Lecture 10-1: Principal Component Analysis

Dr. James Chen, Animal Data Scientist, School of Animal Sciences

Recap: Model Selection

Do we need all the available variables?

We can design several models with different combinations of variables:

- Model A, full model: all eight variables
- Model B, reduced model: `age`, `bmi`, `children`, `smoker_yes`, `region_southeast`, `region_southwest`
- Model C, reduced model: `age`, `bmi`, `smoker_yes`, `region_southeast`
- Model D, reduced model: `age`, `smoker_yes`

age	bmi	children	sex_male	smoker_yes	region_northwest	region_southeast	region_southwest
19	27.900	0	0	1	0	0	1
18	33.770	1	1	0	0	1	0
28	33.000	3	1	0	0	1	0
33	22.705	0	1	0	1	0	0
32	28.880	0	1	0	1	0	0
...	...	...	...	...	...	...	...
50	30.970	3	1	0	1	0	0
18	31.920	0	0	0	0	0	0
18	36.850	0	0	0	0	1	0
21	25.800	0	0	0	0	0	1
61	29.070	0	0	1	1	0	0

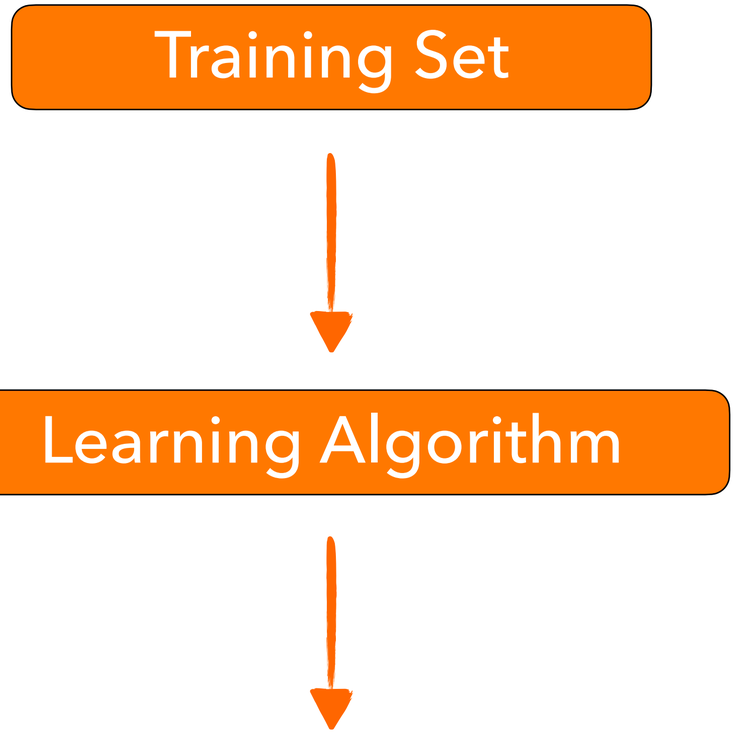
Recap: Model Evaluation


```python
# sample size of each fold
n = round(X.shape[0] / 5)

# inspect AIC and BIC for each model
for model in DICT_MODEL:
    mse = np.median(DICT_MODEL[model]["mse"] * n)
    k = len(DICT_MODEL[model]["var"])
    aic = get_AIC(mse, n, k)
    bic = get_BIC(mse, n, k)
    print("Model: ", model, DICT_MODEL[model]["var"])
    print("AIC: ", aic)
    print("BIC: ", bic)
```

```
Model:  A ['age', 'bmi', 'children', 'sex_male', 'smoker_yes', 'region_northwest', 'region_
AIC:  6183.680080968096
BIC:  6212.407976812183
Model:  B ['age', 'bmi', 'smoker_yes', 'region_southeast']
AIC:  6178.738314582675
BIC:  6193.102262504719
Model:  C ['age', 'bmi', 'smoker_yes']
AIC:  6171.194635640093
BIC:  6181.967596581626
Model:  D ['age', 'smoker_yes']
AIC:  6199.231197573322
BIC:  6206.413171534344
```

Recap: Supervised Learning

	house_size	house_price_k
0	1000	200
1	1250	250
2	1400	370
3	1500	350
4	1560	400



$$y = h(x) = \beta_0 + \beta_1 x$$

The hypothesis that maps x to y

e.g., **Linear regression** is a type of **supervised learning algorithm:**

(x, y) - one training example

feature **label**

Unsupervised Learning

	diameter	weight
0	2.718715	7.784787
1	2.101122	13.257975
2	3.294589	8.643246
3	3.187013	13.127034
4	4.116647	11.494149
..	...	...
495	2.745682	10.155286
496	2.602344	8.924697
497	2.508185	8.756578
498	2.899556	9.330609
499	4.558679	14.319089

Training Set

Learning Algorithm

Hypothesis

No label (Y) is considered

PCA is an **unsupervised** learning algorithm

To find a vector direction that maximizes the variance

[**diameter, weight**] - one training example

↓

Features

X
(Diameter, weight) → Hypothesis → **PCs**
(Principal components)

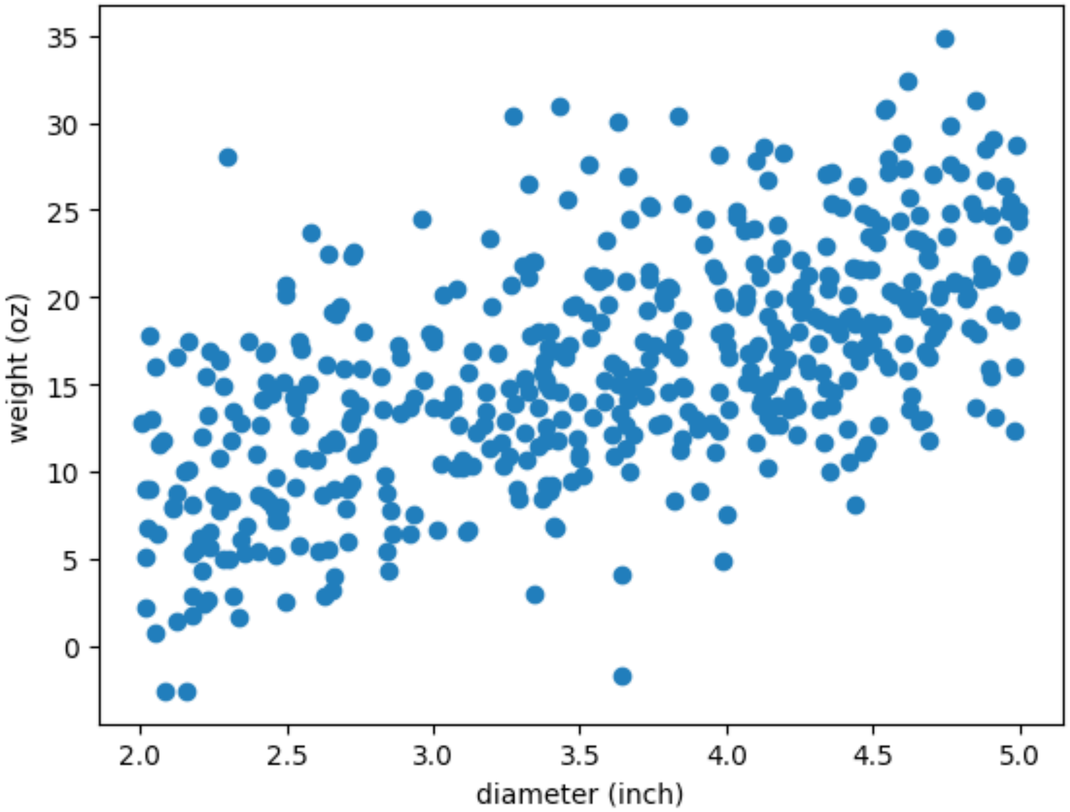
$$\Sigma = V\Lambda V^{-1} \quad X_{pc} = XV$$

The hypothesis that maps X to PCs

PCA Intuition: Orange Example

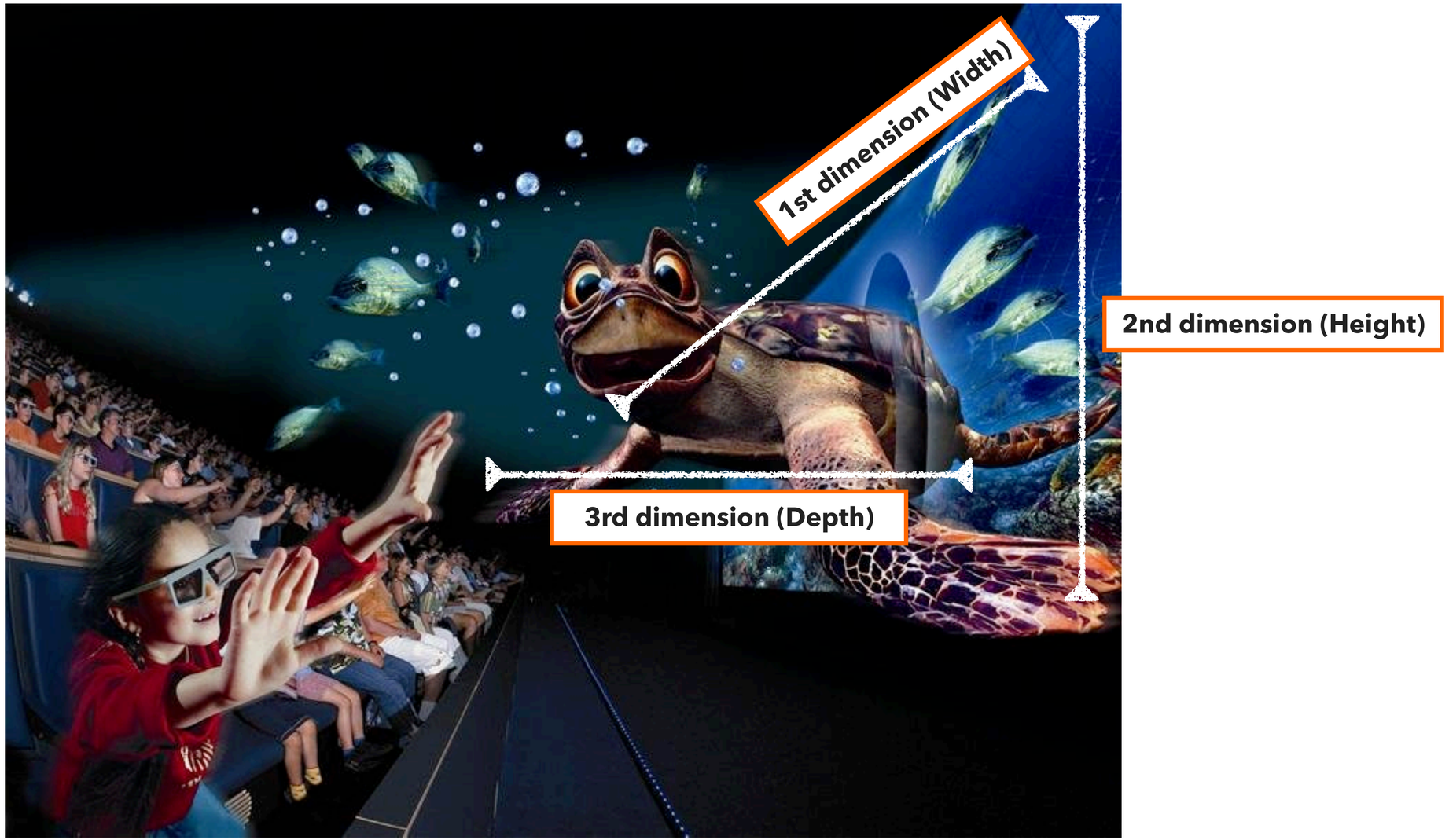
If we know **a linear relationship** between diameter and weight, should we keep both in a linear model?

	diameter	weight
0	2.718715	7.784787
1	2.101122	13.257975
2	3.294589	8.643246
3	3.187013	13.127034
4	4.116647	11.494149
..	...	...
495	2.745682	10.155286
496	2.602344	8.924697
497	2.508185	8.756578
498	2.899556	9.330609
499	4.558679	14.319089



PCA Intuition: Not all Information Is Necessary

Does wearing 3D glasses have any impact on your **comprehension** of a movie beyond enhancing the immersive experience?



https://vulcanpost.com/19508/3d-movies-give-emotional-response-2d/

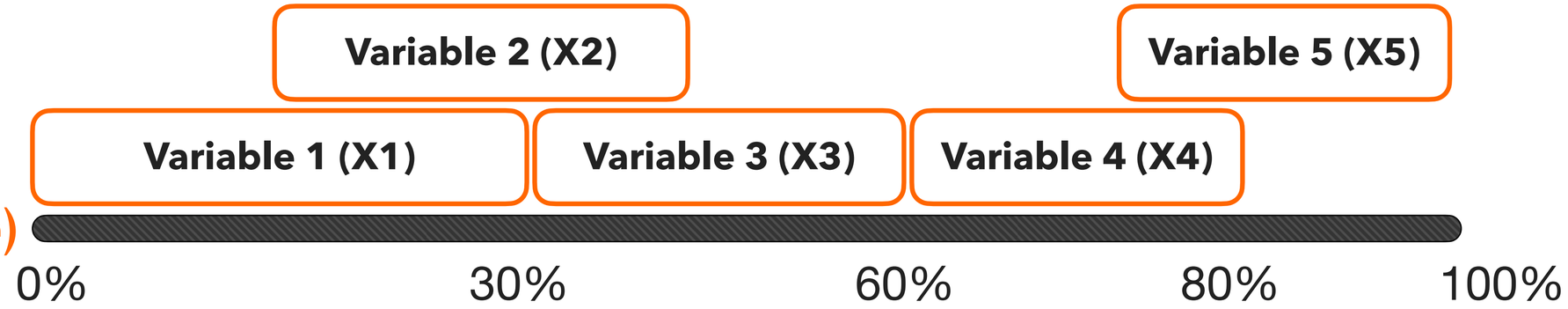
PCA Intuition: Not all Information Is Necessary

Original variables

X1 and X3 are **independent**

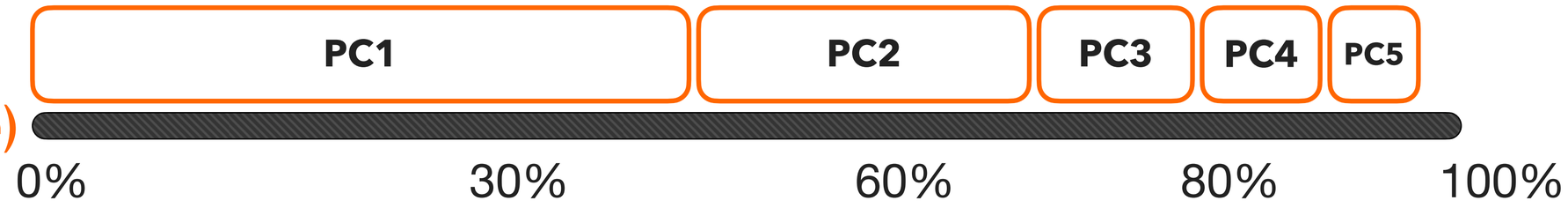
X1 and X2 are **correlated**

X4 and X5 are **correlated** (dependent)

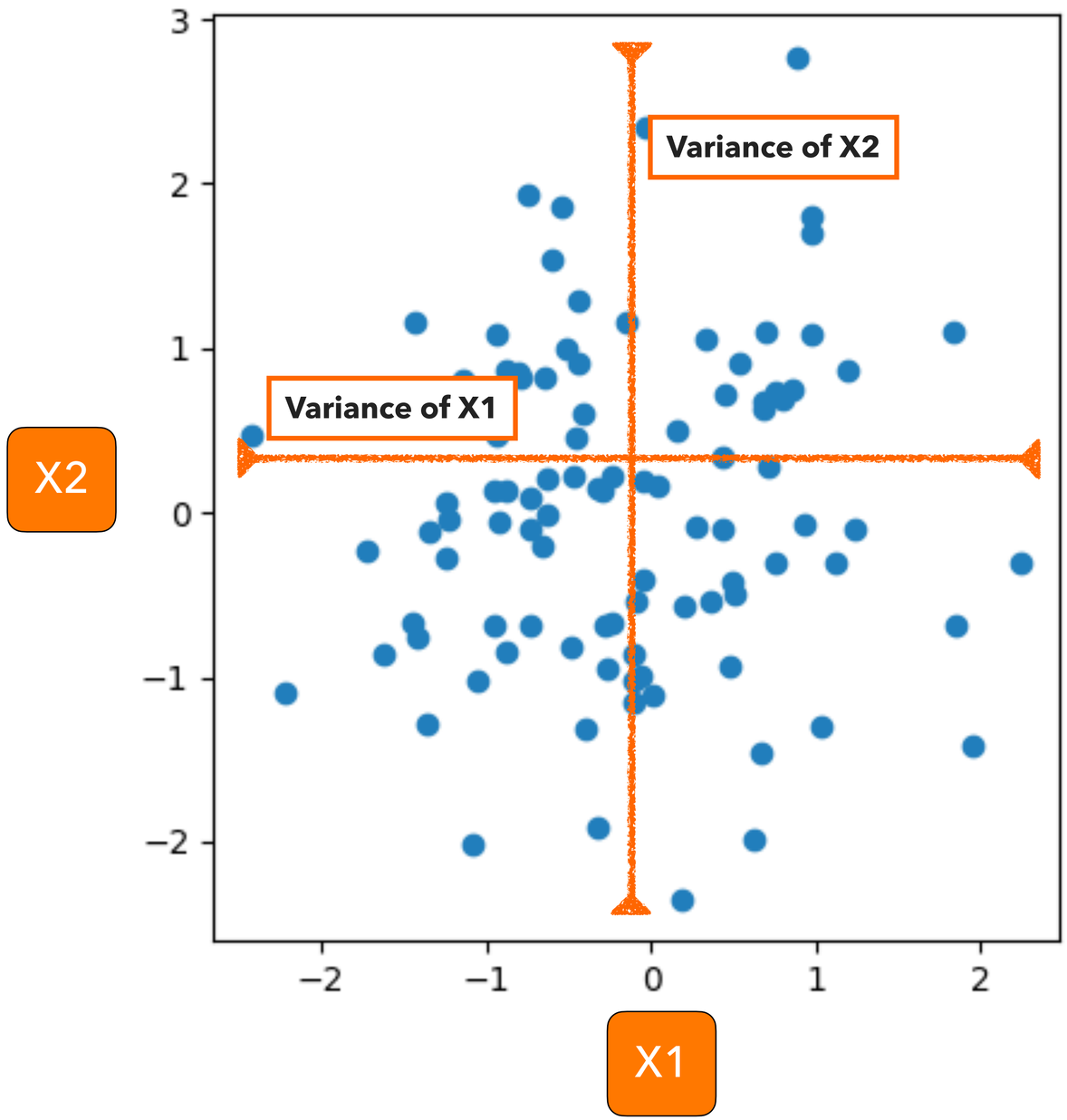


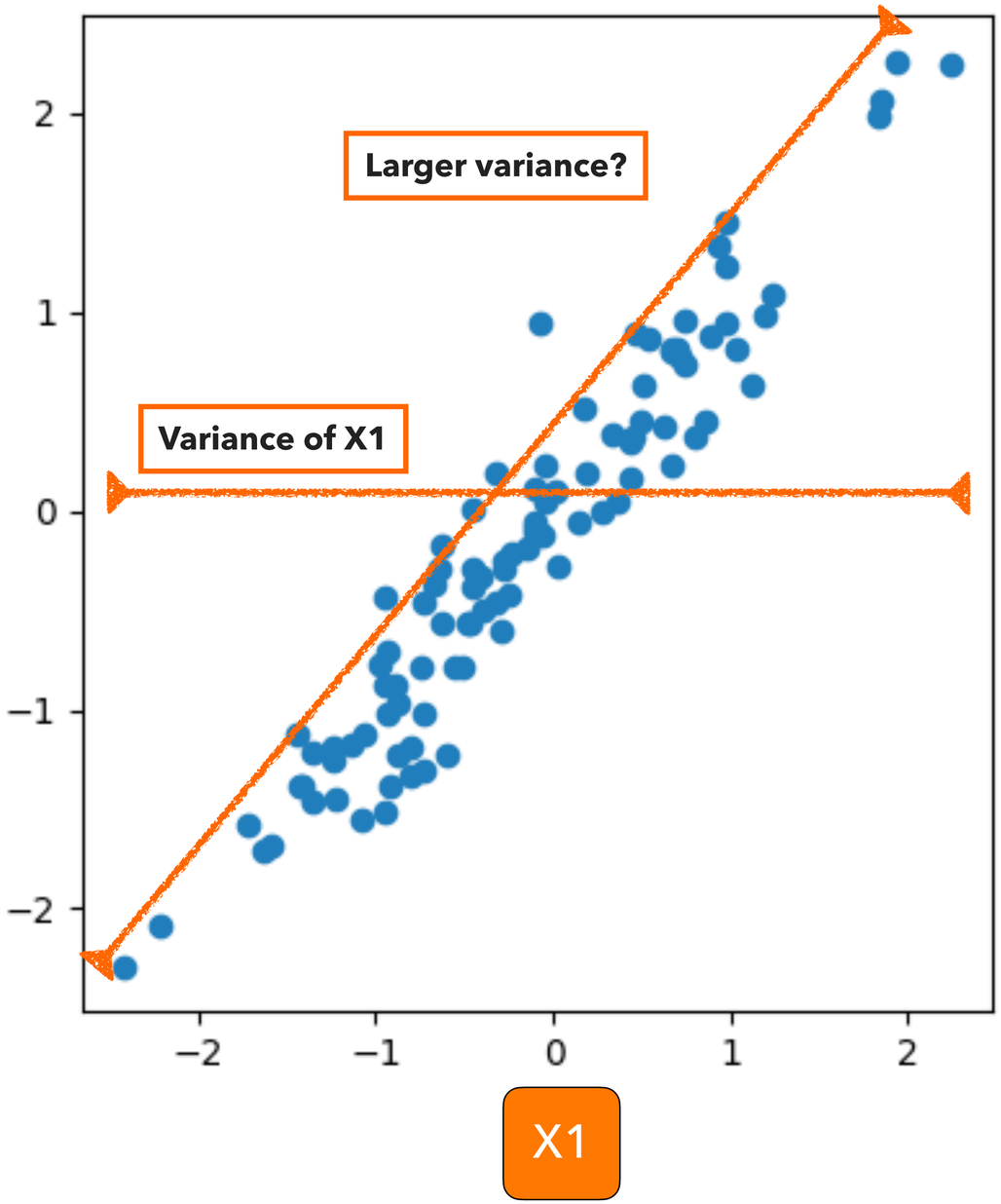
Principal Components

Every variable is **independent**!



How Can the Variance Be Interpreted in a Scatter Plot?



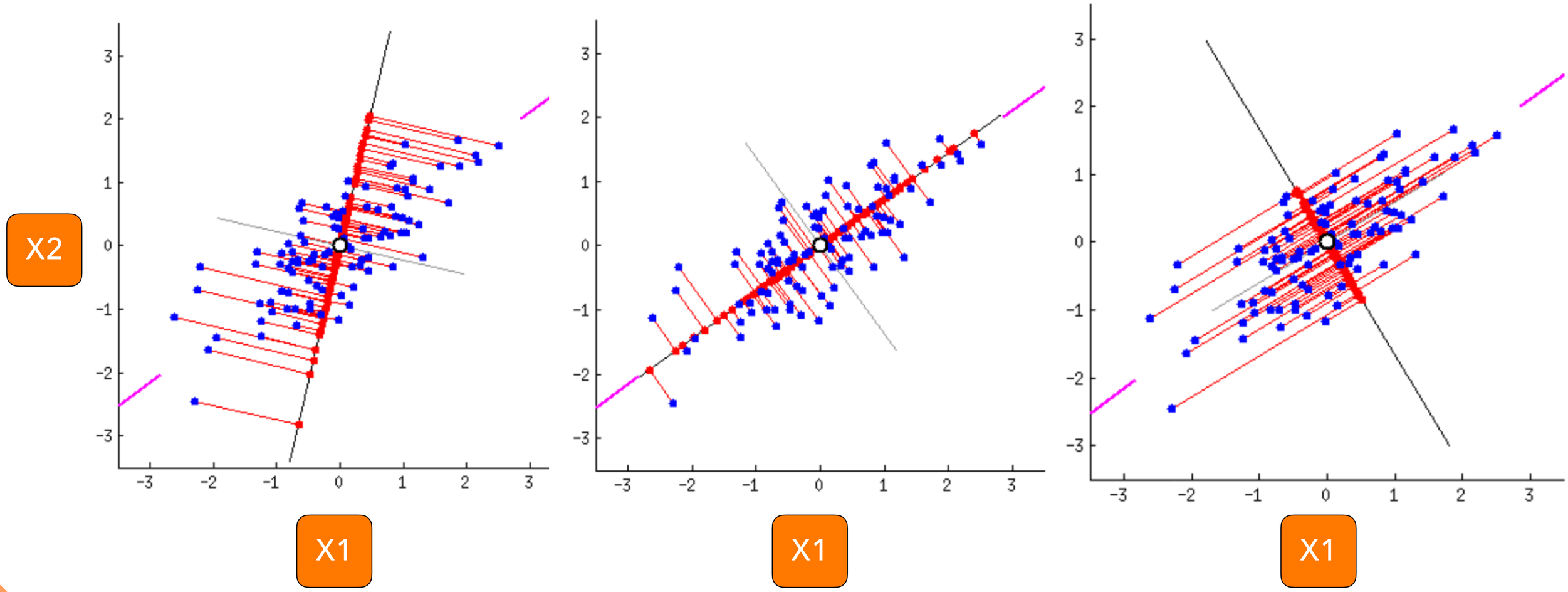


How Can the Variance Be Interpreted in a Scatter Plot?

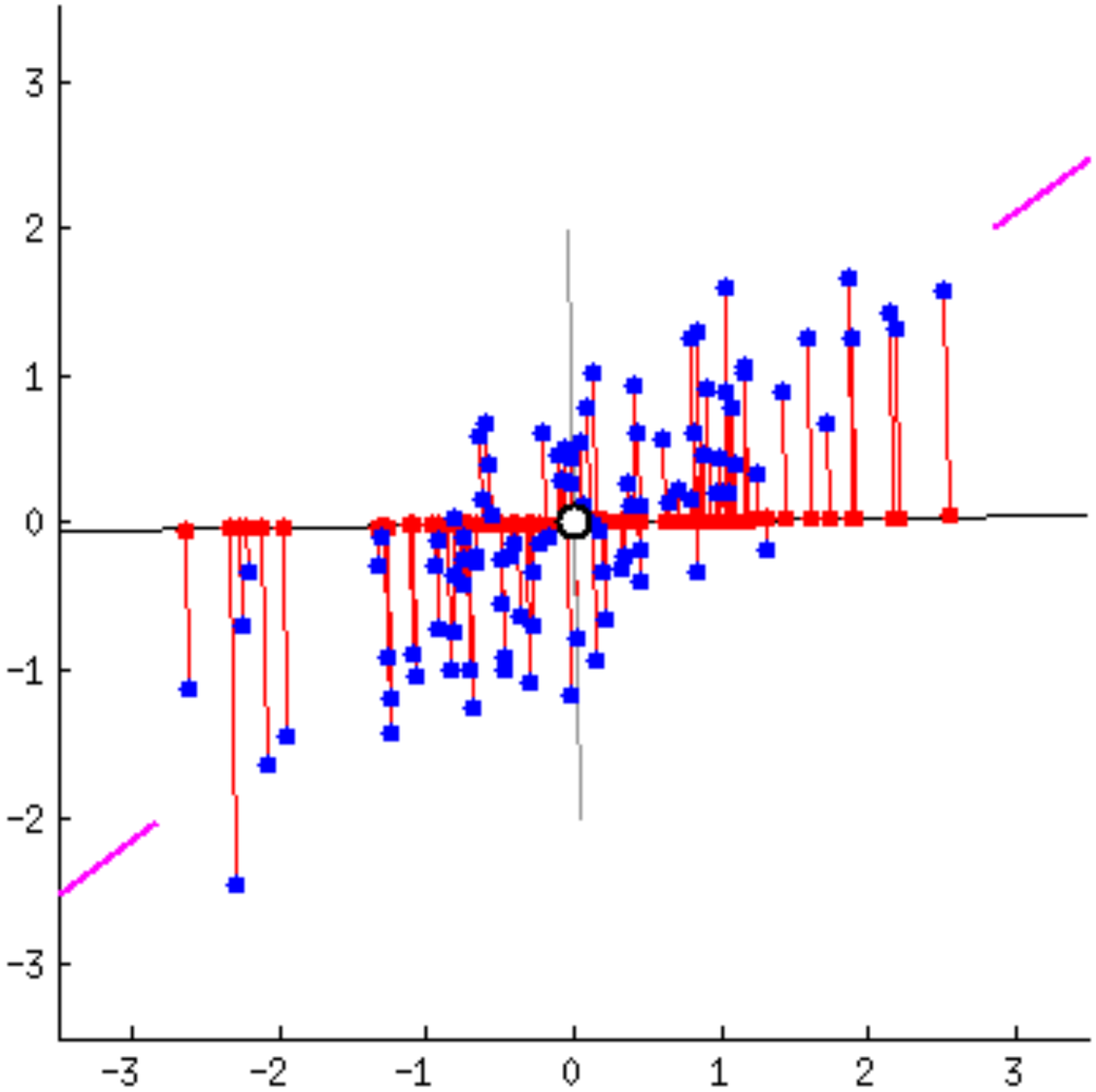
Which new axis gives a larger variance?

Original data point

Projected data point



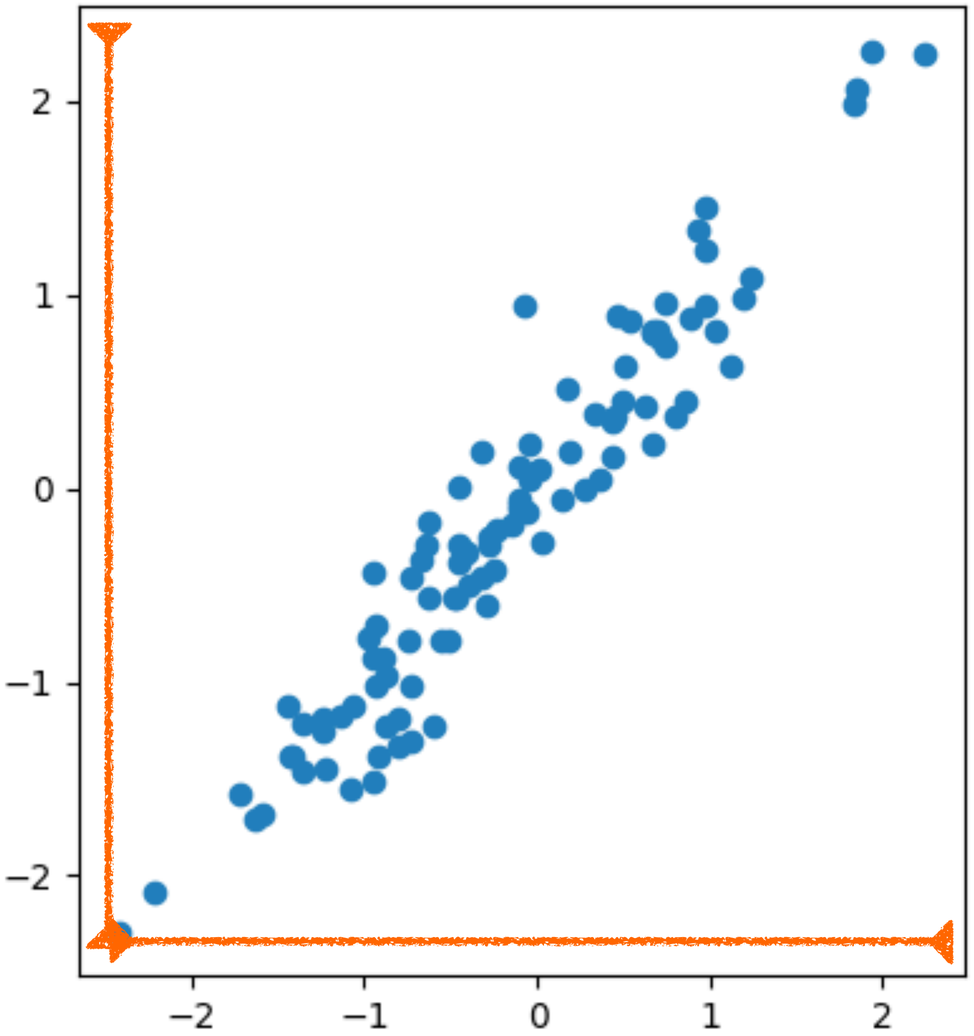
How Can the Variance Be Interpreted in a Scatter Plot?



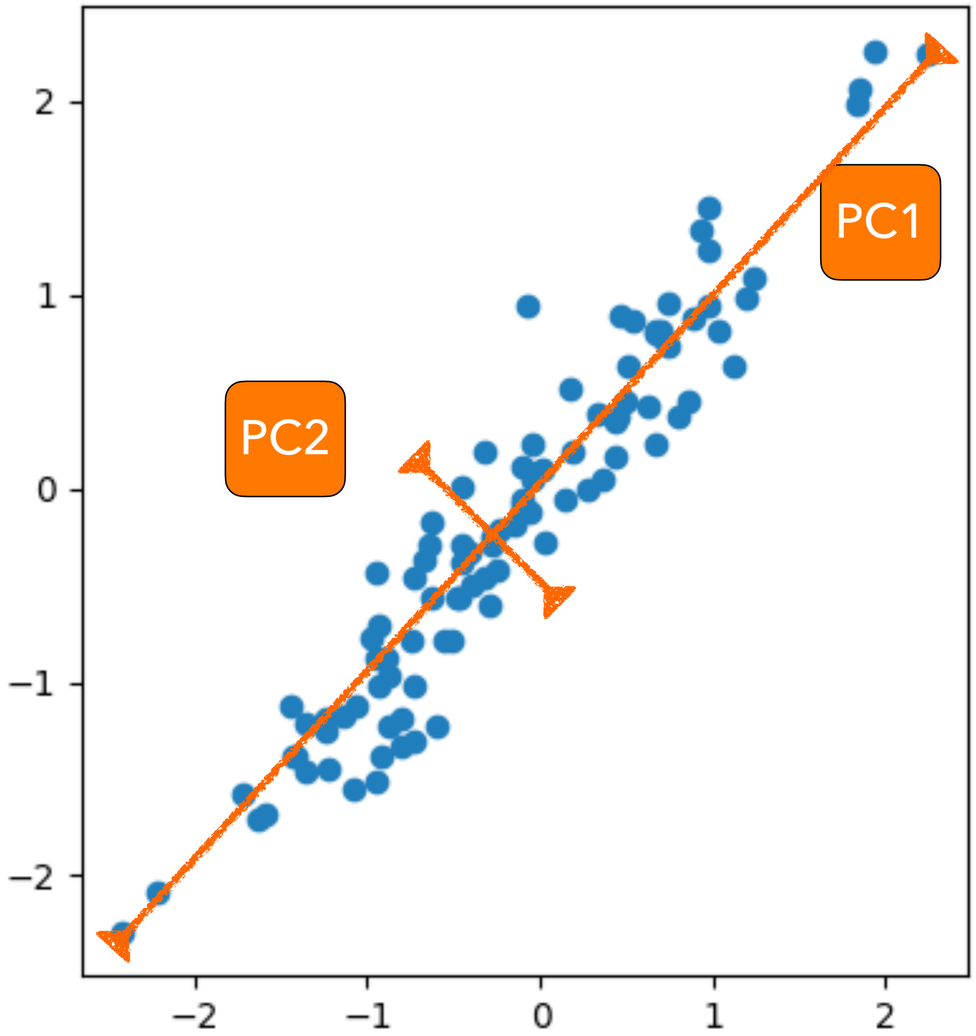
https://stats.stackexchange.com/questions/2691/making-sense-of-principal-component-analysis-eigenvectors-eigenvalues

PCA Intuition

Original space



PCA space



PCA Step 1 - Covariance Matrix

The covariance between i and j features

Always **scale** the features to unify the variance

$$\sigma_{ij} = \frac{1}{n-1}\sum_{k=1}^{n}(x_{ik} - \bar{x}_i)(x_{jk} - \bar{x}_j)$$

Linear algebra form

$$\Sigma = \frac{1}{n-1}X^T X$$

X matrix (500 x 2)

	diameter	weight
0	2.718715	7.784787
1	2.101122	13.257975
2	3.294589	8.643246
3	3.187013	13.127034
4	4.116647	11.494149
..	...	...
495	2.745682	10.155286
496	2.602344	8.924697
497	2.508185	8.756578
498	2.899556	9.330609
499	4.558679	14.319089

Covariance matrix (2 x 2)

	diameter	weight
diameter	1.00200401	0.69301446
weight	0.69301446	1.00200401

PCA Step 1 - Covariance Matrix (Python)


```python
# scale
scaler = StandardScaler()
scaler.fit(X)
X_scaled = scaler.transform(X)

# covariance matrix (numpy)
cov_np = np.cov(X_scaled.T)
# covariance matrix (manual)
n = X_scaled.shape[0]
cov_manual = np.dot(X_scaled.T, X_scaled) / (n - 1)

# compare
print("Numpy covariance matrix:")
display(cov_np)
print("Manual covariance matrix:")
display(cov_manual)
```

$$\Sigma = \frac{1}{n-1} X^T X$$

PCA Step 2 - Eigen Decomposition

Step 2: Eigen decomposition

The purpose of eigen decomposition is to find the eigenvectors and eigenvalues of the covariance matrix. The eigenvectors are the directions of the data with the largest variance. The eigenvalues are the variances of the data along the eigenvectors.

An eigen decomposition of a matrix Σ is a factorization of the form:

$$\Sigma = V \Lambda V^{-1}$$

where V is a matrix whose columns are the eigenvectors of Σ, and Λ is a diagonal matrix whose diagonal elements are the eigenvalues of Σ.

Python implementation

```
# eigenvalues and eigenvectors
eigvals, eigvecs = np.linalg.eig(cov_np)
```

$$\Sigma = V \Lambda V^{-1}$$

Dimensions 2 x 2 2 x 2 2 x 2 2 x 2

V

Rotation vectors

```
Eigenvectors:

array([[ 0.70710678, -0.70710678],
       [ 0.70710678,  0.70710678]])
```

Λ

Variance of each component

```
Eigenvalues:

array([1.66781682, 0.33619119])
```

PCA Step 3 - Transform the Data

Direction — Rotation vectors

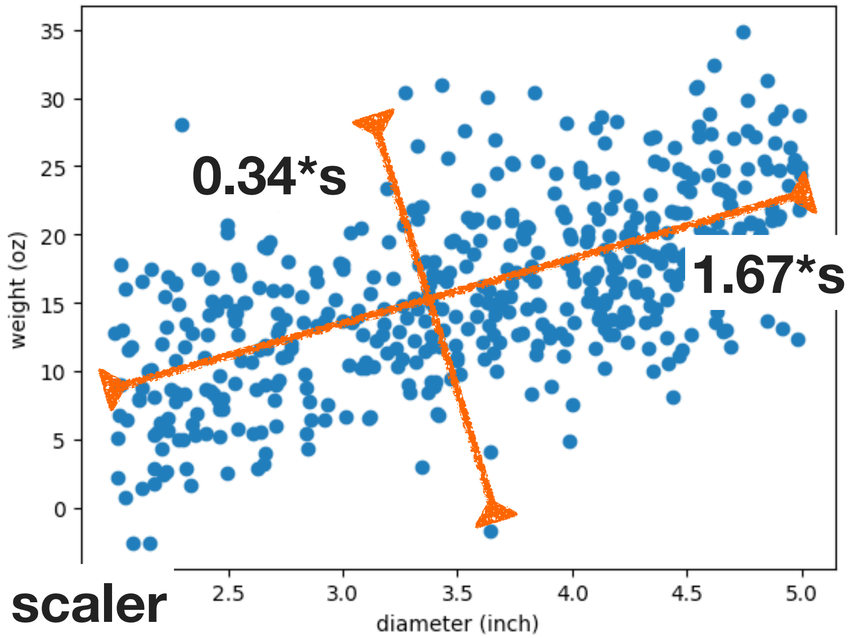
V

```
Eigenvectors:

array([[ 0.70710678, -0.70710678],
       [ 0.70710678,  0.70710678]])
```

Length — Variance of each component

Λ

```
Eigenvalues:

array([1.66781682, 0.33619119])
```



s is a scaler

Calculate the first principal component

$$X_{pc1} = XV_1$$

500 x 2 — X

diameter	weight
2.718715	7.784787
2.101122	13.257975
3.294589	8.643246
3.187013	13.127034
4.116647	11.494149
...	...
2.745682	10.155286
2.602344	8.924697
2.508185	8.756578
2.899556	9.330609
4.558679	14.319089

2 x 2 — V_1

```
0.70710678
0.70710678
```

Case Study - Orange Sweetness

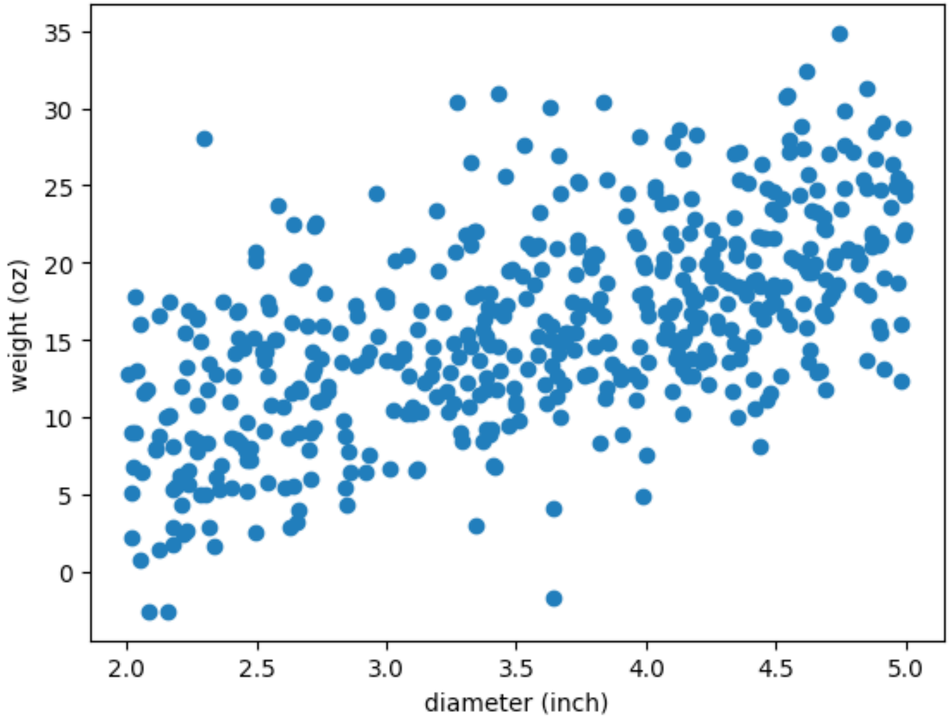
	X_r	X_w	y
	diameter	weight	sweetness
0	3.909821	11.347427	8.929665
1	3.634806	10.616417	15.855482
2	4.082725	9.325585	15.796729
3	4.255215	12.855802	19.531253
4	4.591354	9.261760	24.361757
..	...	...	...
495	4.822162	15.420119	20.163324
496	3.672326	11.915929	9.952746
497	4.545179	13.622556	30.863298
498	3.545564	15.468268	21.228367
499	2.761562	13.314906	18.061219

We created the dependency

$$X_w = 4 + 2X_r + \epsilon$$

We simulated the sweetness

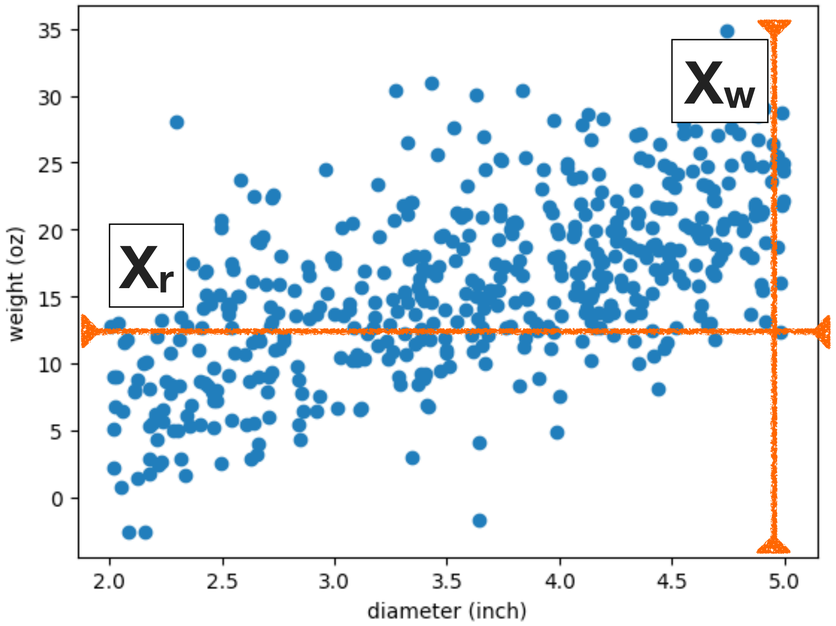
$$y = 1.6 + 5X_r - 0.3X_w + \epsilon$$



Model Selection

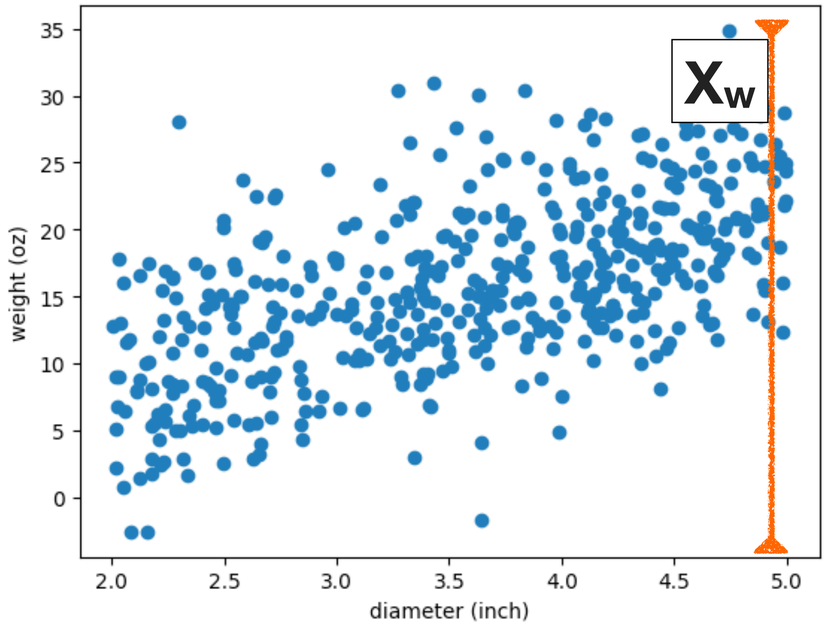
Model 1

$$y = \beta_0 + \beta_1 X_r + \beta_2 X_w + \epsilon$$



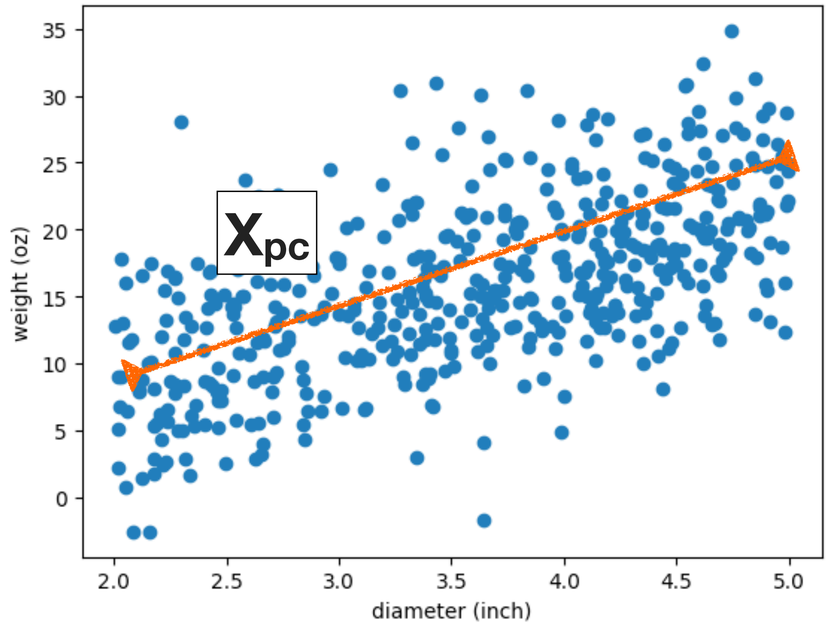
Model 2

$$y = \beta_0 + \beta_1 X_w + \epsilon$$



Model 3

$$y = \beta_0 + \beta_1 X_{pc} + \epsilon$$



Which model will perform better?

Model Selection - 5-Fold Validation

Data splitting

To examine the performance of the three models, we will split the data into training and testing sets. We will use the training set to fit the three models and use the testing set to evaluate the performance of the three models.

```
# define X and y
y = df["sweetness"]
X = df[["diameter", "weight"]]
# split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

80% X_train y_train

20% X_test y_test

	diameter	weight
386	4.106289	16.341501
293	2.396014	8.773522
203	4.691757	14.744386
173	3.770923	8.961336
103	4.250209	12.833600
...	...	...
125	3.525157	11.022015
357	2.392490	8.697894
336	3.945740	9.192041
142	3.206554	7.970732
237	4.510199	11.257973

400 rows × 2 columns

```
386    17.426316
293    11.589055
203    24.106225
173    10.425830
103    11.945412
       ...
125    12.444706
357    14.026105
336    27.104735
142    13.175794
237    22.622746
Name: sweetness, Length: 400
```

	diameter	weight
75	4.178276	7.883033
246	2.501508	11.432733
255	4.895138	11.304026
259	4.541063	13.928769
65	4.462825	12.100212
...	...	...
486	4.611015	14.046816
441	4.159833	13.464206
464	3.989959	12.199414
76	3.662335	11.712724
289	4.107439	12.590956

100 rows × 2 columns

```
75     16.853919
246     8.835451
255    16.349592
259    16.832470
65     21.089342
       ...
486    15.944648
441    12.494469
464    26.037963
76     14.803543
289    15.535487
Name: sweetness, Length: 100
```

Model Selection - 5-Fold Validation

Model 1: full model

```
model1 = LinearRegression().fit(X_train, y_train)
y_pred = model1.predict(X_test)
eval(model1, X_test, y_test)
```

```
def eval(model, X_test, y_test):
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    n = len(y_test)
    k = X_test.shape[1]
    bic = n * np.log(mse) + k * np.log(n)
    print("MSE: {:.4f}".format(mse))
    print("BIC: {:.4f}".format(bic))
```

Model 2: reduced model with only weight as a feature

```
model2 = LinearRegression().fit(X_train[["weight"]], y_train)
y_pred = model2.predict(X_test[["weight"]])
eval(model2, X_test[["weight"]], y_test)
```

Model Selection - 5-Fold Validation

Model 3: reduced model with only principal component as a feature

```
# standardize the data
scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)

# create a PCA instance
pca = PCA(n_components=1)
pca.fit(X_train_scaled)

# transform the data
pc_train = pca.transform(X_train_scaled)
pc_test = pca.transform(X_test_scaled)

# fit model
model3 = LinearRegression().fit(pc_train, y_train)
y_pred = model3.predict(pc_test)
eval(model3, pc_test, y_test)
```

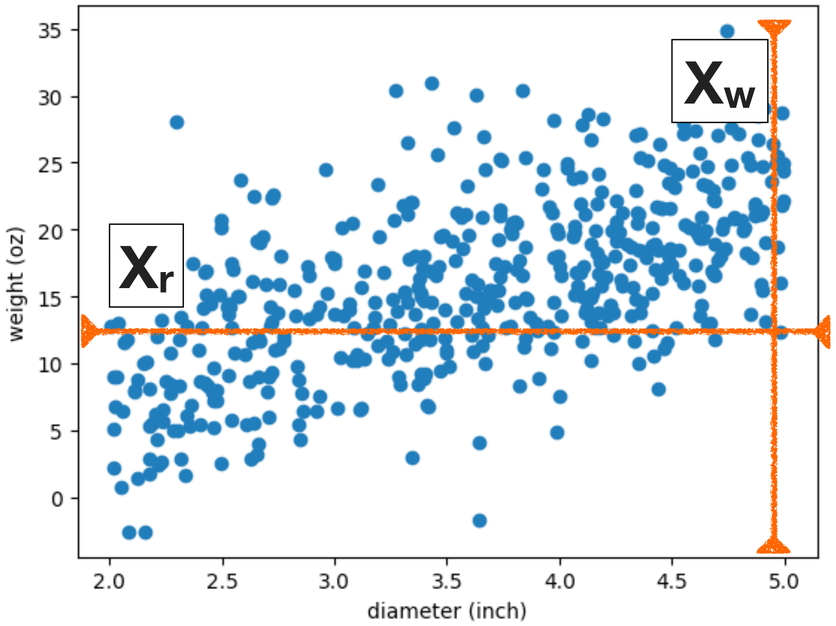
Must apply the **same** scaler on the testing data

Must apply the **same** Eigen vectors on the testing data

Model Selection - 5-Fold Validation

Model 1

$$y = \beta_0 + \beta_1 X_r + \beta_2 X_w + \epsilon$$



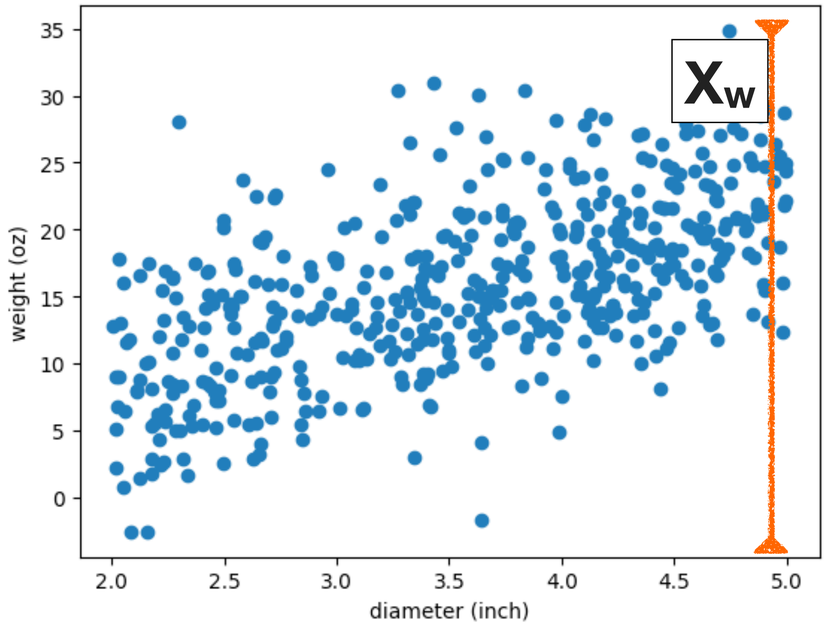
2 variables

MSE: 22.7934

BIC: 321.8573

Model 2

$$y = \beta_0 + \beta_1 X_w + \epsilon$$



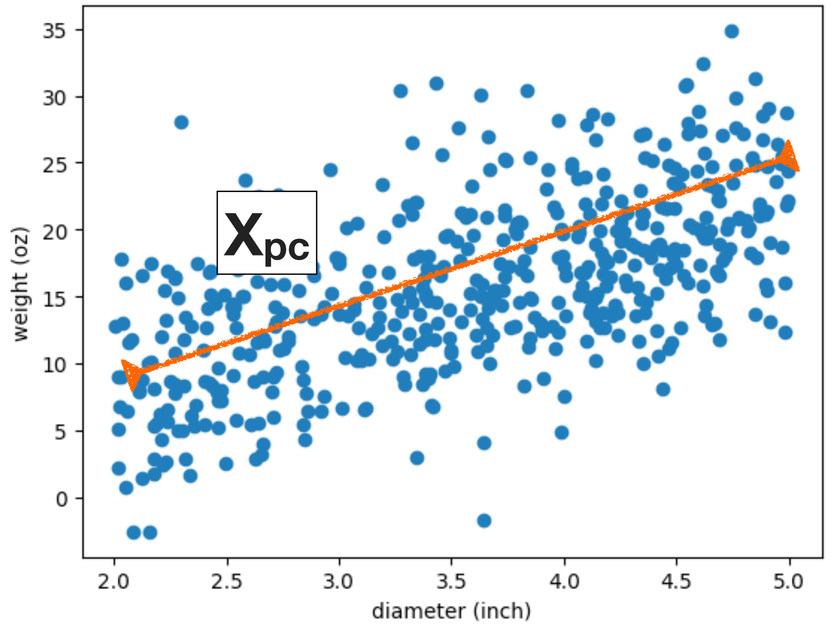
1 variable

MSE: 29.1027

BIC: 341.6882

Model 3

$$y = \beta_0 + \beta_1 X_{pc} + \epsilon$$



1 variable

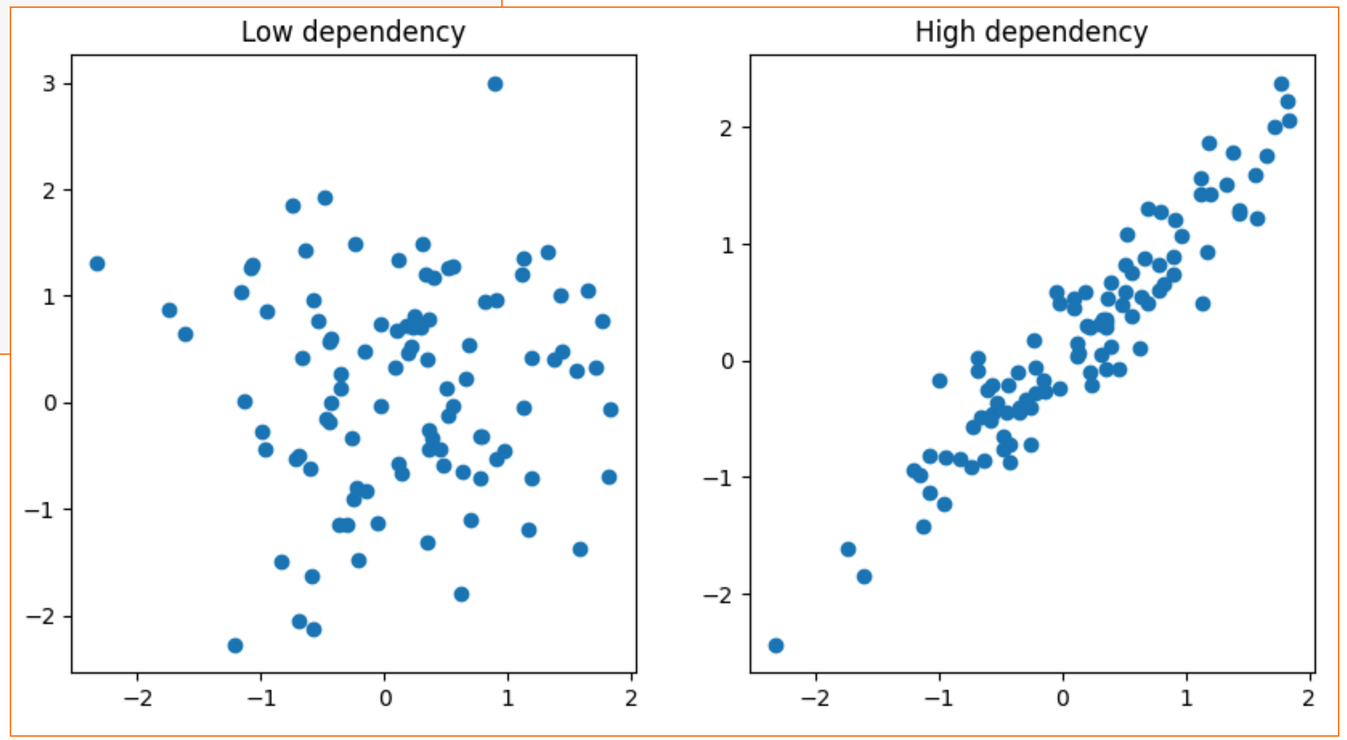
MSE: 24.5193

BIC: 324.5512

Feature Dependency

We want to discuss the effectiveness of PCA when the features are highly correlated with each other. Let's simulate two datasets with different linear dependencies between the features. The first dataset has a strong linear dependency between the features, and the second dataset has a weak linear dependency between the features.

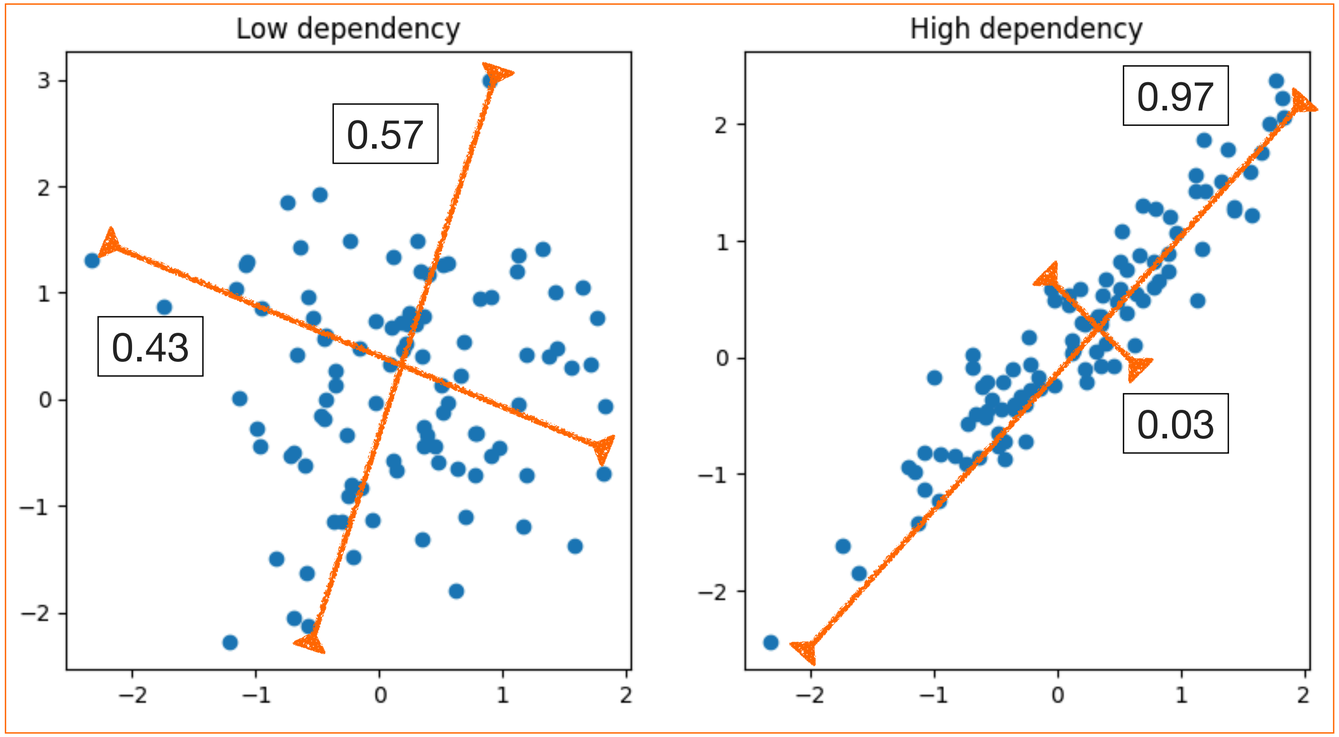
```
x = np.random.normal(0, 1, 100)
X1 = np.vstack([x, np.random.normal(0, 1, 100)]).T # low dependency
X2 = np.vstack([x, x + np.random.normal(0, .3, 100)]).T # high dependency

# plot
fig, ax = plt.subplots(1, 2, figsize=(10, 5))
ax[0].scatter(X1[:, 0], X1[:, 1])
ax[0].set_title("Low dependency")
ax[1].scatter(X2[:, 0], X2[:, 1])
ax[1].set_title("High dependency")
plt.show()
```



Feature Dependency


```
# X1: low dependency
pca = PCA(n_components=2)
pca.fit(X1)
print("X1: Variance explained by each PC:")
display(pca.explained_variance_ratio_)

# X2: high dependency
pca = PCA(n_components=2)
pca.fit(X2)
print("X2: Variance explained by each PC:")
display(pca.explained_variance_ratio_)
```

```
X1: Variance explained by each PC:

array([0.57393067, 0.42606933])

X2: Variance explained by each PC:

array([0.97227375, 0.02772625])
```



PCA works better on datasets that exhibit **strong dependencies** or correlations between their variables (features)

Visualization of the Transformed Data