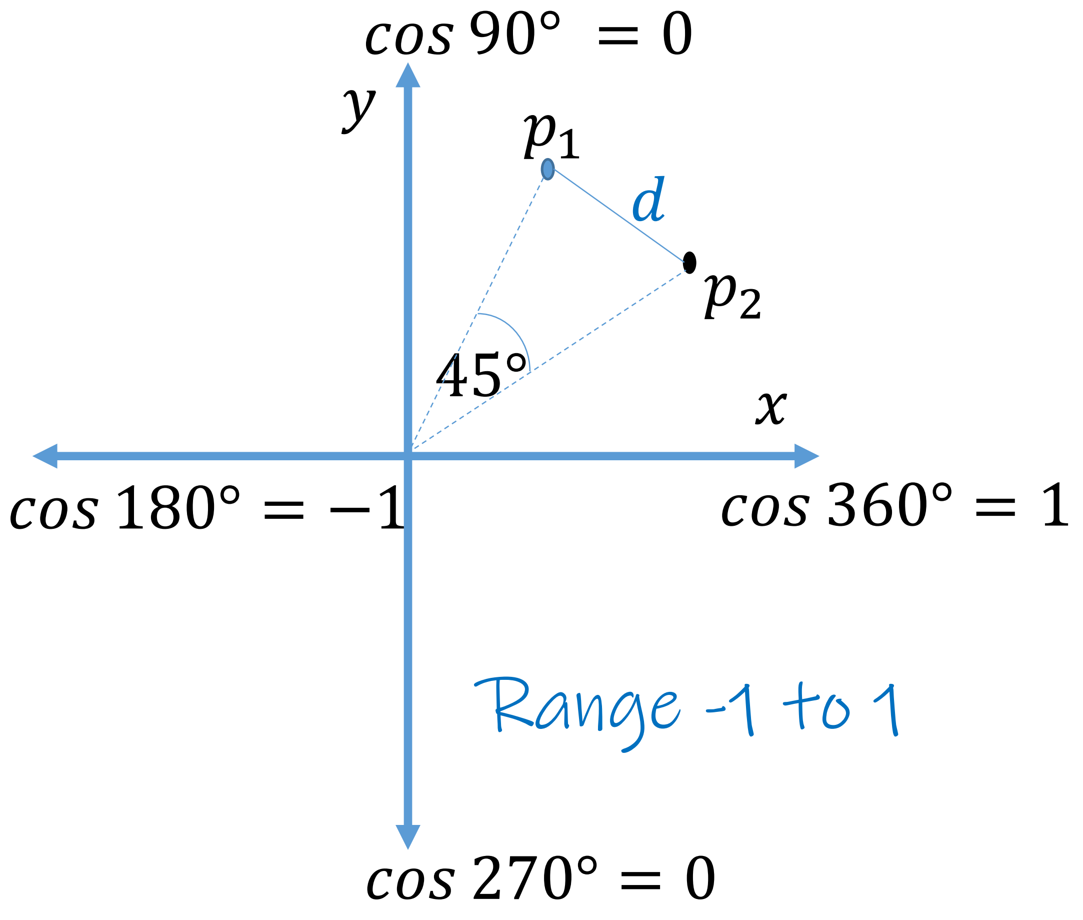
Angle between p_1 & p_2

Cosine Similarity = $Cos\ \theta$

- $cos\,45° = 0.53$
- $cos\,90° = 0$
- $cos\,0° = 1$

Similarity	Distance
↓	↑
↑	↓

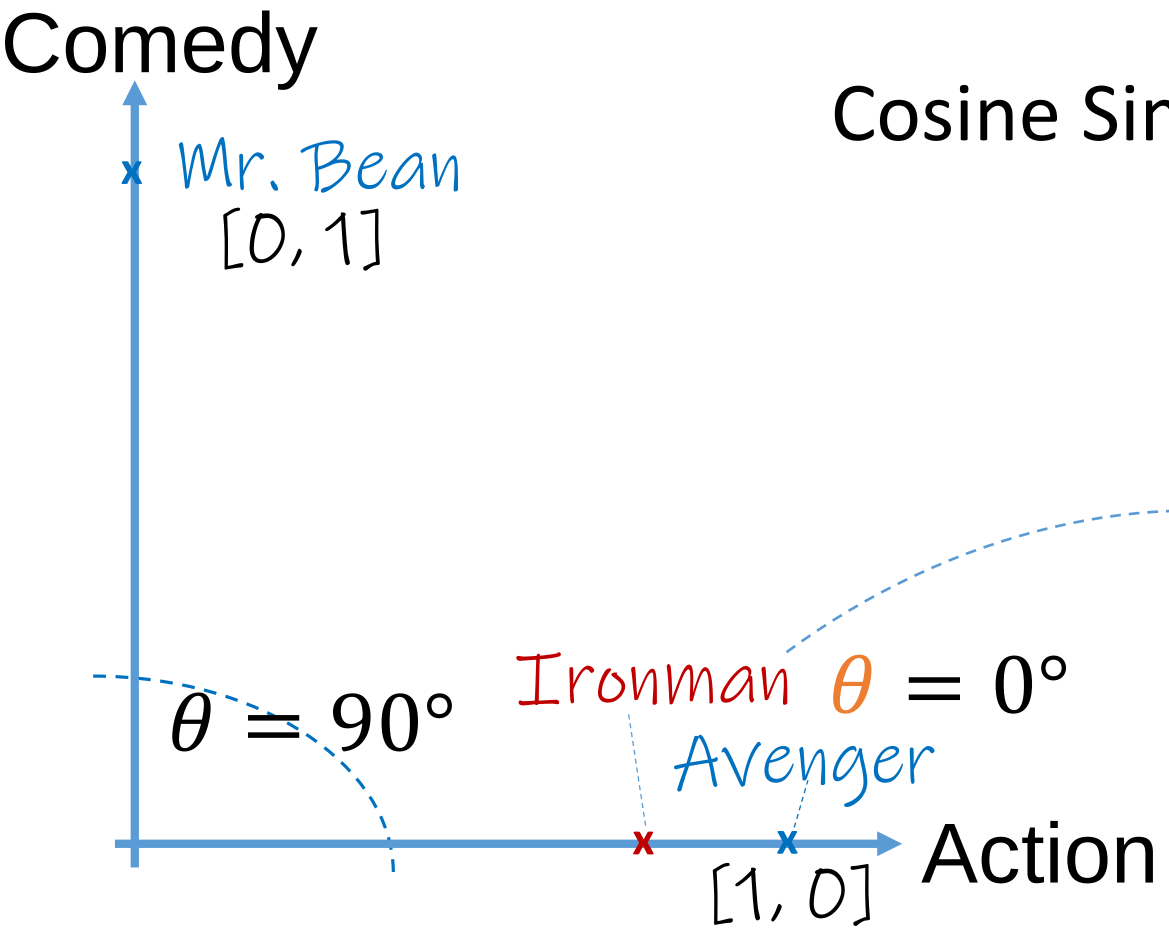
Cosine similarity & cosine distance



$\boxed{\text{Cosine Distance} = 1 - Cos.Sim.}$

Similarity	Distance
$cos\, 45° = 0.53$	$1 - 0.53 = 0.47$
$cos\, 90° = 0$	$1 - 0 = 1$
$cos\, 0° = 1$	$1 - 1 = 0$

Similarity in Recommendation system



Cosine Sim. = $Cos\ \theta$

= $Cos\ 90$

= 0

= $Cos\ \theta$

= $Cos\ 0$

= 1

∴ Viewers from Avenger will not get recommendation from Mr. Bean, and otherwise

Cosine similarity (Word occurrence)

```
# Scikit Learn
from sklearn.feature_extraction.text import CountVectorizer
import pandas as pd

# Define the documents
corpus = ["I'd like an apple",
          "An apple a day keeps the doctor away",
          "Never compare an apple to an orange",
          "I prefer scikit-learn to Orange",
          "The scikit-learn docs are Orange and Blue"]

# Create the Document Term Matrix
count_vectorizer = CountVectorizer(stop_words='english')
count_vectorizer = CountVectorizer()
sparse_matrix = count_vectorizer.fit_transform(corpus) #(documents)

# OPTIONAL: Convert Sparse Matrix to Pandas Dataframe if you want to see the word frequencies.
doc_term_matrix = sparse_matrix.todense()
df = pd.DataFrame(doc_term_matrix,
                  columns=count_vectorizer.get_feature_names())

df
```

Cosine similarity (Word occurrence)

Word frequencies

Output:

	an	and	apple	are	away	blue	compare	day	docs	doctor	keeps	learn	like	never	orange	prefer	scikit	the	to
0	1	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
1	1	0	1	0	1	0	0	1	0	1	1	0	0	0	0	0	0	1	0
2	2	0	1	0	0	0	1	0	0	0	0	0	0	1	1	0	0	0	1
3	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	1	0	1
4	0	1	0	1	0	1	0	0	1	0	0	1	0	0	1	0	1	1	0

Cosine similarity (word occurrence)

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} \Rightarrow \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}}$$

```python
# Compute Cosine Similarity
from sklearn.metrics.pairwise import cosine_similarity
print(cosine_similarity(df, df))
```

```python
from sklearn.metrics.pairwise import linear_kernel
```

```python
cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)
```

Output:

```
[[1.         0.43643578 0.57735027 0.         0.        ]
 [0.43643578 1.         0.37796447 0.         0.13363062]
 [0.57735027 0.37796447 1.         0.2981424  0.11785113]
 [0.         0.         0.2981424  1.         0.47434165]
 [0.         0.13363062 0.11785113 0.47434165 1.        ]]
```

Cosine similarity (tf*idf)

```
from sklearn.feature_extraction.text import TfidfVectorizer
import numpy as np

corpus = ["I'd like an apple",
          "An apple a day keeps the doctor away",
          "Never compare an apple to an orange",
          "I prefer scikit-learn to Orange",
          "The scikit-learn docs are Orange and Blue"]
vect = TfidfVectorizer(min_df=1, stop_words="english")
tfidf = vect.fit_transform(corpus)
pairwise_similarity = tfidf * tfidf.T
```

Library for math and logic operation in array: https://numpy.org/

CountVectorizer
+
TfidfTransformer

Sparse matrix → square in shape; number of rows & columns == to the number of documents in the corpus

Cosine similarity (tf*idf)

convert the sparse matrix to an array

```
pairwise_similarity.toarray()
```

```
array([[1.        , 0.17668795, 0.27056873, 0.        , 0.        ],
       [0.17668795, 1.        , 0.15439436, 0.        , 0.        ],
       [0.27056873, 0.15439436, 1.        , 0.19635649, 0.16815247],
       [0.        , 0.        , 0.19635649, 1.        , 0.54499756],
       [0.        , 0.        , 0.16815247, 0.54499756, 1.        ]])
```

Find the most similar documents in corpus

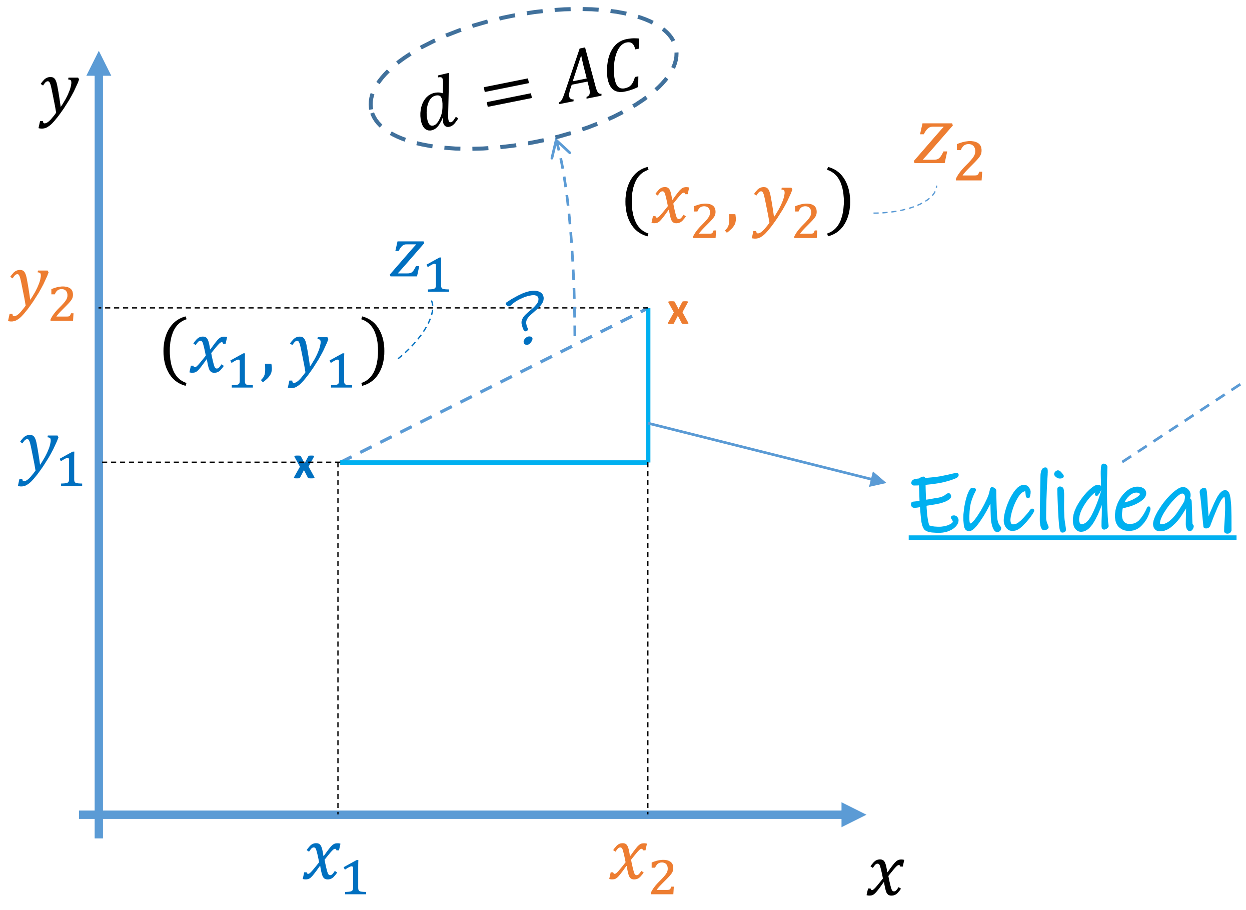
```
arr = pairwise_similarity.toarray()
np.fill_diagonal(arr, np.nan)

input_doc = "The scikit-learn docs are Orange and Blue"
input_idx = corpus.index(input_doc)
```

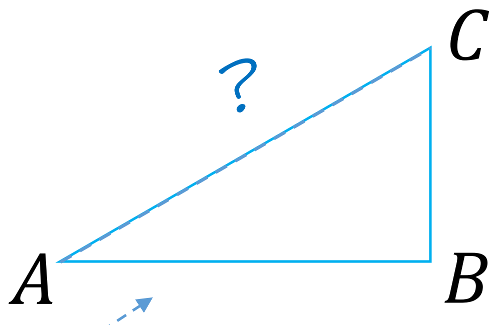
```
result_idx = np.nanargmax(arr[input_idx])
corpus[result_idx]
```

```
'I prefer scikit-learn to Orange'
```

Find the index of the most similar docs

Euclidean distance $(L2\ Norms)$





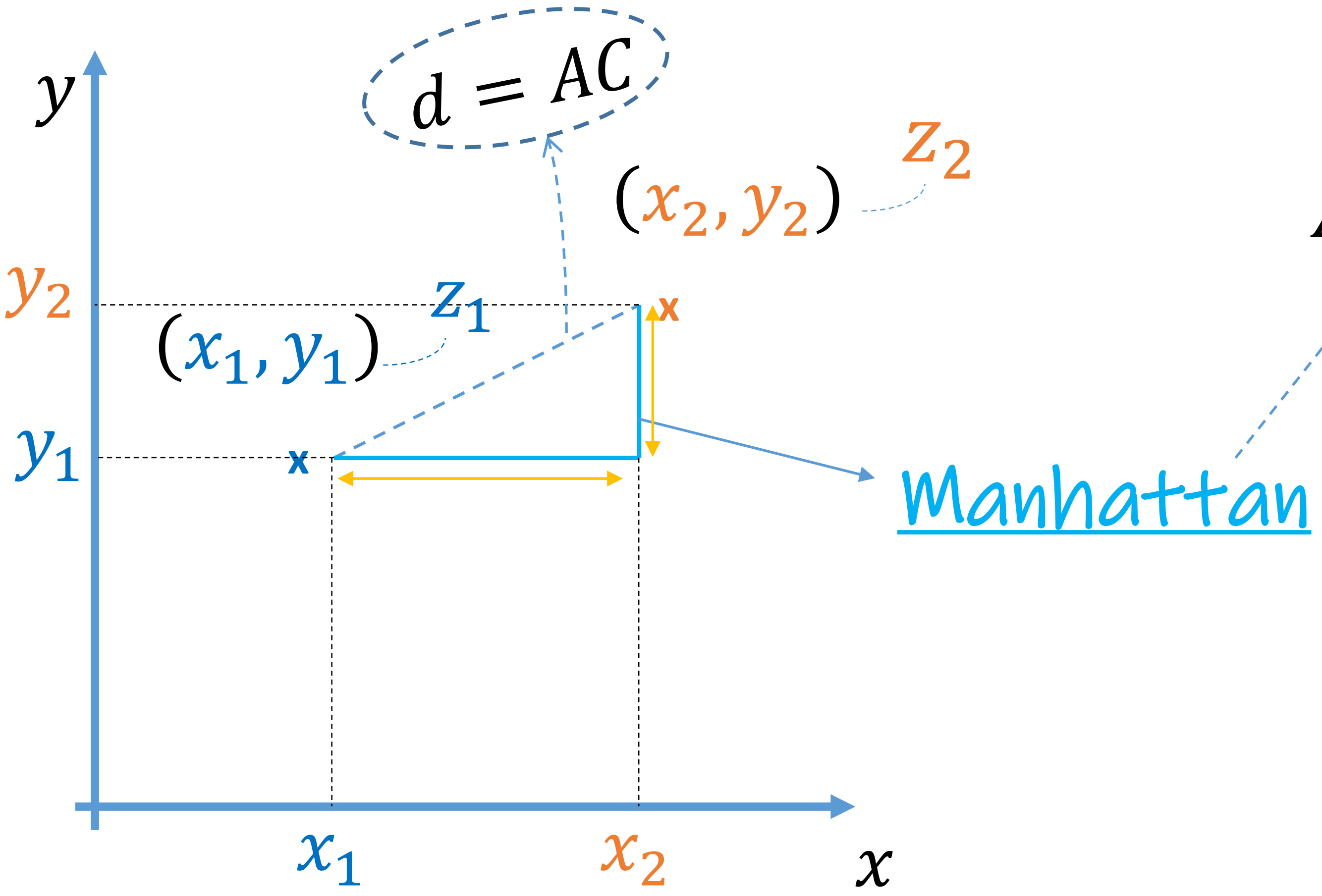
$$AC^2 = AB^2 + BC^2$$

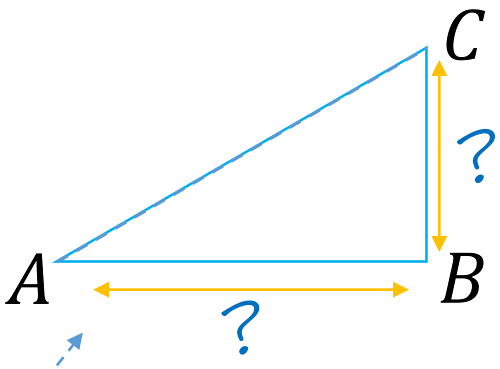
$$AC = \sqrt{AB^2 + BC^2}$$

$$= \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

$$\ldots + (z_2 - z_1)^2$$

Manhattan distance ($L1\ Norms$)





$$= |(x_2 - x_1)| + |(y_2 - y_1)|$$

```python
import itertools
import numpy as np
from scipy.spatial.distance import cityblock

for idx_1, idx_2 in itertools.combinations(range(tfidf.shape[0]), 2):
    v1, v2 = map(lambda idx: tfidf.toarray()[idx], (idx_1, idx_2))
    print(f"{(idx_1, idx_2)}"\
          f" - Euclidean: {np.linalg.norm(v1 - v2):.3f}"\
          f" - Manhattan: {cityblock(v1, v2):.3f}")
```

```
(0, 1) - Euclidean: 1.283 - Manhattan: 2.966
(0, 2) - Euclidean: 1.208 - Manhattan: 2.113
(0, 3) - Euclidean: 1.414 - Manhattan: 3.367
(0, 4) - Euclidean: 1.414 - Manhattan: 3.599
(1, 2) - Euclidean: 1.300 - Manhattan: 3.277
(1, 3) - Euclidean: 1.414 - Manhattan: 4.194
(1, 4) - Euclidean: 1.414 - Manhattan: 4.426
(2, 3) - Euclidean: 1.268 - Manhattan: 2.871
(2, 4) - Euclidean: 1.290 - Manhattan: 3.219
(3, 4) - Euclidean: 0.954 - Manhattan: 1.833
```

Pearson correlation coefficient

Purpose: a measure of the linear relationship between two continuous variables

Co-variance $\Rightarrow$ $$Cov(x,y) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu_x)(y_i - \mu_y)$$

Pearson CC $\Rightarrow$ $$\rho_{(x,y)} = \frac{Cov(x,y)}{\sigma_x \sigma_y} \Rightarrow -1 \le \rho \le 1$$

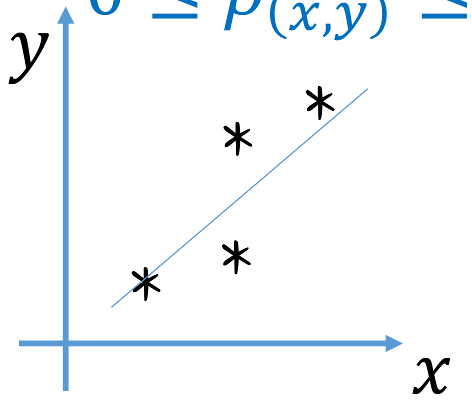
where $Cov(x,y)$ is the covariance of variables
σ_x and σ_y are the standard deviation of variables

Pearson correlation Coefficient

x	y	
Height	Weight	
$x \uparrow$	$\uparrow y$	$\Rightarrow$ + CV
$x \uparrow$	$\downarrow y$	$\Rightarrow$ - CV

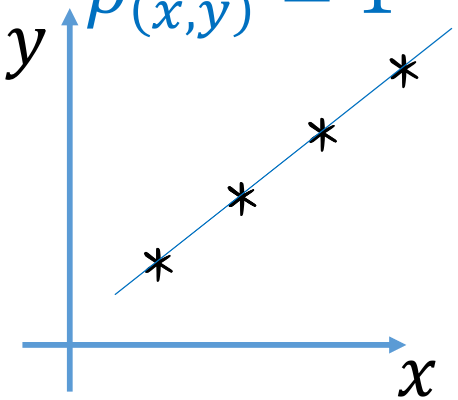
*Strength
*Relationship direction

$0 \le \rho_{(x,y)} \le 1$



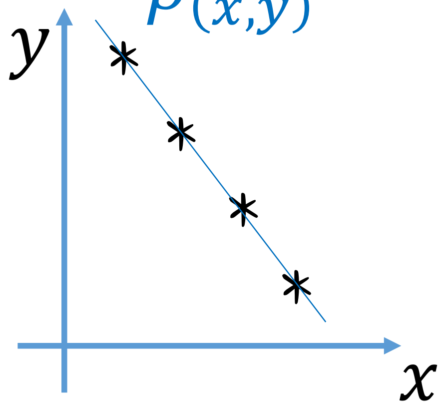
$x \uparrow \quad y \uparrow$

$\rho_{(x,y)} = 1$



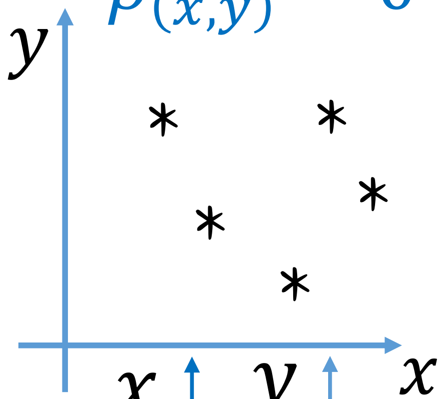
$x \uparrow \quad y \uparrow$

$\rho_{(x,y)} = -1$



$x \downarrow \quad y \uparrow$

$\rho_{(x,y)} = 0$

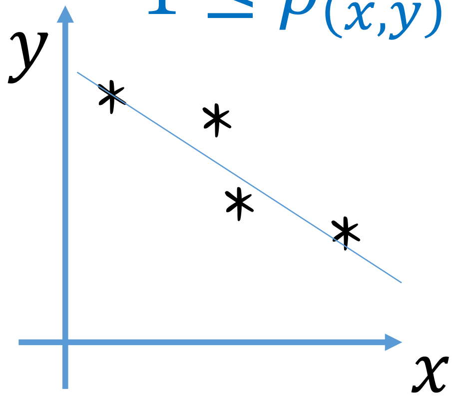


$x \uparrow \quad y \uparrow$

$x \uparrow \quad y \downarrow$

$x \downarrow \quad y \uparrow$

$-1 \le \rho_{(x,y)} \le 0$



$x \downarrow \quad y \uparrow$

```python
from scipy import stats

formater = lambda t: ', '.join(('%.3f' % f) for f in t)

for idx_1, idx_2 in itertools.combinations(range(tfidf.shape[0]), 2):
    v1, v2 = map(lambda idx: tfidf.toarray()[idx], (idx_1, idx_2))
    print(f"{(idx_1, idx_2)}"\
          f" - Pearson: {formater(stats.pearsonr(v1, v2))}")
```

```
(0, 1) - Pearson: -0.082, 0.791
(0, 2) - Pearson: 0.110, 0.721
(0, 3) - Pearson: -0.274, 0.365
(0, 4) - Pearson: -0.324, 0.280
(1, 2) - Pearson: -0.194, 0.526
(1, 3) - Pearson: -0.511, 0.074
(1, 4) - Pearson: -0.604, 0.029
(2, 3) - Pearson: -0.085, 0.784
(2, 4) - Pearson: -0.173, 0.571
(3, 4) - Pearson: 0.315, 0.294
```

Spearman correlation coefficient

Purpose: a non-parametric measure of the strength and direction of association between two ranked variables. (non-linear variables relationship, and ranked data)

$$r_s = \rho_{R(X),R(Y)} = \frac{Cov\big(R(X), R(Y)\big)}{\sigma_{R(X)}\sigma_{R(Y)}}$$

$$r_s = 1 - \frac{6\sum d_i^2}{n(n^2-1)}$$

where

$Cov(x, y)$ is the covariance of the rank variables

σ_x and σ_y are the standard deviation of the rank variables

$d_i = R(X_i) - R(Y_i)$ is the difference between the two ranks of each variable

```python
from scipy import stats

formater = lambda t: ', '.join(('%.3f' % f) for f in t)

for idx_1, idx_2 in itertools.combinations(range(tfidf.shape[0]), 2):
    v1, v2 = map(lambda idx: tfidf.toarray()[idx], (idx_1, idx_2))
    print(f"{(idx_1, idx_2)}"\
          f" - Spearmanr: {formater(stats.spearmanr(v1, v2))}")
```

```
(0, 1) - Spearmanr: -0.056, 0.856
(0, 2) - Spearmanr: 0.195, 0.522
(0, 3) - Spearmanr: -0.278, 0.358
(0, 4) - Spearmanr: -0.325, 0.279
(1, 2) - Spearmanr: -0.188, 0.538
(1, 3) - Spearmanr: -0.508, 0.077
(1, 4) - Spearmanr: -0.593, 0.033
(2, 3) - Spearmanr: -0.082, 0.790
(2, 4) - Spearmanr: -0.185, 0.544
(3, 4) - Spearmanr: 0.273, 0.367
```

Thank you
Q & A