Lecture 6
Distances between Observations

Ames House Prices

Today's Data: Houses in Ames, IA

Area measurements (Gr Liv Area, Lot Area, Garage Area, Total Bsmt SF) → sq ft
Count / unit (Bedroom AbvGr, Full Bath, Garage Cars) → integer / unit
Price → USD
Historical information (Year Built, Yr Sold) → numeric / year

Ames House Prices

Today's Data: Houses in Ames, IA

```
import pandas as pd
df = pd.read_table("https://datasci112.stanford.edu/data/housing.tsv")
df
```

Ames House Prices

Today's Data: Houses in Ames, IA

Tab-Separated Values

```python
import pandas as pd
df = pd.read_table("https://datasci112.stanford.edu/data/housing.tsv")
df
```

	PID	Gr Liv Area	Bedroom AbvGr	Full Bath	Half Bath	...	Mo Sold	Yr Sold	Sale Type	Sale Condition	SalePrice
0	526301100	1656	3	1	0	...	5	2010	WD	Normal	215000
1	526350040	896	2	1	0	...	6	2010	WD	Normal	105000
2	526351010	1329	3	1	1	...	6	2010	WD	Normal	172000
3	526353030	2110	3	2	1	...	4	2010	WD	Normal	244000
4	527105010	1629	3	2	1	...	3	2010	WD	Normal	189900
...	...	...	...	...	...	...	...	...	...	...	...
2925	923275080	1003	3	1	0	...	3	2006	WD	Normal	142500
2926	923276100	902	2	1	0	...	6	2006	WD	Normal	131000
2927	923400125	970	3	1	0	...	7	2006	WD	Normal	132000
2928	924100070	1389	2	1	0	...	4	2006	WD	Normal	170000
2929	924151050	2000	3	2	1	...	11	2006	WD	Normal	188000

2930 rows × 81 columns

Gr Liv Area: Ground living area
Bedroom AbvGr: Bedroom above ground
Mo Sold: Month sold
WD: Waranty deed (normal, garantili tapu)

E.g. Gr Liv Area = 2000 → the above-ground living area is 2000 sq ft (approximately 186 m²)

Square Footage and Bedrooms

We’ve seen how to visualize and summarize the relationship between two quantitative variables.

Square Footage and Bedrooms

We've seen how to visualize and summarize the relationship between two quantitative variables.

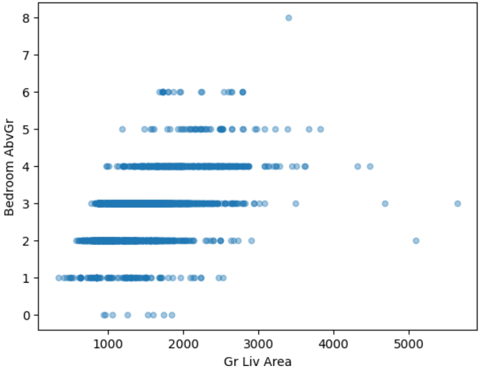
```
df.plot.scatter(x="Gr Liv Area", y="Bedroom AbvGr", alpha=0.4)
df[["Gr Liv Area", "Bedroom AbvGr"]].corr()
```

Square Footage and Bedrooms

We've seen how to visualize and summarize the relationship between two quantitative variables.

```
df.plot.scatter(x="Gr Liv Area", y="Bedroom AbvGr", alpha=0.4)
df[["Gr Liv Area", "Bedroom AbvGr"]].corr()
```

	Gr Liv Area	Bedroom AbvGr
Gr Liv Area	1.000000	0.516808
Bedroom AbvGr	0.516808	1.000000



Type: Scatter plot
X-axis: Gr Liv Area (living area in sq ft)
Y-axis: Bedroom AbvGr (#bedrooms above ground)
Trend: More bedrooms → generally larger living area
Variation: Large spread within each bedroom count
Density: Most houses have 2–4 bedrooms
Outliers: Few very large houses with high Gr Liv Area
every dot a house

1 Distances between Observations

2 Variable Scaling

3 Calculating Distances in Scikit-Learn

4 Summary

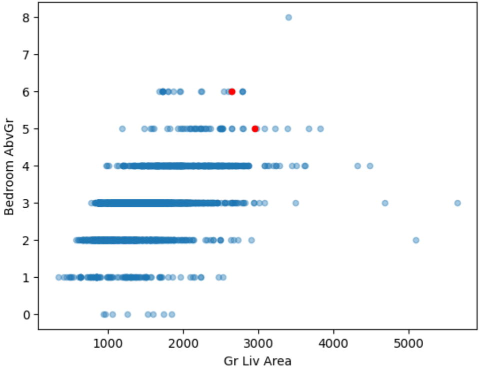
Quantifying Similarity

How do we measure how “similar” two observations are?

Quantifying Similarity

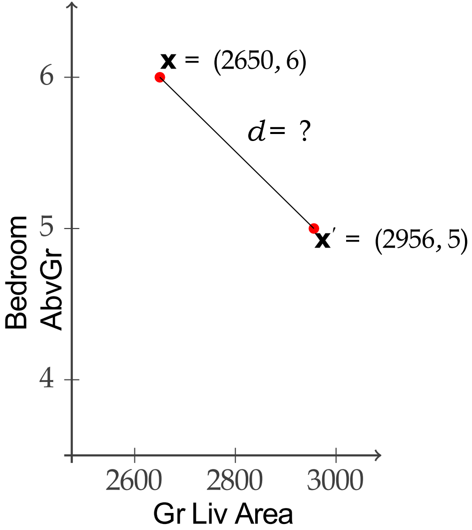
How do we measure how “similar” two observations are?

For example, how would we quantify how similar the following two houses are?



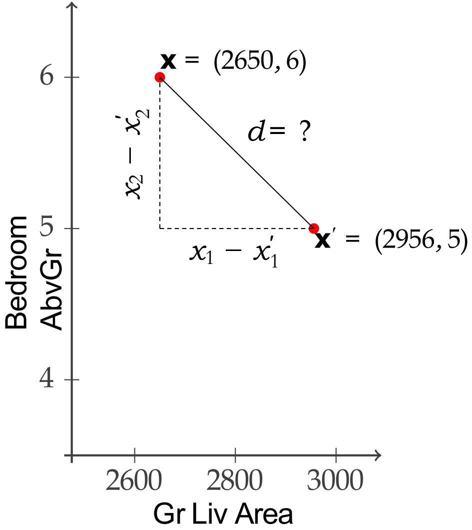
Calculating Distance

One way is to calculate the distance between the two points.



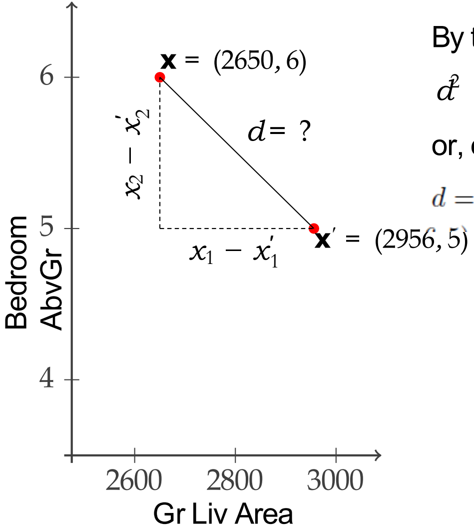
Calculating Distance

One way is to calculate the distance between the two points.



Calculating Distance

One way is to calculate the distance between the two points.



By the Pythagorean Theorem,

$$d^2 = (x_1 - x_1')^2 + (x_2 - x_2')^2$$

or, equivalently,

$$d = \sqrt{(x_1 - x_1')^2 + (x_2 - x_2')^2}.$$

Calculating Distance in Higher Dimensions

The distance formula generalizes to any number of variables m.

$$d(\mathbf{x}, \mathbf{x}') = \sqrt{\sum_{j=1}^{m} (x_j - x'_j)^2}$$

Distance ≈ 0: Houses have almost identical features.

Moderate distance: Houses are similar but with some differences.

Very large distance: Houses differ greatly in size or room counts.

Calculating Distance in Higher Dimensions

The distance formula generalizes to any number of variables m.

$$d(\mathbf{x}, \mathbf{x}') = \sqrt{\sum_{j=1}^{m} (x_j - x'_j)^2}$$

Let's calculate the distance between these two houses, based on $m = 4$ variables: square footage, bedrooms, full baths, and half baths.

1 Distances between Observations

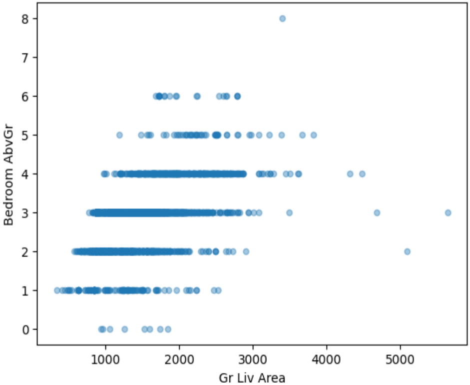
2 Variable Scaling

3 Calculating Distances in Scikit-Learn

4 Summary

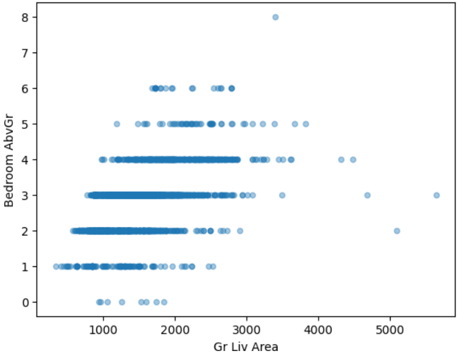
The Importance of Scale

This plot is misleading.

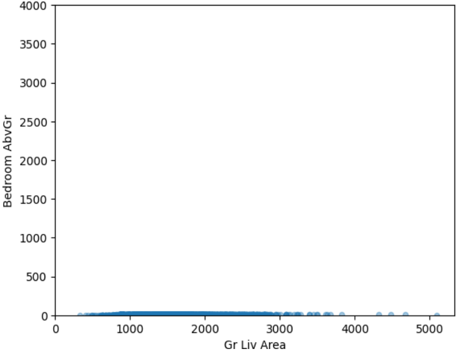


The Importance of Scale

This plot is misleading.

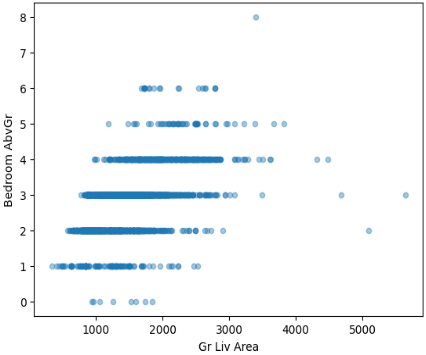


The data really looks like this.

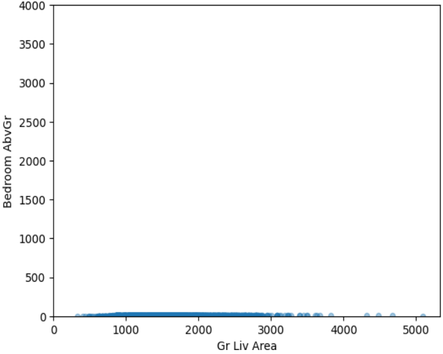


The Importance of Scale

This plot is misleading.



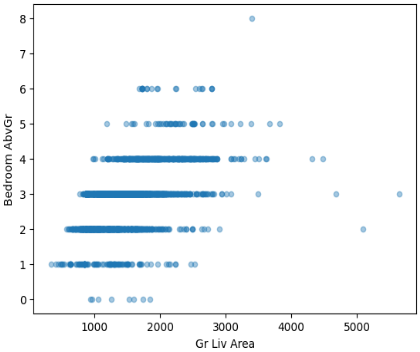
The data really looks like this.



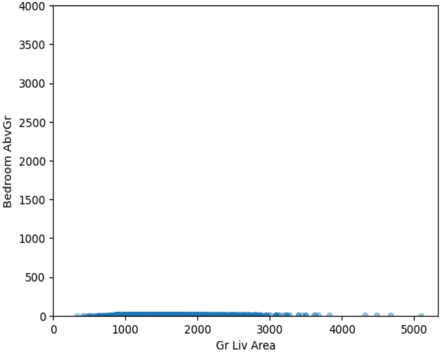
- The variation in square footage completely swamps the variation in the number of bedrooms.

The Importance of Scale

This plot is misleading.



The data really looks like this.



The variation in square footage completely swamps the variation in the number of bedrooms.

We should bring all the variables to the same scale before calculating distance.

Standardization

We’ve seen one way to bring different variables to the same scale: **standardization.**

Standardization is a type of scaling, similar to normalization

but it uses the mean (0) and standard deviation (1) instead of compressing values into a fixed range.

standart normal dist. (mean=0, sd=1) vs (mean!=0 and sd !=1)

Standardization

We've seen one way to bring different variables to the same scale: **standardization.**

$$x_i \leftarrow \frac{x_i - \bar{\mathbf{x}}}{\mathrm{sd}(\mathbf{x})}$$

```
features = ["Gr Liv Area", "Bedroom AbvGr", "Full Bath", "Half Bath"]
df_standardized = ((df[features] - df[features].mean(axis="rows")) /
                   df[features].std(axis="rows"))
df_standardized
```

obtain a new DataFrame where each column has a **mean of 0** and a **standard deviation of 1.**

Standardization

We've seen one way to bring different variables to the same scale: **standardization.**

$$x_i \leftarrow \frac{x_i - \bar{\mathbf{x}}}{\text{sd}(\mathbf{x})}$$

```
features = ["Gr Liv Area", "Bedroom AbvGr", "Full Bath", "Half Bath"]
df_standardized = ((df[features] - df[features].mean(axis="rows")) /
                   df[features].std(axis="rows"))
df_standardized
```

	Gr Liv Area	Bedroom AbvGr	Full Bath	Half Bath
0	0.309212	0.176064	-1.024618	-0.755074
1	-1.194223	-1.032058	-1.024618	-0.755074
2	-0.337661	0.176064	-1.024618	1.234464
3	1.207317	0.176064	0.783894	1.234464
4	0.255801	0.176064	0.783894	1.234464
...	...	...	...	...
2925	-0.982555	0.176064	-1.024618	-0.755074
2926	-1.182354	-1.032058	-1.024618	-0.755074
2927	-1.047836	0.176064	-1.024618	-0.755074
2928	-0.218968	-1.032058	-1.024618	-0.755074
2929	0.989715	0.176064	0.783894	1.234464

2930 rows × 4 columns

The Effect of Standardization

What did standardization do?

Each feature now has mean = 0.

Each feature now has standard deviation = 1.

All features are on the same scale.

Prevents features with larger ranges from dominating analyses.

The Effect of Standardization

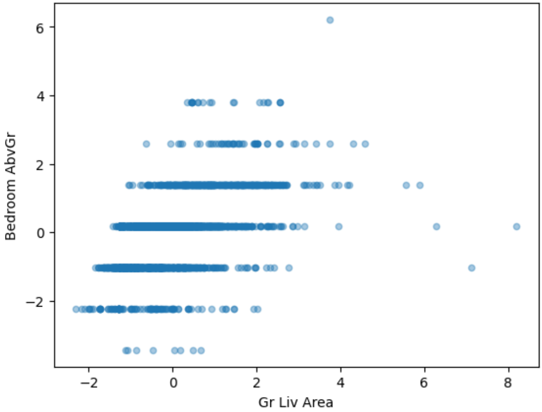
What did standardization do?

```
df_standardized.plot.scatter(
    x="Gr Liv Area", y="Bedroom AbvGr", alpha=0.4)
```

The Effect of Standardization

What did standardization do?

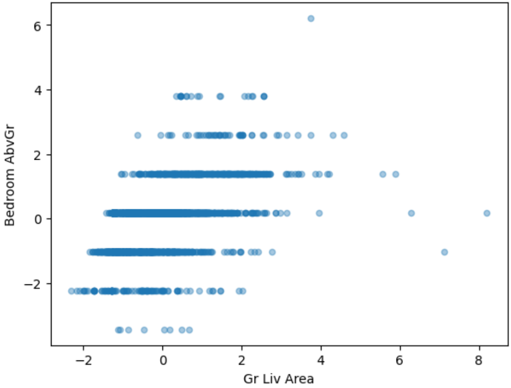
```
df_standardized.plot.scatter(
    x="Gr Liv Area", y="Bedroom AbvGr", alpha=0.4)
```



The Effect of Standardization

What did standardization do?

```
df_standardized.plot.scatter(
    x="Gr Liv Area", y="Bedroom AbvGr", alpha=0.4)
```



Standardization ensures that every variable has mean 0 and standard deviation 1 so that every variable is treated equally.

Finding the Most Similar Home

Let’s find the most similar house by calculating distances on the ***standardized*** data.

Finding the Most Similar Home

Let's find the most similar house by calculating distances on the ***standardized*** data.

```
import numpy as np
diffs = df_standardized - df_standardized.loc[1707]
dists = np.sqrt((diffs ** 2).sum(axis="columns"))
dists.sort_values()
```

Computes the difference between each row and row 1707.

Squares the differences and sums them across columns.

Takes the square root to get Euclidean distances.

Sorts distances from closest to farthest.

Finding the Most Similar Home

Let's find the most similar house by calculating distances on the ***standardized*** data.

```
import numpy as np
diffs = df_standardized - df_standardized.loc[1707]
dists = np.sqrt((diffs ** 2).sum(axis="columns"))
dists.sort_values()
```

```
1707    0.000000
160     0.043521
909     0.249254
2592    0.625113
585     0.625113
          ...
2880    8.024014
1385    8.050646
2279    8.095655
158     8.225392
2723    8.313887
Length: 2930, dtype: float64
```

Finding the Most Similar Home

Let's find the most similar house by calculating distances on the ***standardized*** data.

```
import numpy as np
diffs = df_standardized - df_standardized.loc[1707]
dists = np.sqrt((diffs ** 2).sum(axis="columns"))
dists.sort_values()
```

```
1707    0.000000
160     0.043521
909     0.249254
2592    0.625113
585     0.625113
          ...
2880    8.024014
1385    8.050646
2279    8.095655
158     8.225392
2723    8.313887
Length: 2930, dtype: float64
```

Now the most similar house to #1707 is #160, which matches our intuition when we looked at the data.

1 Distances between Observations

2 Variable Scaling

3 Calculating Distances in Scikit-Learn

4 Summary

Scaling in Scikit-Learn

The machine learning library Scikit-Learn can be used to scale variables and calculate distances.

Scaling in Scikit-Learn

- The machine learning library Scikit-Learn can be used to scale variables and calculate distances.

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

# This calculates the mean and standard deviation of each variable.
scaler.fit(df[features])

# This actually does the standardization.
array_standardized = scaler.transform(df[features])
array_standardized
```

Scaling in Scikit-Learn

The machine learning library Scikit-Learn can be used to scale variables and calculate distances.

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

# This calculates the mean and standard deviation of each variable.
scaler.fit(df[features])

# This actually does the standardization.
array_standardized = scaler.transform(df[features])
array_standardized
```

```
array([[ 0.30926506,  0.17609421, -1.02479289, -0.75520269],
       [-1.19442705, -1.03223376, -1.02479289, -0.75520269],
       [-0.33771825,  0.17609421, -1.02479289,  1.23467491],
       ...,
       [-1.04801492,  0.17609421, -1.02479289, -0.75520269],
       [-0.21900572, -1.03223376, -1.02479289, -0.75520269],
       [ 0.9898836 ,  0.17609421,  0.7840283 ,  1.23467491]])
```

scaled numerical array

Scaling in Scikit-Learn

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

# This calculates the mean and standard deviation of each variable.
scaler.fit(df[features])

# This actually does the standardization.
array_standardized = scaler.transform(df[features])
array_standardized
```

```
array([[ 0.30926506,  0.17609421, -1.02479289, -0.75520269],
       [-1.19442705, -1.03223376, -1.02479289, -0.75520269],
       [-0.33771825,  0.17609421, -1.02479289,  1.23467491],
       ...,
       [-1.04801492,  0.17609421, -1.02479289, -0.75520269],
       [-0.21900572, -1.03223376, -1.02479289, -0.75520269],
       [ 0.9898836 ,  0.17609421,  0.7840283 ,  1.23467491]])
```

The result is the same as before, but it's in an array, not a DataFrame!

Calculating Distances in Scikit-Learn

Now let's calculate the distance between house #1707 and *all* houses, using Scikit-Learn.

Calculating Distances in Scikit-Learn

Now let's calculate the distance between house #1707 and *all* houses, using Scikit-Learn.

```
from sklearn.metrics import pairwise_distances

dists = pairwise_distances(array_standardized[[1707]], array_standardized)
dists
```

Calculating Distances in Scikit-Learn

Now let's calculate the distance between house #1707 and *all* houses, using Scikit-Learn.

```
from sklearn.metrics import pairwise_distances

dists = pairwise_distances(array_standardized[[1707]], array_standardized)
dists
```

```
array([[4.43704819, 6.08145351, 4.41300228, ..., 5.33963844, 5.4757908 ,
        3.06886734]])
```

Calculating Distances in Scikit-Learn

Now let's calculate the distance between house #1707 and *all* houses, using Scikit-Learn.

```
from sklearn.metrics import pairwise_distances

dists = pairwise_distances(array_standardized[[1707]], array_standardized)
dists
```

```
array([[4.43704819, 6.08145351, 4.41300228, ..., 5.33963844, 5.4757908 ,
        3.06886734]])
```

We should sort these values...

```
dists[0].sort()  # the sorting is done in place
dists
```

```
array([[0.        , 0.04352793, 0.24929632, ..., 8.09703665, 8.22679607,
        8.31530563]])
```

Calculating Distances in Scikit-Learn

Now let's calculate the distance between house #1707 and *all* houses, using Scikit-Learn.

```
from sklearn.metrics import pairwise_distances

dists = pairwise_distances(array_standardized[[1707]], array_standardized)
dists
```

```
array([[4.43704819, 6.08145351, 4.41300228, ..., 5.33963844, 5.4757908 ,
        3.06886734]])
```

We should sort these values...

```
dists[0].sort()  # the sorting is done in place
dists
```

```
array([[0.        , 0.04352793, 0.24929632, ..., 8.09703665, 8.22679607,
        8.31530563]])
```

...but `.sort()` doesn't quite give us what we want.

Calculating Distances in Scikit-Learn

Now let's calculate the distance between house #1707 and *all* houses, using Scikit-Learn.

```
from sklearn.metrics import pairwise_distances

dists = pairwise_distances(array_standardized[[1707]], array_standardized)
dists
```

```
array([[4.43704819, 6.08145351, 4.41300228, ..., 5.33963844, 5.4757908 ,
        3.06886734]])
```

We should sort these values...

```
dists[0].sort()  # the sorting is done in place
dists
```

```
array([[0.        , 0.04352793, 0.24929632, ..., 8.09703665, 8.22679607,
        8.31530563]])
```

...but `.sort()` doesn't quite give us what we want.

We want to know *which* house had the smallest distance, not the value of that distance.

```
dists[0].argsort()
```

Calculating Distances in Scikit-Learn

Now let's calculate the distance between house #1707 and *all* houses, using Scikit-Learn.

```
from sklearn.metrics import pairwise_distances

dists = pairwise_distances(array_standardized[[1707]], array_standardized)
dists
```

```
array([[4.43704819, 6.08145351, 4.41300228, ..., 5.33963844, 5.4757908 ,
        3.06886734]])
```

argsort() returns the indices that would sort an array or Series.
The smallest values' indices come first, largest last.
Useful for ranking or finding nearest/farthest items without changing the original array.

We should sort these values...

```
dists[0].sort()  # the sorting is done in place
dists
```

```
array([[0.        , 0.04352793, 0.24929632, ..., 8.09703665, 8.22679607,
        8.31530563]])
```

...but `.sort()` doesn't quite give us what we want.

We want to know *which* house had the smallest distance, not the value of that distance.

```
dists[0].argsort()
```

```
array([1707,  160,  909, ..., 2279,  158, 2723])
```

The Distance Matrix

We can also calculate the distance between every pair of observations and store the result in a matrix.

The Distance Matrix

We can also calculate the distance between every pair of observations and store the result in a matrix.

```
dist_matrix = pairwise_distances(array_standardized)
dist_matrix
```

The Distance Matrix

We can also calculate the distance between every pair of observations and store the result in a matrix.

```
dist_matrix = pairwise_distances(array_standardized)
dist_matrix
```

```
array([[0.        , 1.92902733, 2.09241494, ..., 1.35727998, 1.31875946, 2.77393016],
       [1.92902733, 0.        , 2.48064897, ..., 1.21716597, 0.97542133, 3.66915745],
       [2.09241494, 2.48064897, 0.        , ..., 2.11284979, 2.33104312, 2.24373812],
       ...,
       [1.35727998, 1.21716597, 2.11284979, ..., 0.        , 1.4653712 , 3.37408911],
       [1.31875946, 0.97542133, 2.33104312, ..., 1.4653712 , 0.        , 3.1863642 ],
       [2.77393016, 3.66915745, 2.24373812, ..., 3.37408911, 3.1863642 , 0.        ]])
```

The Distance Matrix

We can also calculate the distance between every pair of observations and store the result in a matrix.

```
dist_matrix = pairwise_distances(array_standardized)
dist_matrix
```

```
array([[0.        , 1.92902733, 2.09241494, ..., 1.35727998, 1.31875946, 2.77393016],
       [1.92902733, 0.        , 2.48064897, ..., 1.21716597, 0.97542133, 3.66915745],
       [2.09241494, 2.48064897, 0.        , ..., 2.11284979, 2.33104312, 2.24373812],
       ...,
       [1.35727998, 1.21716597, 2.11284979, ..., 0.        , 1.4653712 , 3.37408911],
       [1.31875946, 0.97542133, 2.33104312, ..., 1.4653712 , 0.        , 3.1863642 ],
       [2.77393016, 3.66915745, 2.24373812, ..., 3.37408911, 3.1863642 , 0.        ]])
```

Note that row #1707 of this matrix contains the `dists` we calculated previously.

```
dist_matrix[1707]
```

```
array([4.43704819, 6.08145351, 4.41300228, ..., 5.33963844, 5.4757908 ,
       3.06886734])
```

1 Distances between Observations

2 Variable Scaling

3 Calculating Distances in Scikit-Learn

4 Summary

Summary

To measure similarity between two observations, we calculate the distance between them.

Summary

To measure similarity between two observations, we calculate the distance between them.

We need to first make sure that all the variables are on the same scale.

Summary

To measure similarity between two observations, we calculate the distance between them.

We need to first make sure that all the variables are on the same scale.

There are actually different choices of scalings and distance metrics, which you will explore in section tomorrow.

Scaling Variables

- Standardization

$$x_i \leftarrow \frac{x_i - \bar{\mathbf{x}}}{\mathrm{sd}(\mathbf{x})}$$

- Min-Max Scaling

$$x_i \leftarrow \frac{x_i - \min(\mathbf{x})}{\max(\mathbf{x}) - \min(\mathbf{x})}$$

Distance Metrics

- Euclidean (ℓ_2)

$$\sqrt{\sum_{j=1}^{m}(x_j - x_j')^2}$$

- Manhattan (ℓ_1)

$$\sum_{j=1}^{m}|x_j - x_j'| \quad ,$$