



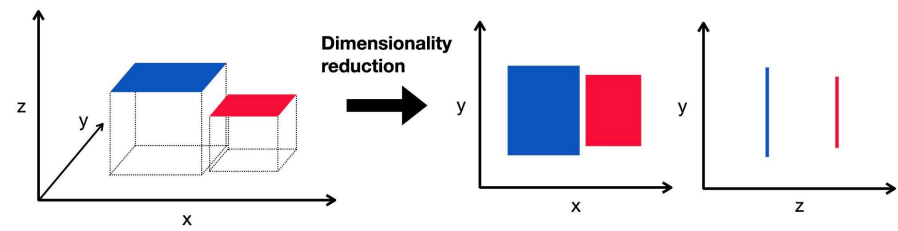
Dimensionality Reduction



THE KNOWLEDGE HOUSE

Agenda - Schedule

1. Linear Algebra Review
2. Eigenvalues & Eigenvectors
3. Principal Component Analysis
4. Other Techniques (ICA, t-SNE)
5. PCA Lab



It's all perspective



Agenda - Goals

- Review the fundamentals of linear algebra
- Understand the necessity for dimension decomposition
- Understand the intuition of singular value decomposition (SVD)
- Apply SVD to implement PCA
- Implement dimensionality reduction

Review



Review Question

Why do we resample our dataset via the **bootstrap** when implementing bagged trees?

- A) Because resampling **reduces variance**
- B) Because resampling **increases variance**
- C) Because decision trees need to learn to pull themselves up by their bootstraps if they want to get anywhere in this world.



Review Question

Random forests provide an **improvement** over bagged trees by selecting a subset of size **\sqrt{P}** predictors, where **P** is our total number of predictors. Which effect does this have?

- A) De-correlates our variance
- B) Re-correlates our variance
- C) De-correlates our trees
- D) Re-correlates our trees



Review Question

How do we measure **feature importance** in random forest classifiers?

- A) Purity **lost** for each conditional node that contains a feature
- B) Purity **gained** for each conditional node that contains a feature
- C) MSE **lost** for each conditional node that contains a feature
- D) MSE **gained** for each conditional node that contains a feature

PCA Introduction

The Curse of Dimensionality

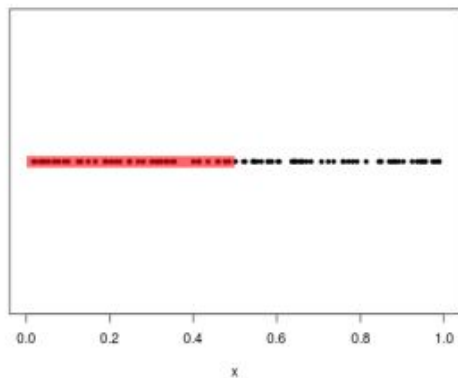
THE CURSE (...of dimensionality)

As you increase your dimensions (the # of predictors), the distance between your points become uniform!

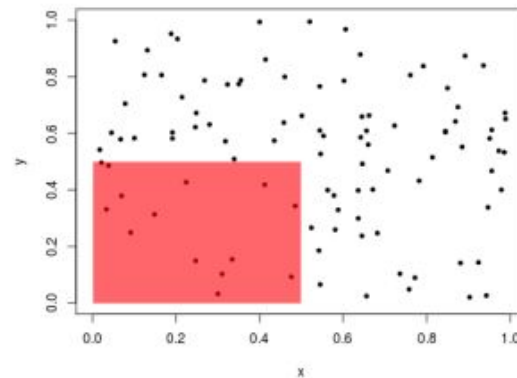
Basically it becomes harder to distinguish which class of data a new test sample belongs to as you add more and more dimensions to your dataset.



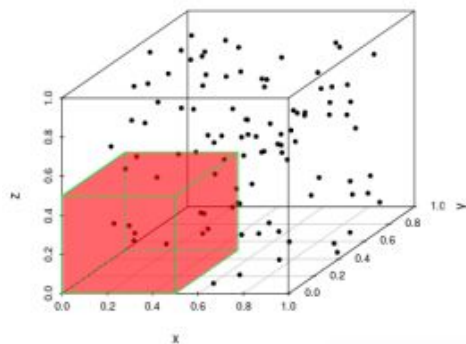
1-D: 42% of data captured.



2-D: 14% of data captured.

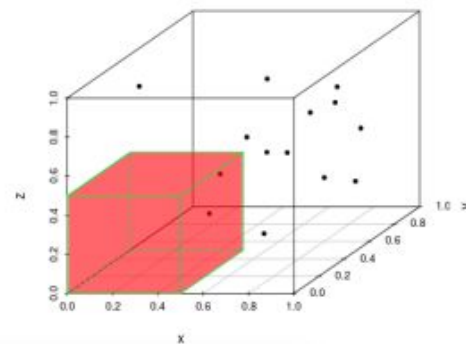


3-D: 7% of data captured.

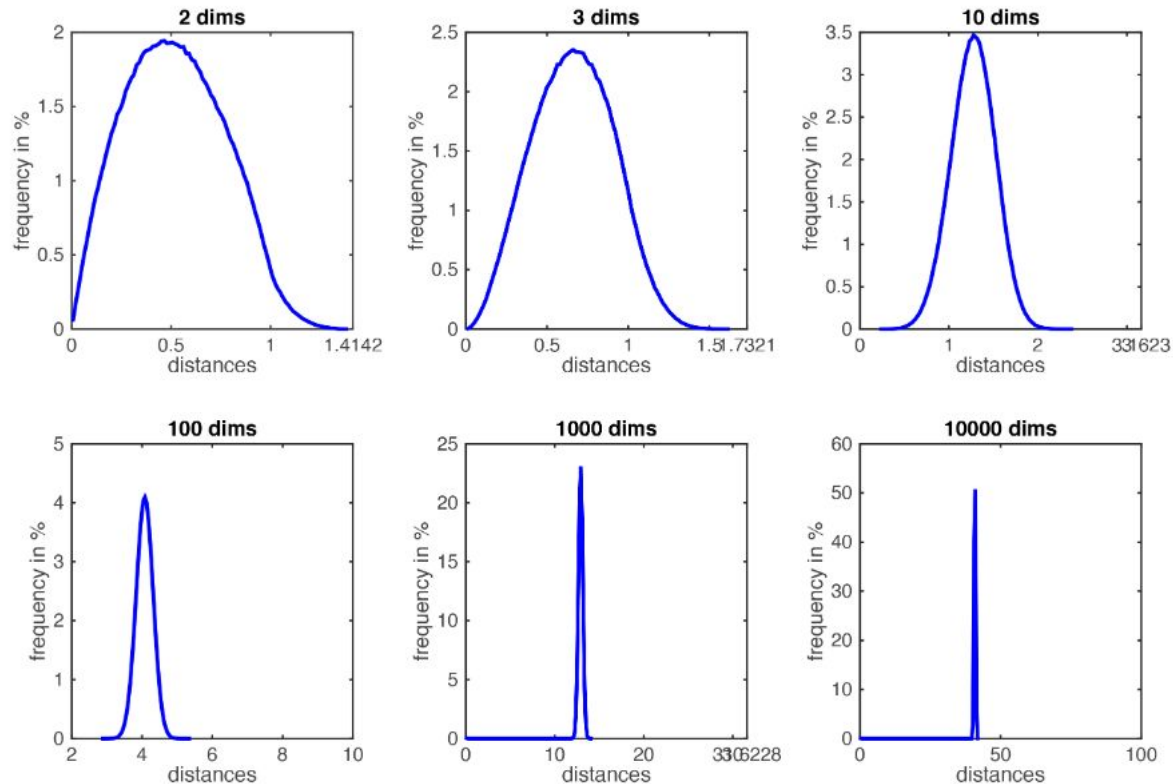


4-D: 3% of data captured.

$t = 0$



What do you notice about the **sparsity** of our data as we increase dimensions?



Notice how the frequency of distances converges on a specific value as we increase dimensions:
https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote02_kNN.html



The Curse of Dimensionality

This becomes more of an issue as we begin to deal with more advanced machine learning algorithms that take in highly rich data.

The most “cutting-edge” algorithms are incredible data consumers:

- **Number Classification Model:** 64 predictors
- **Computer Vision Models:** 1000 predictors
- **GPT-4:** *any guesses?*



The Curse of Dimensionality

This becomes more of an issue as we begin to deal with more advanced machine learning algorithms that take in highly rich data.

The most “cutting-edge” algorithms are incredible data consumers:

- **Number Classification Model:** 64 predictors
- **Computer Vision Models:** 1000 predictors
- **GPT-4:** Untold number of predictors to train *1.8 TRILLION parameters*

By nature, these algorithms are data-greedy. We cannot expect to feed them small amounts of data and expect them to work sufficiently.



PCA Introduction



However we can ward off this curse using an observation on highly dimensional data.

Does anyone recall what this **blessing** was?



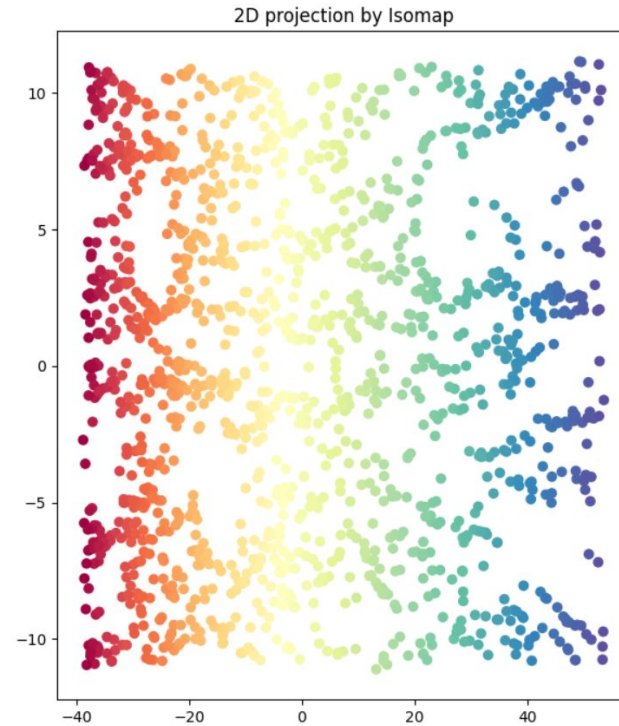
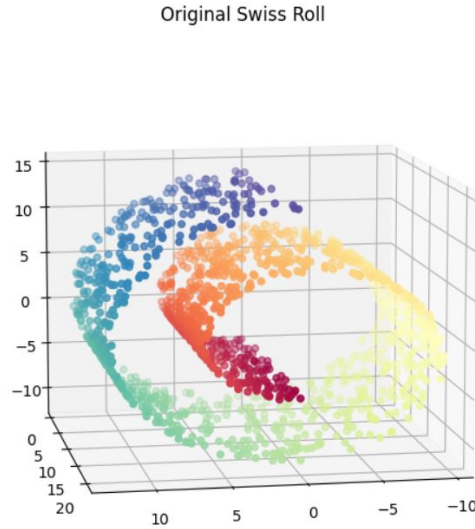
The Curse of Dimensionality

Blessing of non-uniformity: highly dimensional datasets are concentrated near lower-dimension manifolds.

Most of our relevant data does not exist in higher-dimensions. Instead, we could map our data to a lower dimensional sheet that still **retains most if not all** of our dataset characteristics.

Today, we will learn more about the utilization of this fact via **Principal Component Analysis**.

All this empty
space is “noise.”



Sure this data is plotted in 3-dimensions. However we could easily unfold this into 2 dimensions (i.e. a lower dimension manifold).



PCA Summary

The **Principal Component Analysis (PCA)** entails the following steps:

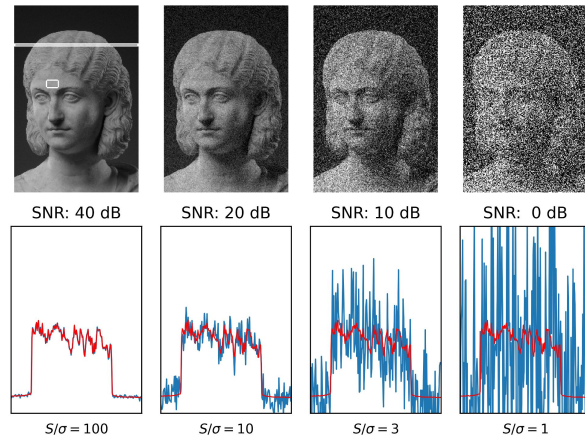
1. **Standardize** the data via the mean
2. Calculate the **covariance** matrix
3. Calculate the **eigenvalues and eigenvectors** of the covariance matrix
4. Utilize eigenvectors to calculate your new **principal components**.

PCA Summary

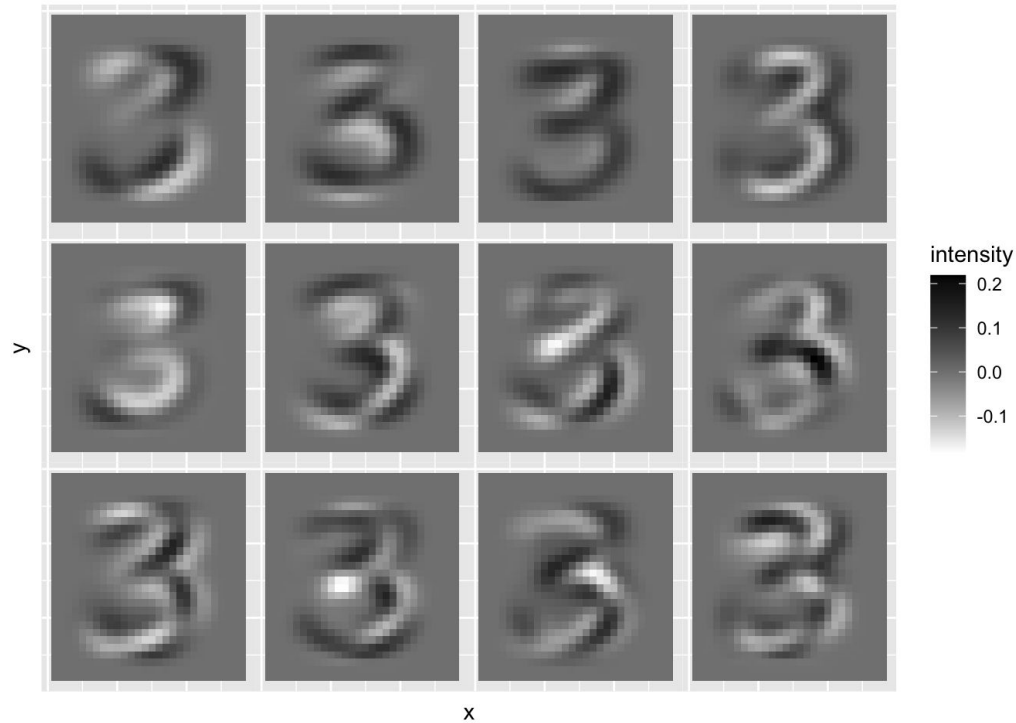
Think of your **principal components** as the essence of your multidimensional dataset.

PCA asks: If we could reduce **all the noise of a dataset** and boil it down to its **most important features**, what would be revealed?

Almost all real-world datasets are extremely noisy, we need a method to clean up this noise somehow.



Namely we highlight the curves and dips of this number.



Principal component analysis on the hand-written digit “3”: notice how we highlight the **consistent** shape of a 3 regardless of handwriting style.



Principal component analysis on human faces: a bit spooky, but notice that the same features are highlighted regardless of face-shape, race, age: *2 eyes, a nose, and a mouth.*



PCA Summary

Let's first review linear algebra before breaking this algorithm down.

Scalar

24

Vector

$\begin{bmatrix} 2 & -8 & 7 \end{bmatrix}$

row

or
column $\begin{bmatrix} 2 \\ -8 \\ 7 \end{bmatrix}$

Matrix

$\begin{bmatrix} 6 & 4 & 24 \\ 1 & -9 & 8 \end{bmatrix}$

row(s) × column(s)

Geometric Interpretation of Linear Algebra

*Generally, we say *mathematics* in the US. But in the UK, they abbreviate this as *maths*.

Scalar

24

Vector

$\begin{bmatrix} 2 & -8 & 7 \end{bmatrix}$

row

or
column $\begin{bmatrix} 2 \\ -8 \\ 7 \end{bmatrix}$

Matrix

$\begin{bmatrix} 6 & 4 & 24 \\ 1 & -9 & 8 \end{bmatrix}$

row(s) × column(s)

Before we go any further, let's review linear algebra concepts as dimensionality reduction is **completely reliant** on this field of **mathematics***.

Linear Algebra

Let's try to reason as to **why** we often use linear algebra in data science.

What is the one central object that always re-emerges in applied data science

More specifically, what is the **one structure** we always rely on to store our data

$$\begin{matrix} & \text{Columns} \\ & \begin{matrix} 1 & 2 & \dots & n \end{matrix} \\ \begin{matrix} \text{Rows} \\ \left\{ \begin{matrix} 1 \\ 2 \\ 3 \\ \vdots \\ m \end{matrix} \right\} \end{matrix} & \left[\begin{array}{cccc} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ a_{31} & a_{32} & \dots & a_{3n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{array} \right] = A_{m \times n} \end{matrix}$$

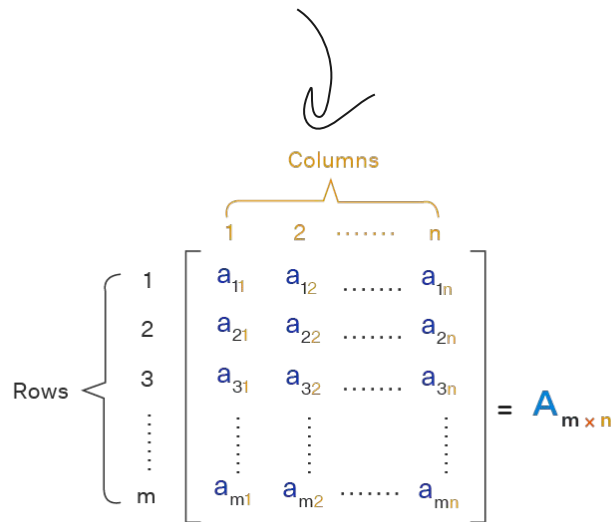
A dataset!

A dataset!

Taxes and Home Prices

<http://lib.stat.cmu.edu/DASL/Stories/hometax.html>

House	Sale price (100\$)	Size (sqft)	Age (years)
Avalon	2050	2650	13
Cross Winds	2080	2600	*
The White House	2150	2554	6
The Rectory	2150	2921	3
Larchwood	1999	2580	4
Orchard House	1900	2580	4
Shangri-La	1800	2774	2
The Stables	1560	1920	1
Cobweb Cottage	1450	2150	*
Nairn House	1449	1710	1



PointsPerGame	AssistsPerGame
18.72	3.38
22.88	7.97
20.07	5.11
25.0	20.0
18.62	6.12
15.59	1.17

Instead of staying put in the **formal language of linear algebra**, let's instead view a toy dataset of **basketball players and their stats** and see **how we can apply linear algebra concepts to it to extract meaningful insight.**

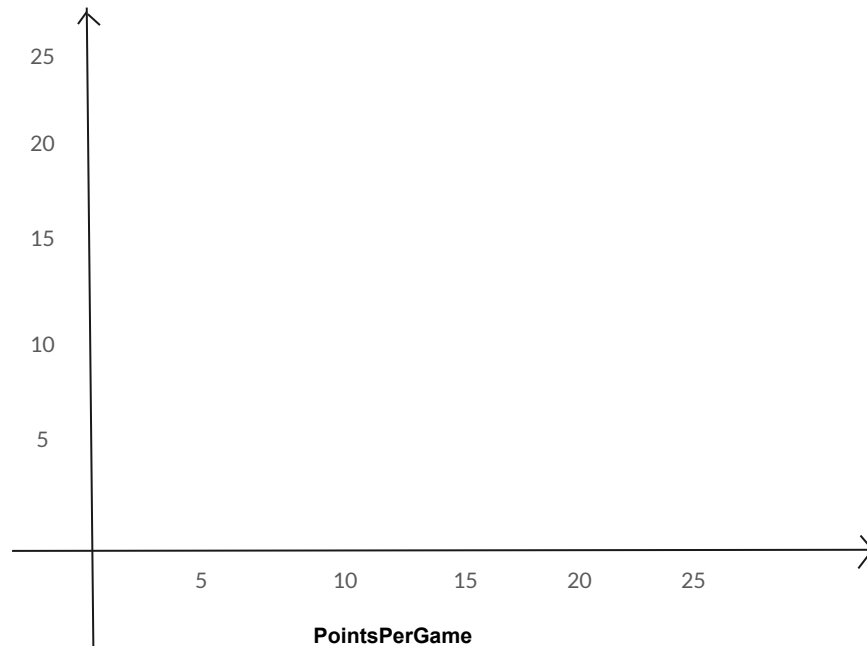
PointsPerGame	AssistsPerGame
18.72	3.38
22.88	7.97
20.07	5.11
25.0	20.0
18.62	6.12
15.59	1.17



And for the sake of story-telling, we are a Jonah Hill-like statistician from the movie **MoneyBall**, and we are trying to teach Brad Pitt how to interpret his team

PointsPerGame	AssistsPerGame
18.72	3.38
22.88	7.97
20.07	5.11
25.0	20.0
18.62	6.12
15.59	1.17

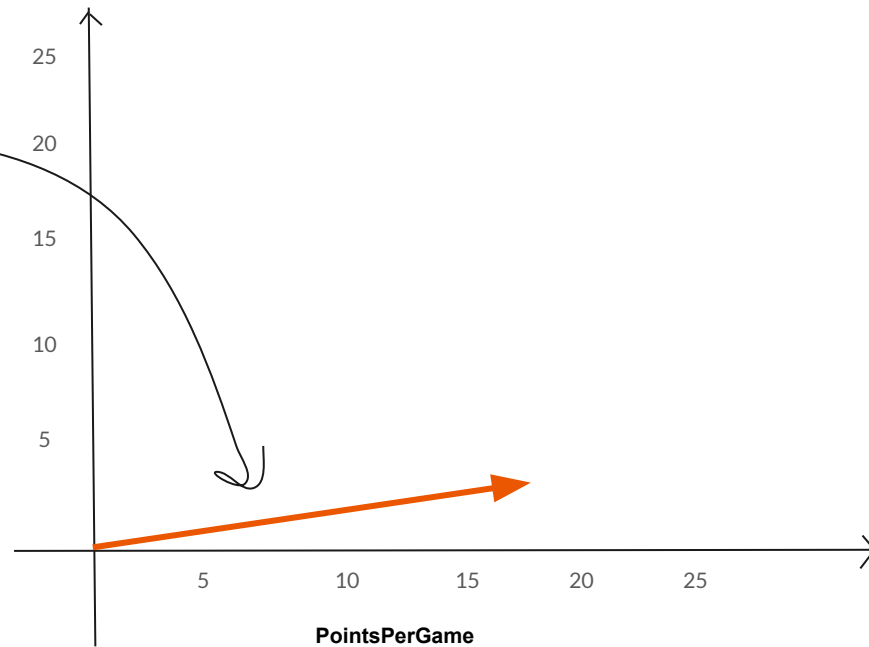
AssistsPerGame



Instead of seeing each sample as a static number, we will view each sample as a **row-vector (1 row and multiple columns)**

Furthermore, we will visualize each vector as **having magnitude and direction on a cartesian plot.** To keep things simple, we interpret vectors according to their “X & Y” coordinates.

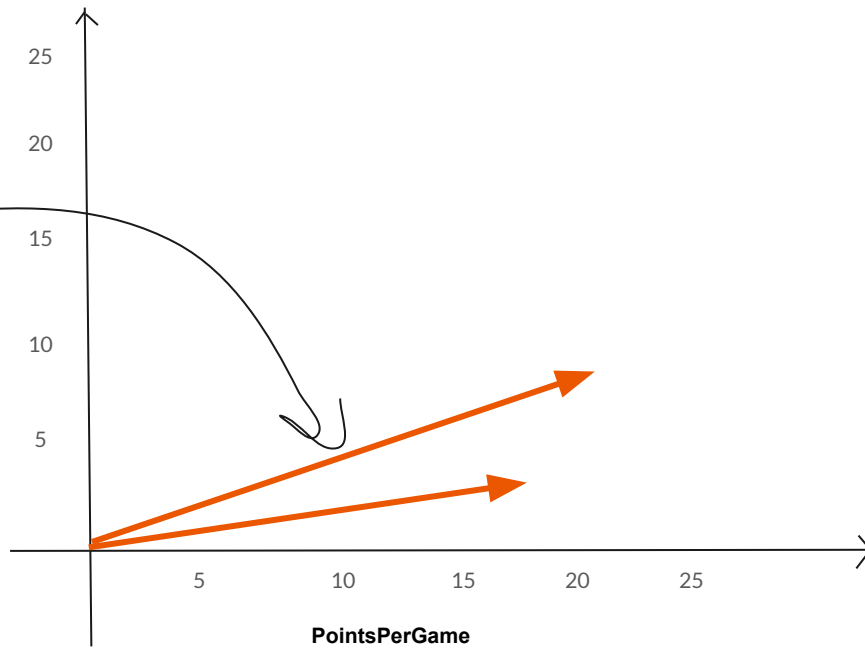
PointsPerGame	AssistsPerGame
18.72	3.38
22.88	7.97
20.07	5.11
25.0	20.0
18.62	6.12
15.59	1.17



First we consider the row-vector $[18.72, 3.38]$. For simplicity's sake we start at the origin $[0,0]$ of our coordinate system

PointsPerGame	AssistsPerGame
18.72	3.38
22.88	7.97
20.07	5.11
25.0	20.0
18.62	6.12
15.59	1.17

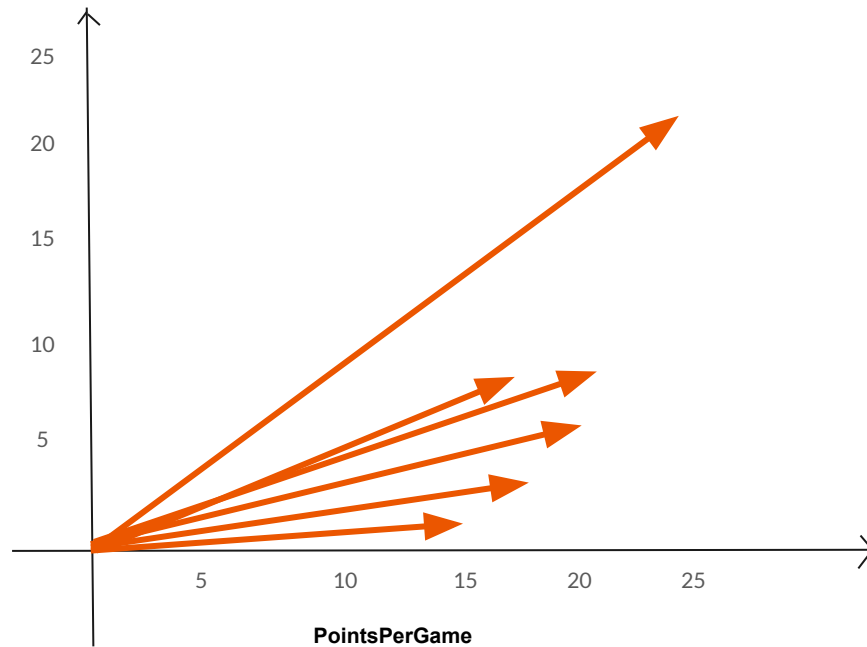
AssistsPerGame



Next [22.88, 7.97]

PointsPerGame	AssistsPerGame
18.72	3.38
22.88	7.97
20.07	5.11
25.0	20.0
18.62	6.12
15.59	1.17

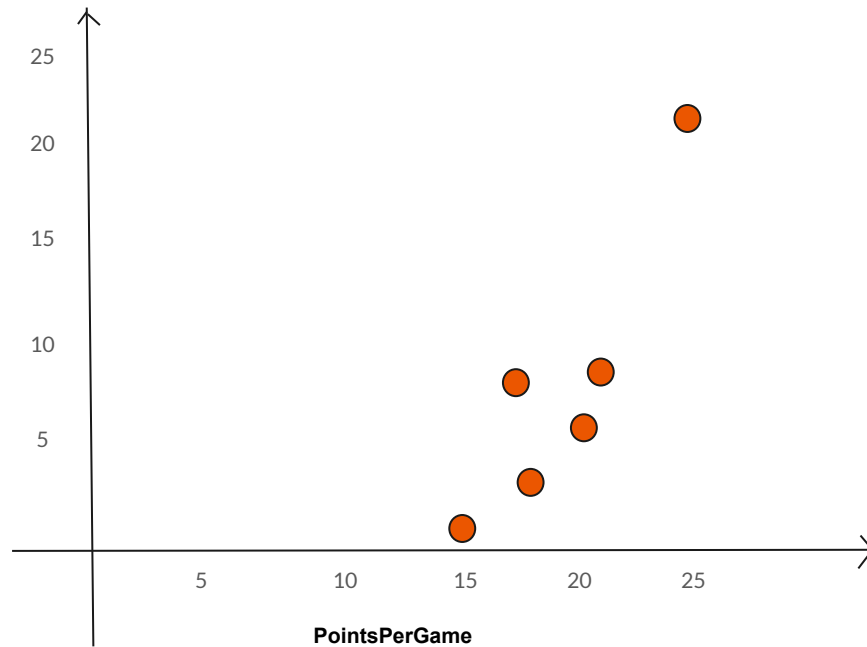
AssistsPerGame



You get the idea, as well as the rest...

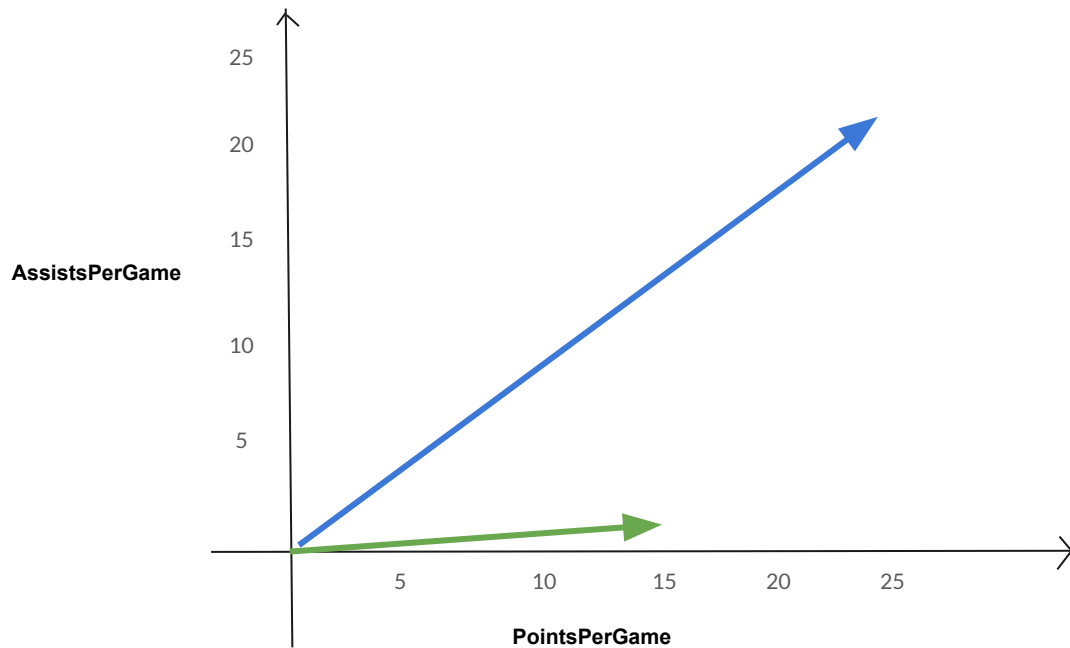
PointsPerGame	AssistsPerGame
18.72	3.38
22.88	7.97
20.07	5.11
25.0	20.0
18.62	6.12
15.59	1.17

AssistsPerGame



This is no different from plotting these players on a scatter-plot! However we gain a few insights by treating them as directional magnitudes (which we will see later)

PointsPerGame	AssistsPerGame
18.72	3.38
22.88	7.97
20.07	5.11
25.0	20.0
18.62	6.12
15.59	1.17



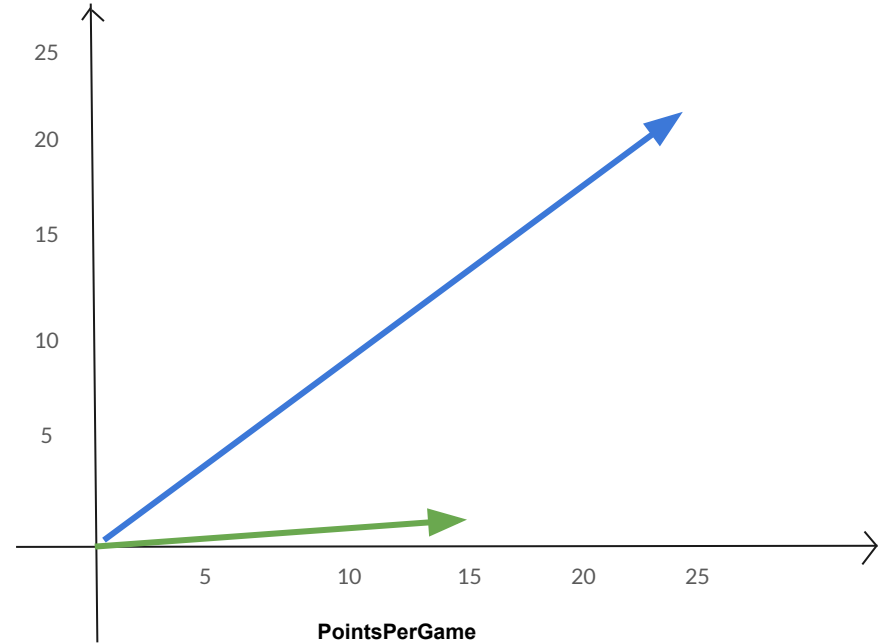
Let's consider only two of these vectors. We will look at our **high-performing player**, as well as our **lower performing player**.

Player1 = [25.00, 20.00]

Player2 = [15.59, 1.17]

Addition

$$\begin{aligned}\text{Player1} + \text{Player2} &= [25.00, 20.00] + [15.59, 1.17] \\ &= [25.00 + 15.59, 20.00 + 1.17] \\ &= [\dots, \dots]\end{aligned}$$



Basic vector arithmetic is simple. If you can match your rows and columns together, you will be able to apply operations such as addition and subtraction to individual values of the vector.

Addition results in a new “larger vector” (in this case)

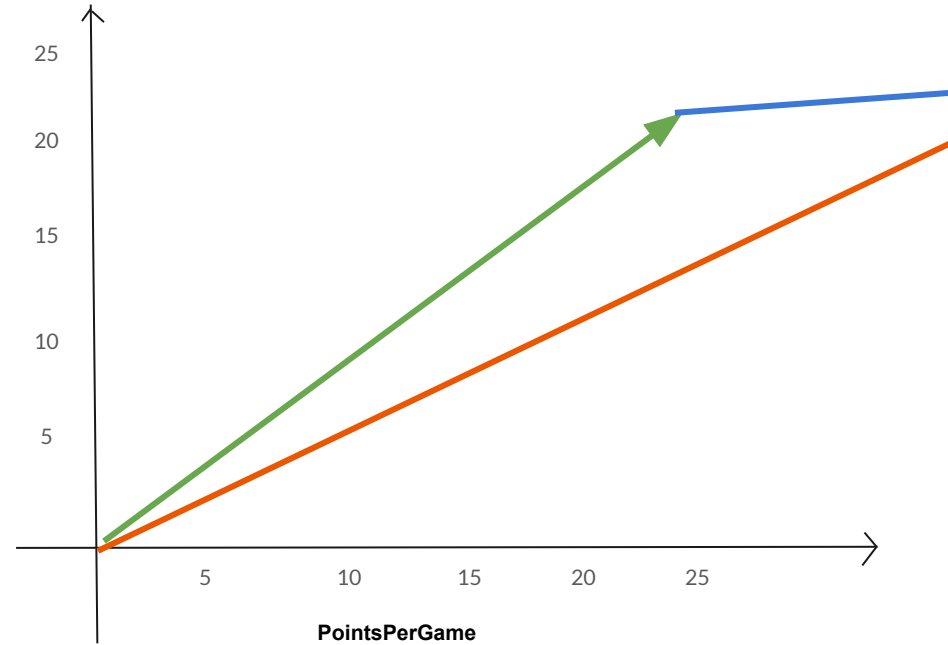
Player1 = [25.00, 20.00]

Player2 = [15.59, 1.17]

Addition

Player1 + **Player2** = [25.00, 20.00] + [15.59, 1.17]
= [25.00 + 15.59, 20.00 + 1.17]
= [40.59, 21.17]

AssistsPerGame



Basic vector arithmetic is simple. If you can match your rows and columns together, you will be able to apply operations such as addition and subtraction to individual values of the vector.

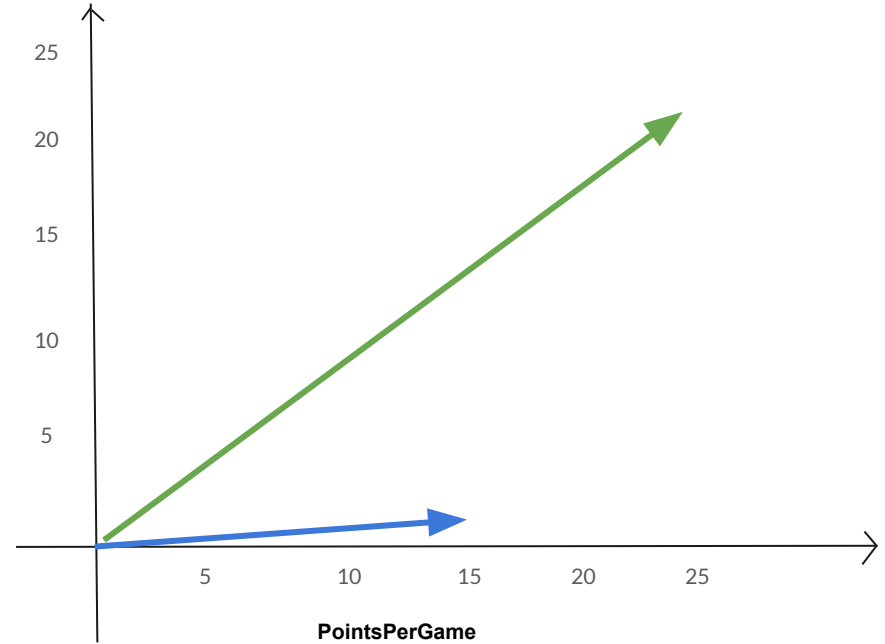
Addition results in a new “larger vector” (in this case)

Player1 = [25.00, 20.00]

Player2 = [15.59, 1.17]

Subtraction

$$\begin{aligned} \text{Player1} - \text{Player2} &= [25.00, 20.00] - [15.59, 1.17] \\ &= [25.00 - 15.59, 20.00 - 1.17] \\ &= [\dots, \dots] \end{aligned}$$



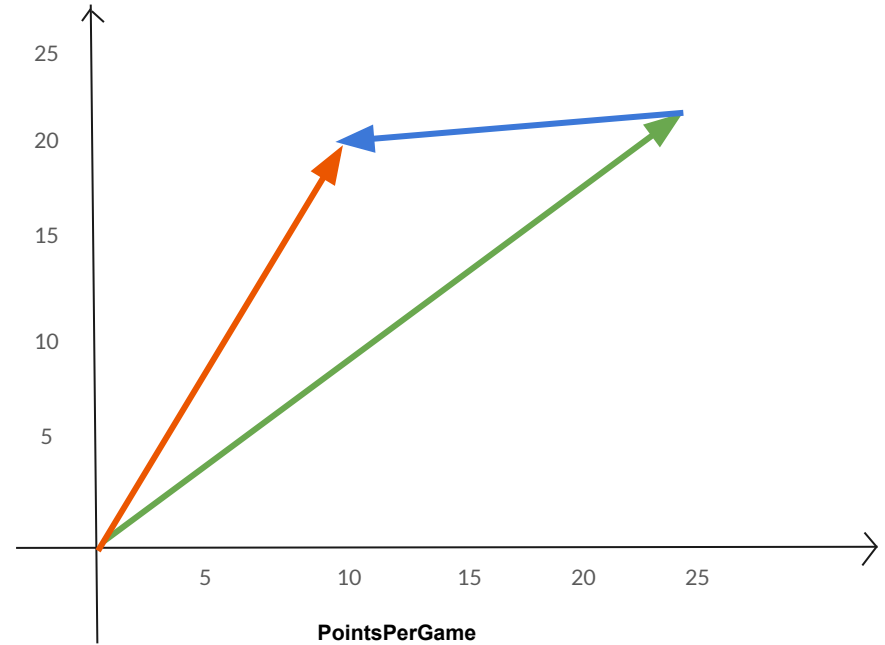
Whereas subtraction results in a new “smaller” vector (in this case)

Player1 = [25.00, 20.00]

Player2 = [15.59, 1.17]

Subtraction

$$\begin{aligned}\text{Player1} - \text{Player2} &= [25.00, 20.00] - [15.59, 1.17] \\ &= [25.00 - 15.59, 20.00 - 1.17] \\ &= [9.41, 18.83]\end{aligned}$$



*what if we did [15.59, 1.17] - [25.00, 20.00] instead?

Whereas subtraction results in a new “smaller” vector (in this case)

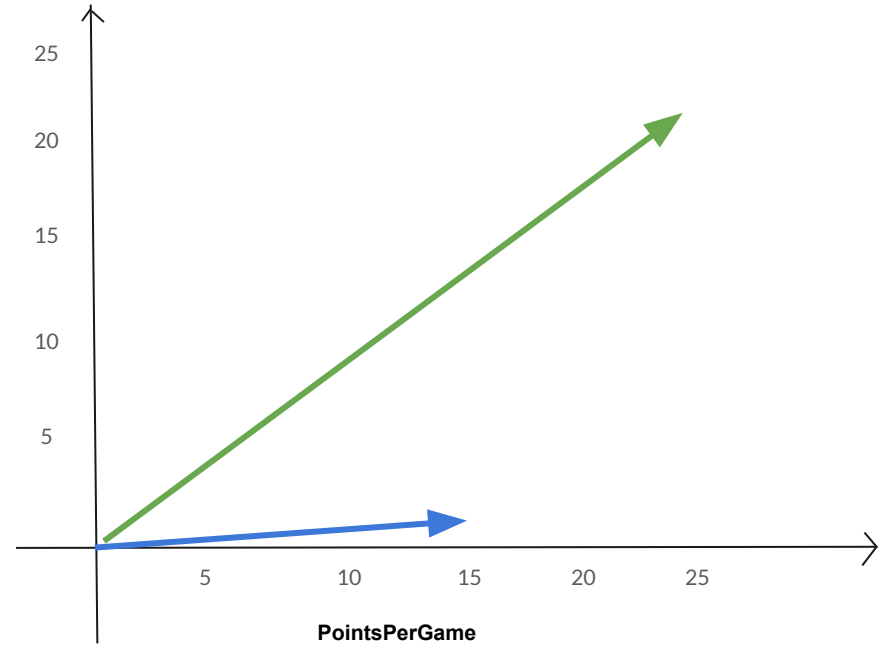
Player1 = [25.00, 20.00]

Player2 = [15.59, 1.17]

Scalar Multiplication

$$\begin{aligned} 2 * \text{Player2} &= 2 * [15.59, 1.17] \\ &= [2 * 15.59, 2 * 1.17] \\ &= [\dots, \dots] \end{aligned}$$

AssistsPerGame



We also have **scalar multiplication** has the effect of “**stretching**” a vector outwards (if we apply a positive scalar to a vector with **positive vector**)

Player1 = [25.00, 20.00]

Player2 = [15.59, 1.17]

Scalar Multiplication

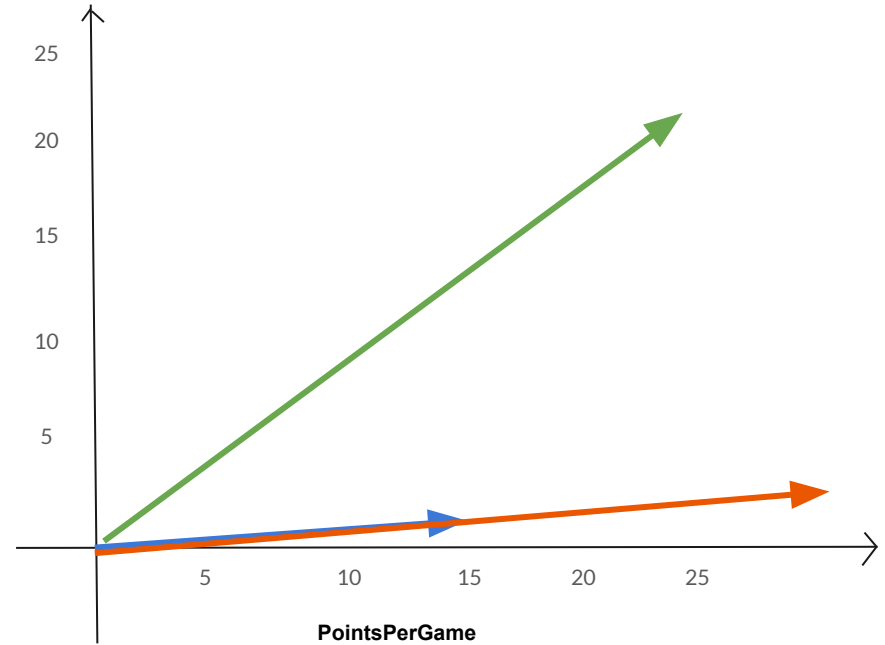
2 * **Player2**

= 2 * [15.59, 1.17]

= [2 * 15.59, 2 * 1.17]

= [31.18, 2.34]

AssistsPerGame



We also have **scalar multiplication** has the effect of “**stretching**” a vector outwards (if we apply a positive scalar to a vector with **positive vector**)

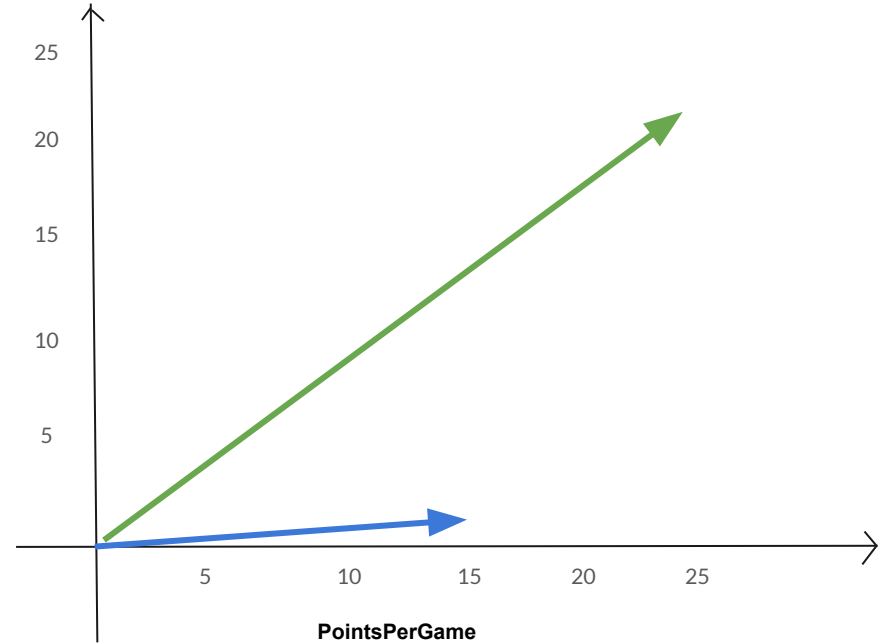
Player1 = [25.00, 20.00]

Player2 = [15.59, 1.17]

Magnitude

$$\begin{aligned} ||\textit{Player1}|| &= \text{sqrt}(A^2 + B^2) \\ &= \text{sqrt}(25^2 + 20^2) \\ &= \dots \end{aligned}$$

AssistsPerGame



We also have the magnitude of a vector, which is usually denoted by two bars “||” on either side of a vector. This denotes the total **distance that a vector covers**. Think back to the kNN algorithm, **what kind of distance measurement does this look similar to?**

Player1 = [25.00, 20.00]

Player2 = [15.59, 1.17]

Magnitude

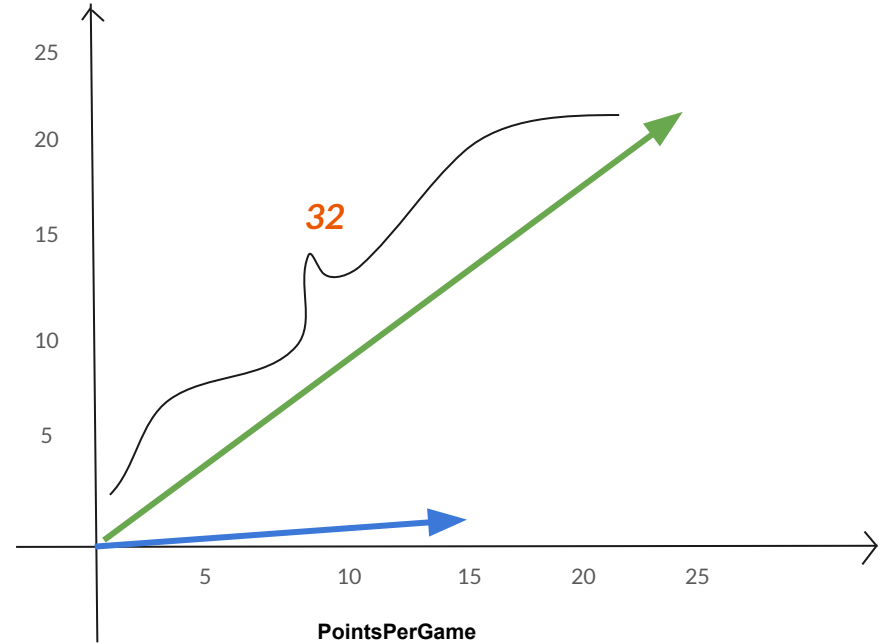
||Player1||

$= \text{sqrt}(A^2 + B^2)$

$= \text{sqrt}(25^2 + 20^2)$

~ 32

AssistsPerGame



This is just the euclidean distance AKA **solving the hypotenuse**.

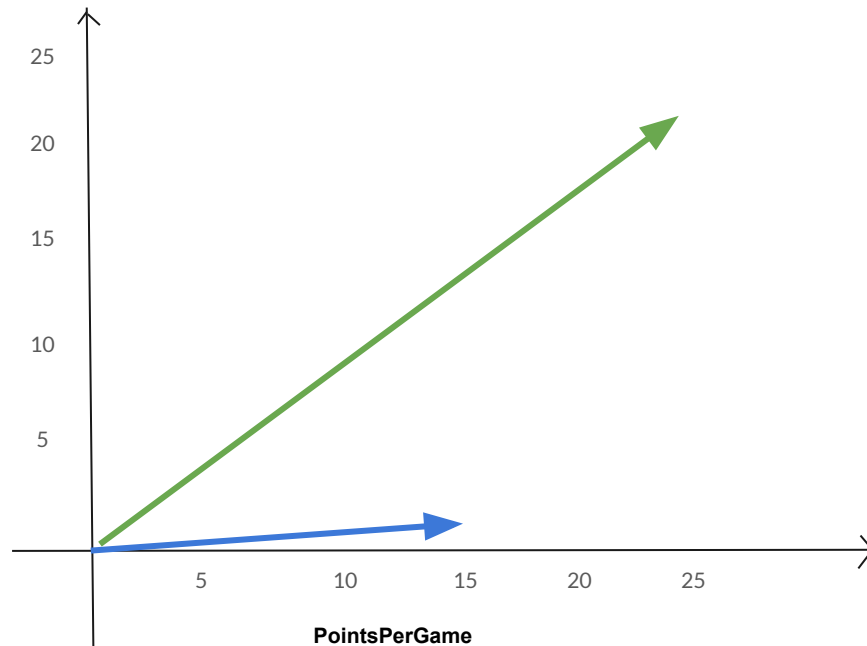
Player1 = [25.00, 20.00]

Player2 = [15.59, 1.17]

Dot Product

$$\begin{aligned} \text{Player1} * \text{Player2} &= \\ &= [25.00, 20.00] * [15.59, 1.17] \\ &= (25.00 * 15.59) + (20.00 * 1.17) \\ &= \dots \end{aligned}$$

AssistsPerGame



We also have the ability to apply the “**dot product**” to two vectors. **Note:** we must ensure that the **column** of the first matrix equals the **row** of the second matrix to take dot product. However if we are simply using two vectors, we can simply align values.

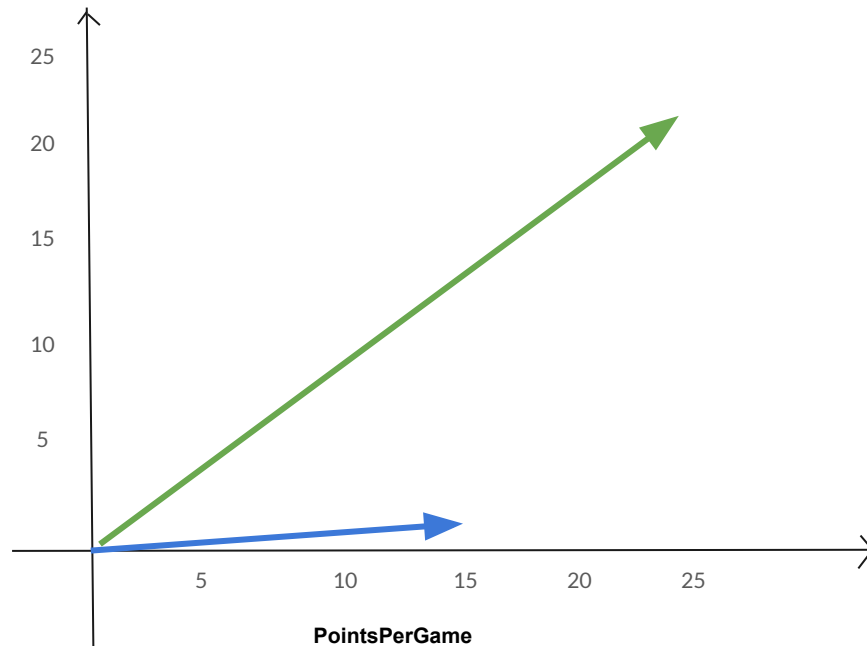
Player1 = [25.00, 20.00]

Player2 = [15.59, 1.17]

Dot Product

$$\begin{aligned}\text{Player1} * \text{Player2} &= \\ &= [25.00, 20.00] * [15.59, 1.17] \\ &= (25.00 * 15.59) + (20.00 * 1.17) \\ &= 413.15\end{aligned}$$

AssistsPerGame



But how do we interpret this value? Dot product is also known as **scalar product**. It represents how closely **two vectors align in direction**. For this reason, we often use dot product as an intermediate calculation for measures such as **cosine similarity**

Player1 = [25.00, 20.00]

Player2 = [15.59, 1.17]

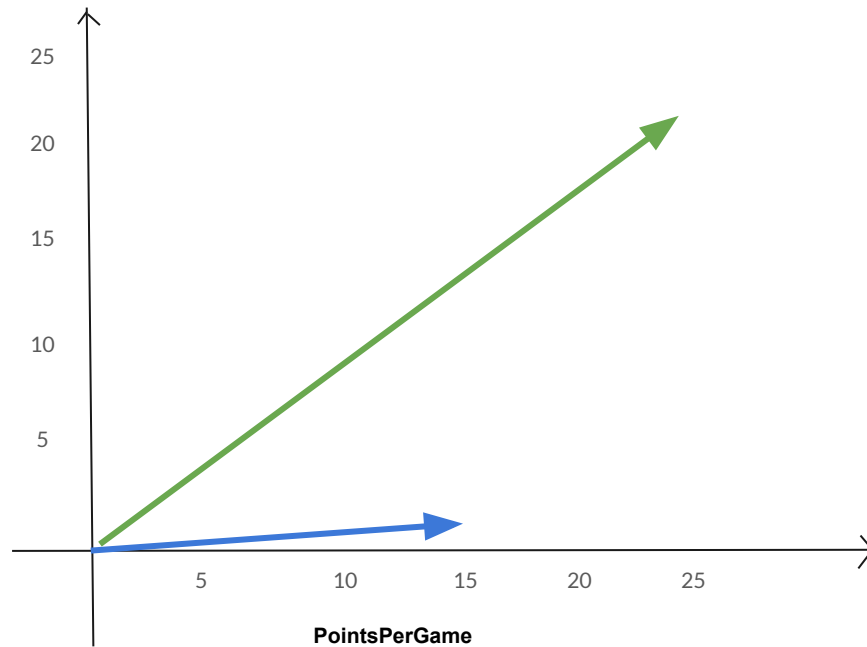
Cosine Similarity

Player1 * *Player2* = 413.15

$||\textit{Player1}|| = 32$

$||\textit{Player2}|| = 15.6$

$$\cos(\theta) = \frac{\textit{Player1} * \textit{Player2}}{||\textit{Player1}|| ||\textit{Player2}||}$$



Using the **dot product** and **magnitude**, we can calculate the angle (in radians) between two vectors and see if they are pointed in the same direction.

Player1 = [25.00, 20.00]

Player2 = [15.59, 1.17]

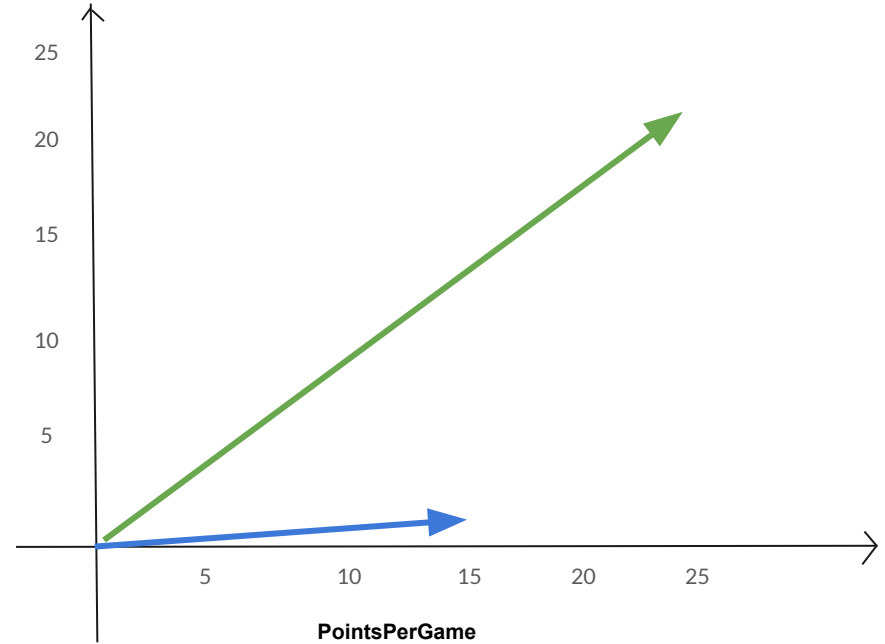
Cosine Similarity

Player1 * *Player2* = 413.15

$||\textit{Player1}|| = 32$

$||\textit{Player2}|| = 15.6$

$$\cos(\theta) = \frac{413.15}{32 * 15.6} \rightarrow \cos(\theta) = 0.82$$



Now that we have this equality, how do we solve for θ ???

Player1 = [25.00, 20.00]

Player2 = [15.59, 1.17]

Cosine Similarity

Player1 * **Player2** = 413.15

$||\mathbf{Player1}|| = 32$

$||\mathbf{Player2}|| = 15.6$

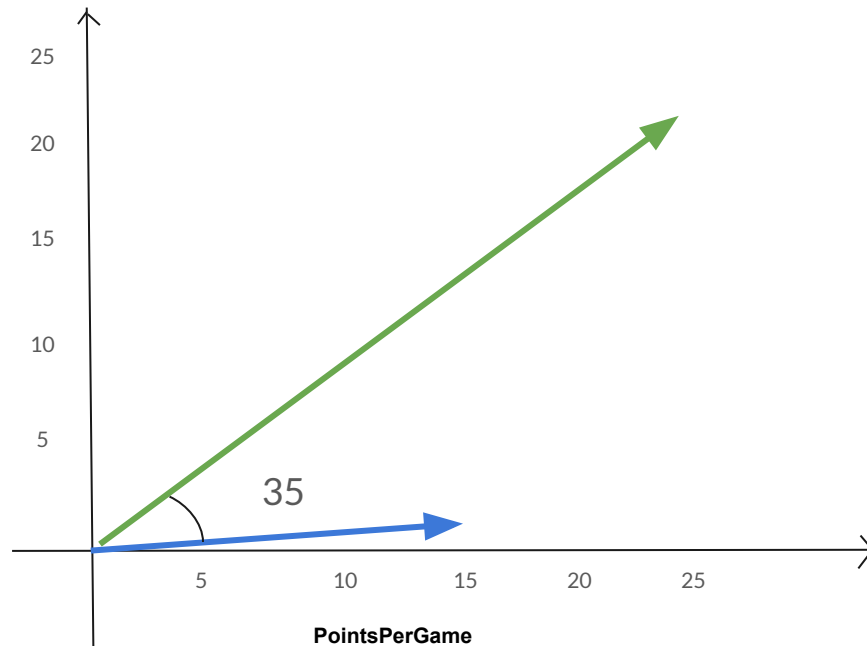
$$\cos(\theta) = \frac{413.15}{32 * 15.6}$$



$$\cos(\theta) = 0.82$$

$$\theta = \cos^{-1}(0.82)$$

$$\theta = 0.609 \text{ radians}$$



We take the inverse of cosine (aka arccosine) to calculate the radians between two vectors.

In this case we got a θ of 0.609 radians aka ~35 degrees. If our two vectors pointed in the same direction, what would we get instead??? What if they were perpendicular to one another?

Player1 = [25.00, 20.00]

Player2 = [15.59, 1.17]

Player3 = [18.62, 6.12]

Cosine Similarity

Player1 * Player3 = 587.9

||Player1|| = 32

||Player2|| = 19.59

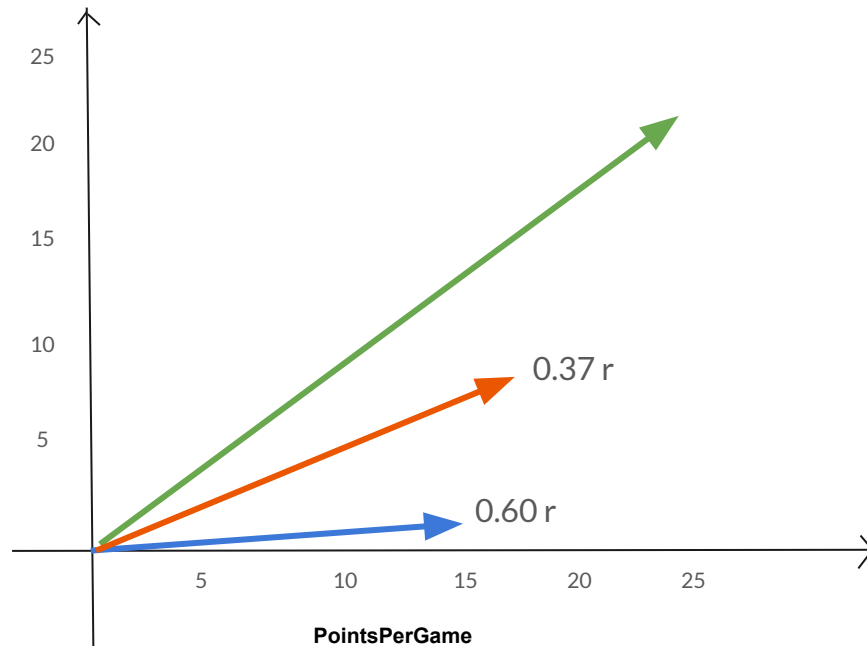
$$\cos(\theta) = \frac{587.9}{32 * 19.59}$$



$$\cos(\theta) = 0.93$$

$$\theta = \cos^{-1}(0.93)$$

$$\theta = 0.37 \text{ radians}$$



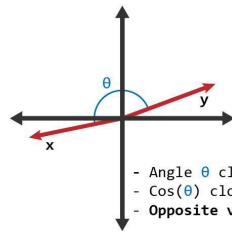
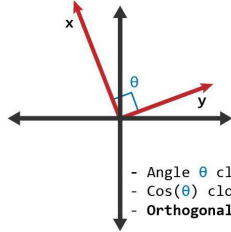
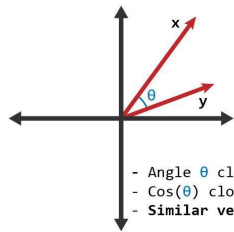
Let's consider **Player3** to see how **cosine similarity** differs across vectors. What do you notice about the cosine similarity of Player1 and Player3?

If we got a **cosine similarity** of 0, what would this indicate. What about a **cosine similarity** of $\pi/2$?

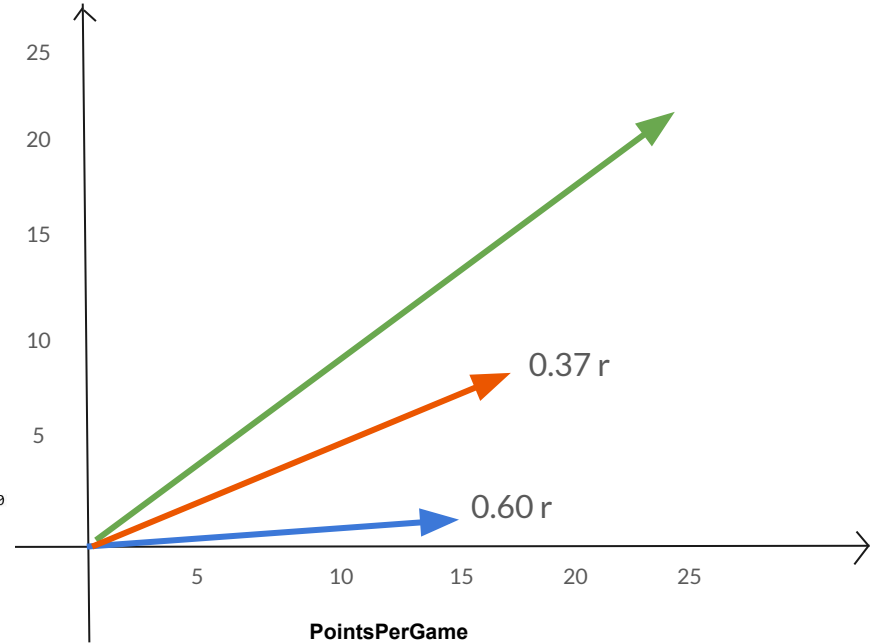
Player1 = [25.00, 20.00]

Player2 = [15.59, 1.17]

Cosine Similarity



AssistsPerGame

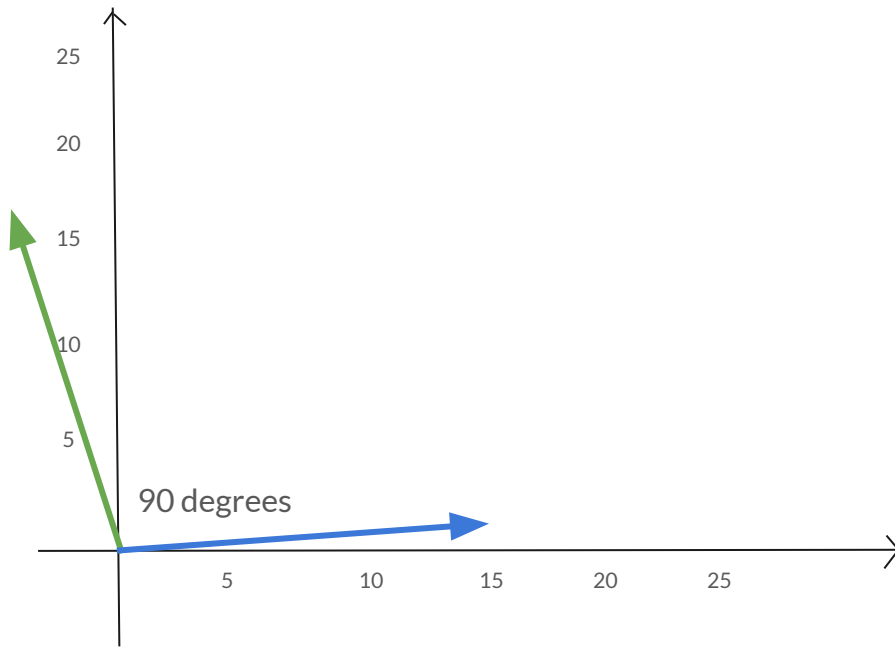


By solving for the angle between two vectors (players or data points), we observe how “similar” two data points are.

Note that this could also be used to measure the distance between two samples in kNN!

When two vectors are orthogonal to each other, they are completely **uncorrelated**.

That is, they share no linear relationship, therefore **they do not influence each other** and instead represent **two separate sources** of unbiased information.

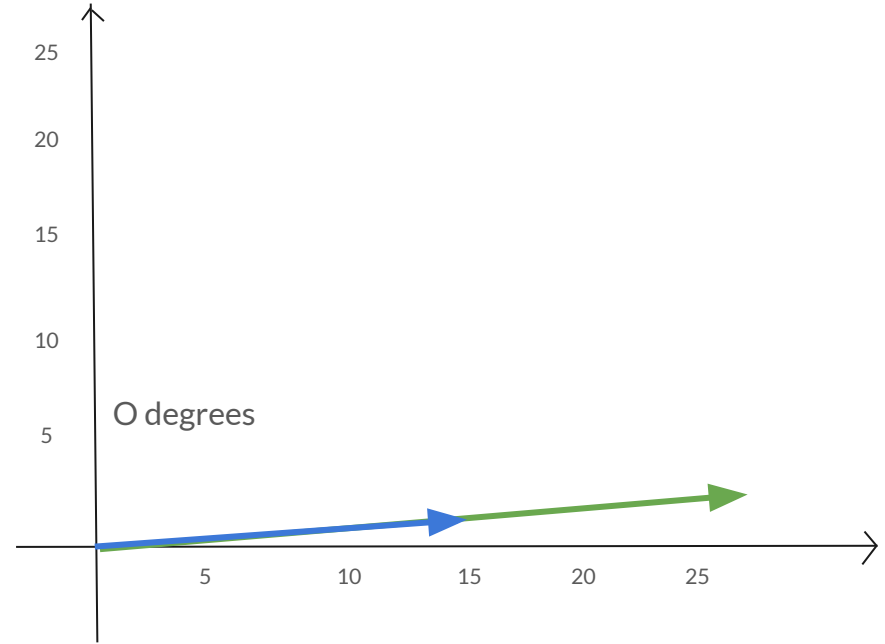


Before we leave the world of vectors, let's identify a few key terminologies that you will see spring up in the world of ML.

First is **orthogonality**, which is when two vectors are perpendicular to each other (90 degrees).

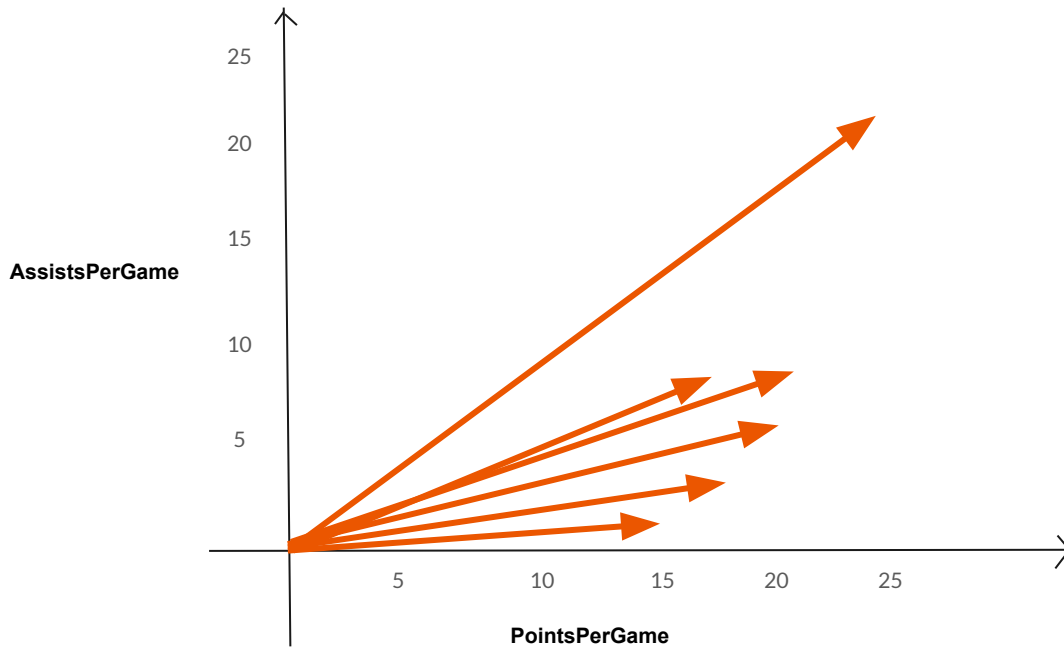
When two vectors are collinear to each other, they are completely **correlated**.

You are essentially looking at the same vector twice, and if you incorporate information from this vector, you run the risk of adding noise to your model.



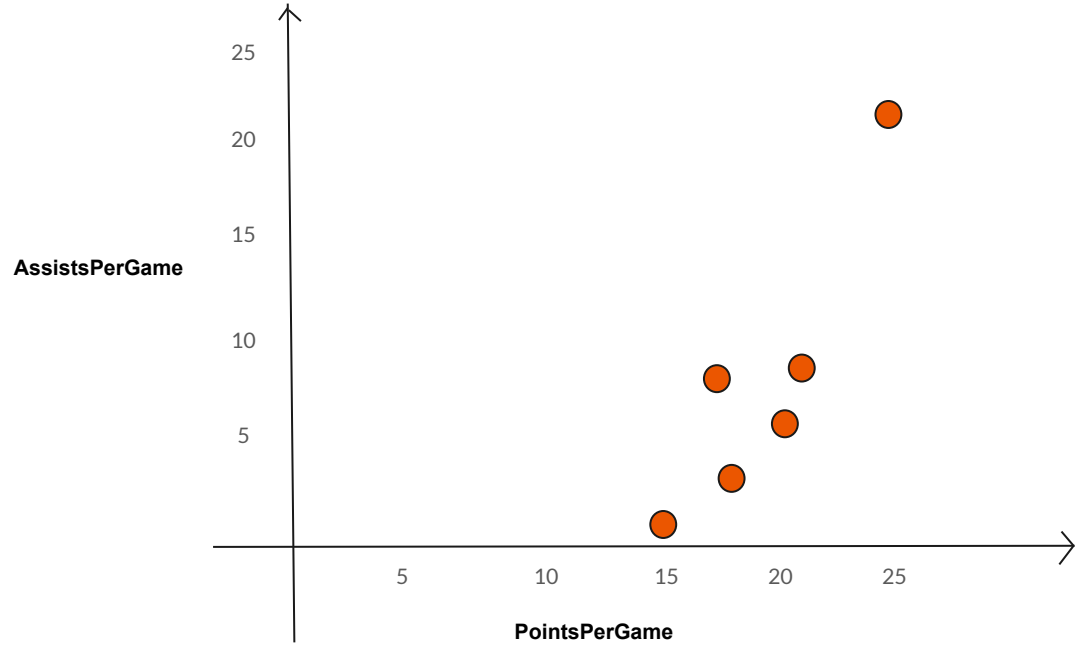
Next is **collinearity**. This is when vectors line on the same line or when they are parallel to each other.

```
team1 = [ 18.72, 3.38  
         22.88, 7.97  
         20.07, 5.11  
         25.0, 20.0  
         18.62, 6.12  
         15.59, 1.17]
```



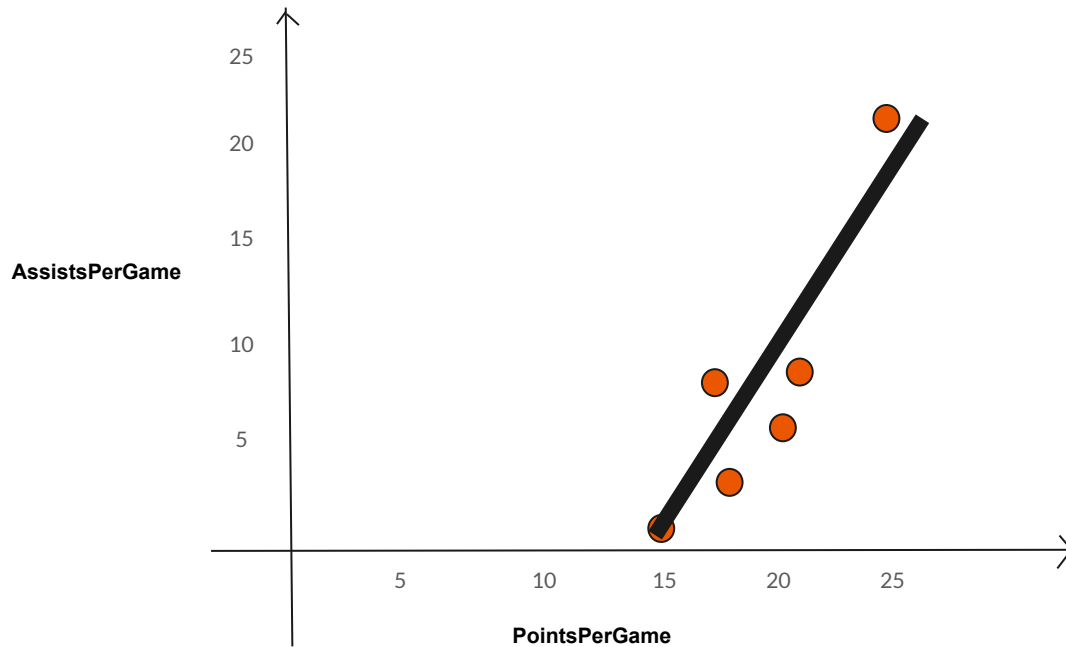
These vector definitions will become relevant again soon, but for now, let's combine all our players into a single matrix so that we can begin exploring dimensionality reduction.

```
team1 = [ 18.72, 3.38  
         22.88, 7.97  
         20.07, 5.11  
         25.0, 20.0  
         18.62, 6.12  
         15.59, 1.17]
```



Can you see any way we can express these data points on a single axes (line)?

```
team1 = [ 18.72, 3.38  
         22.88, 7.97  
         20.07, 5.11  
         25.0, 20.0  
         18.62, 6.12  
         15.59, 1.17]
```



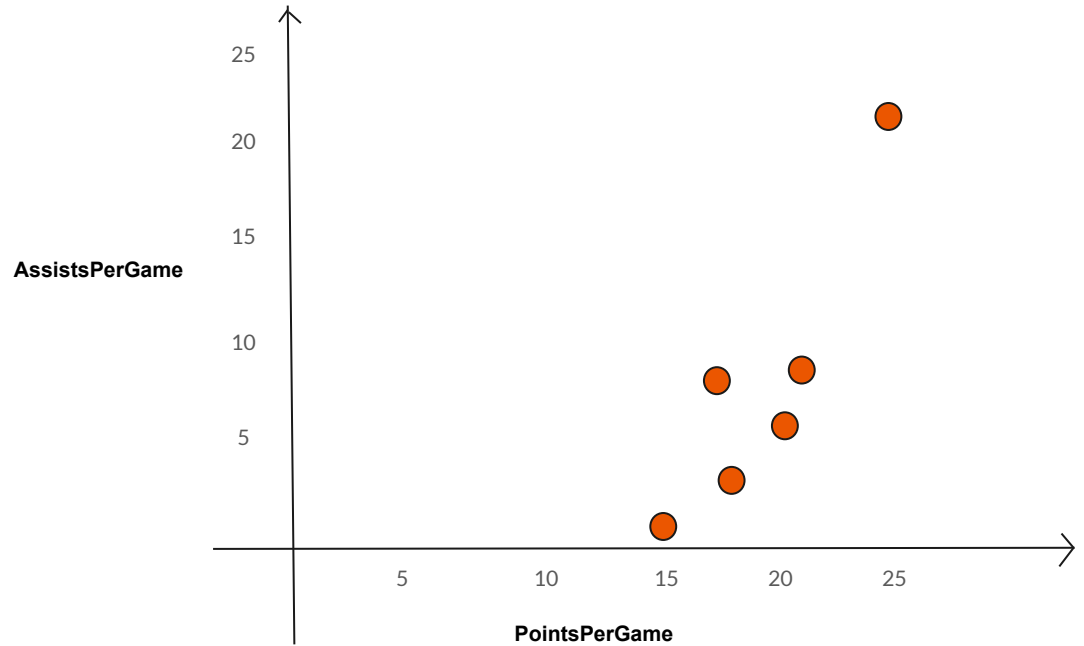
Notice how most of these points fall around **one axes** (sort of like a line of best fit). We will utilize PCA to reduce these points to this axes (1-dimension). **However we must first review eigenvectors.**

PCA Step 1 - Standardize

```
team1 = [ 18.72 - 20.15, 3.38 - 7.29  
         22.88 - 20.15, 7.97 - 7.29  
         20.07 - 20.15, 5.11 - 7.29  
         25.0 - 20.15, 20.0 - 7.29  
         18.62 - 20.15, 6.12 - 7.29  
         15.59 - 20.15, 1.17 - 7.29]
```

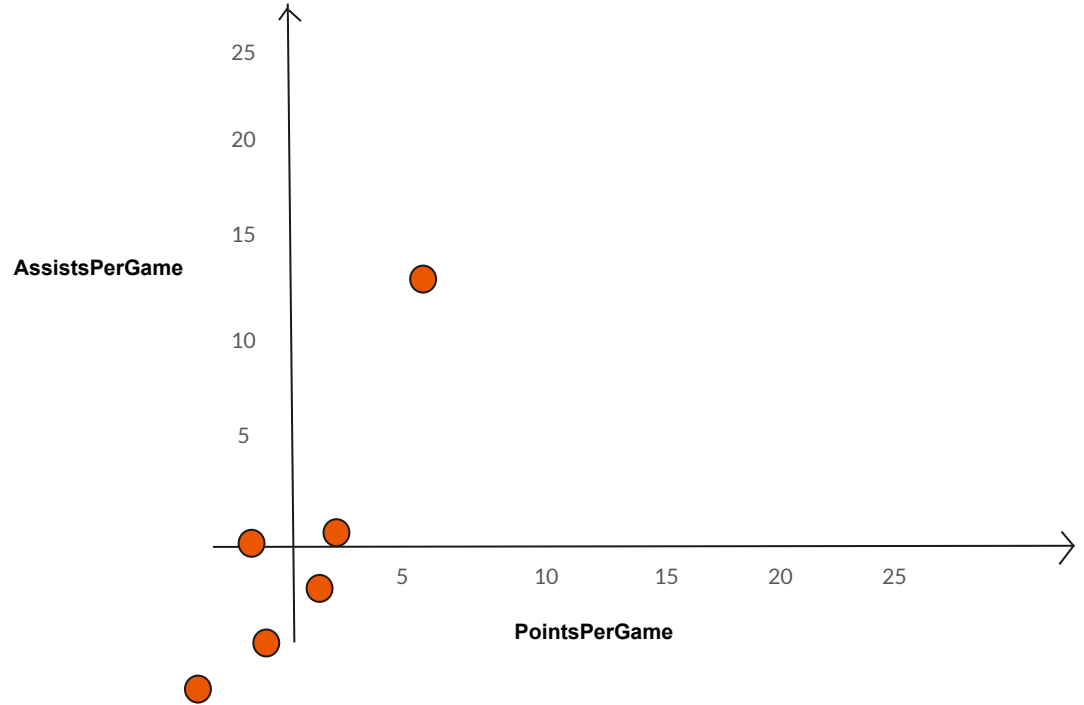
Col1 avg = 20.15

Col2 avg = 7.29



Step 1: Before moving forward, we must **standardize** this dataset around 0 by subtracting the average off of each sample.

```
team1 = [ -1.43, -3.91  
          2.73, 0.68  
         -0.08, -2.18  
          4.85, 12.71  
         -1.53, -1.17  
         -4.56, -6.12]
```



This simplifies all subsequent calculations.

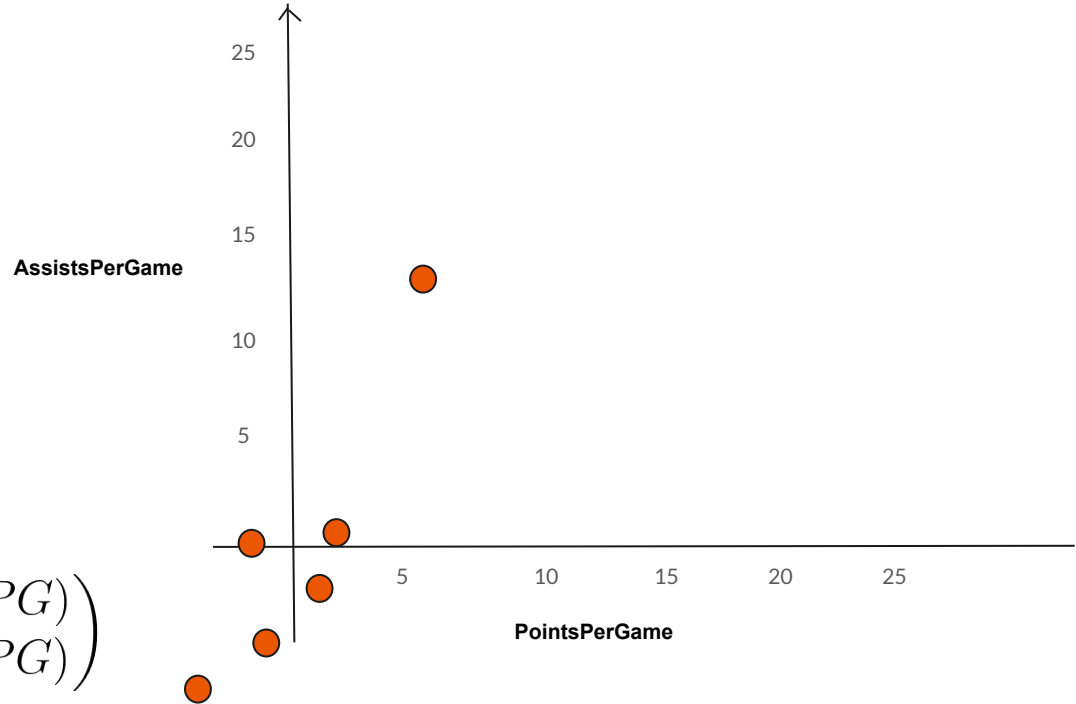
Next we calculate the **covariance matrix**.

PCA Step 2 - Covariance Matrix

```
team1 = [-1.43, -3.91  
         2.73, 0.68  
        -0.08, -2.18  
         4.85, 12.71  
        -1.53, -1.17  
        -4.56, -6.12]
```

For each column possible
combination of features,
we calculate covariance

$$\begin{pmatrix} Cov(PPG, PPG) & Cov(PPG, APG) \\ Cov(APG, PPG) & Cov(APG, APG) \end{pmatrix}$$



First, let's go over the concept of a **covariance matrix**.



An Aside - Covariance

It might seem like we are introducing a brand new formula with no context, but we've actually interacted with **covariance** already!

Notice that along our diagonal of the covariance matrix, **we calculate the covariance of a feature with itself.**

$$Cov(X, X)$$

$$Cov(x,y) = \frac{\sum (x_i - \bar{x}) * (y_i - \bar{y})}{(N - 1)}$$



An Aside - Covariance

Let's place the predictor "X" into our covariance function.

Is there any way we can simplify the numerator with an exponent?

$$Cov(X, X)$$



$$\frac{\sum (X_i - \bar{X}) * (\sum X_i - \bar{X})}{N - 1}$$



An Aside - Covariance

Yes, we can convert this into the **following** formula.

What is this formula also known as???

$$Cov(X, X)$$




$$\frac{\sum (X_i - \bar{X})^2}{N - 1}$$




An Aside - Covariance

Notice that this is the same as our
SAMPLE VARIANCE!


$$S^2 = \frac{\sum (x_i - \bar{x})^2}{n - 1}$$

$$\text{Cov}(X, X)$$


$$\frac{\sum (X_i - \bar{X})^2}{N - 1}$$

*What do you think a covariance of 0 indicates?

An Aside - Covariance

In summary, our sample covariance is the measure of joint variability between **two random variables**.

If covariance is large & positive, our two random variables move in the same direction (e.g. height & weight), whereas negative covariances indicate opposite directions (e.g. exercise & blood pressure).

$$\text{Cov}(x,y) = \frac{\sum (x_i - \bar{x}) * (y_i - \bar{y})}{(N - 1)}$$



An Aside - Covariance

As we observed, calculating the covariance of a predictor with itself gives us spread (variance)

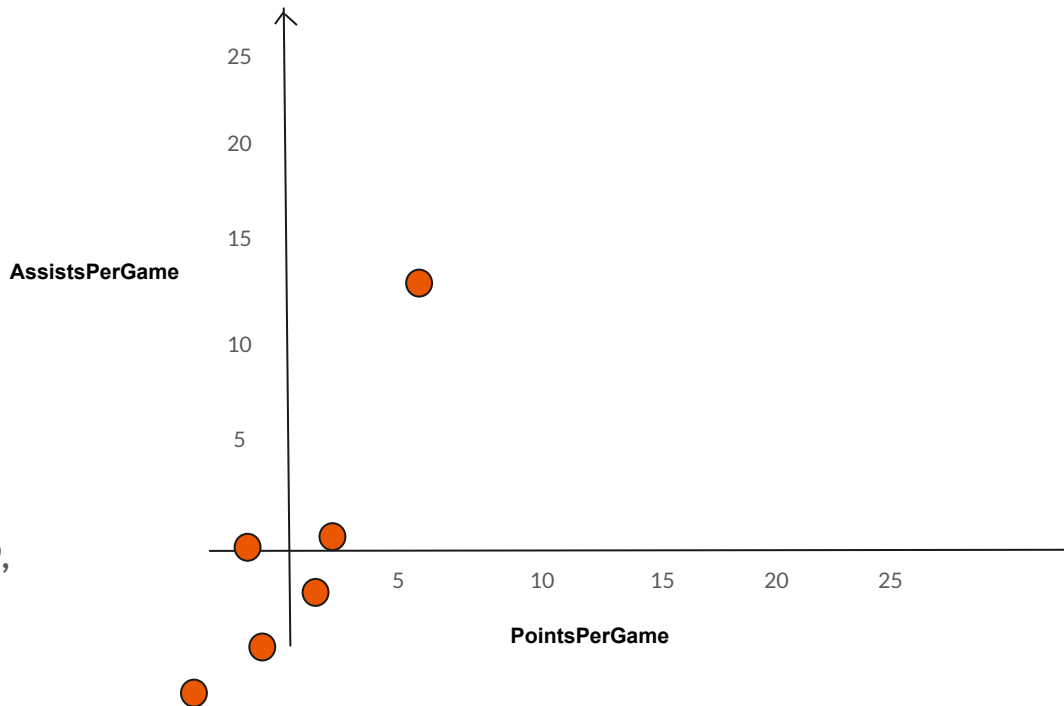
$$\frac{\sum (X_i - \bar{X})^2}{N - 1}$$

$team1 = [-1.43, -3.91$
 $2.73, 0.68$
 $-0.08, -2.18$
 $4.85, 12.71$
 $-1.53, -1.17$
 $-4.56, -6.12]$



$\begin{pmatrix} Cov(PPG, PPG) & Cov(PPG, APG) \\ Cov(APG, PPG) & Cov(APG, APG) \end{pmatrix}$

$[11.23, 19.79,$
 $19.79, 44.18]$



Let's continue our calculation of our covariance metric.

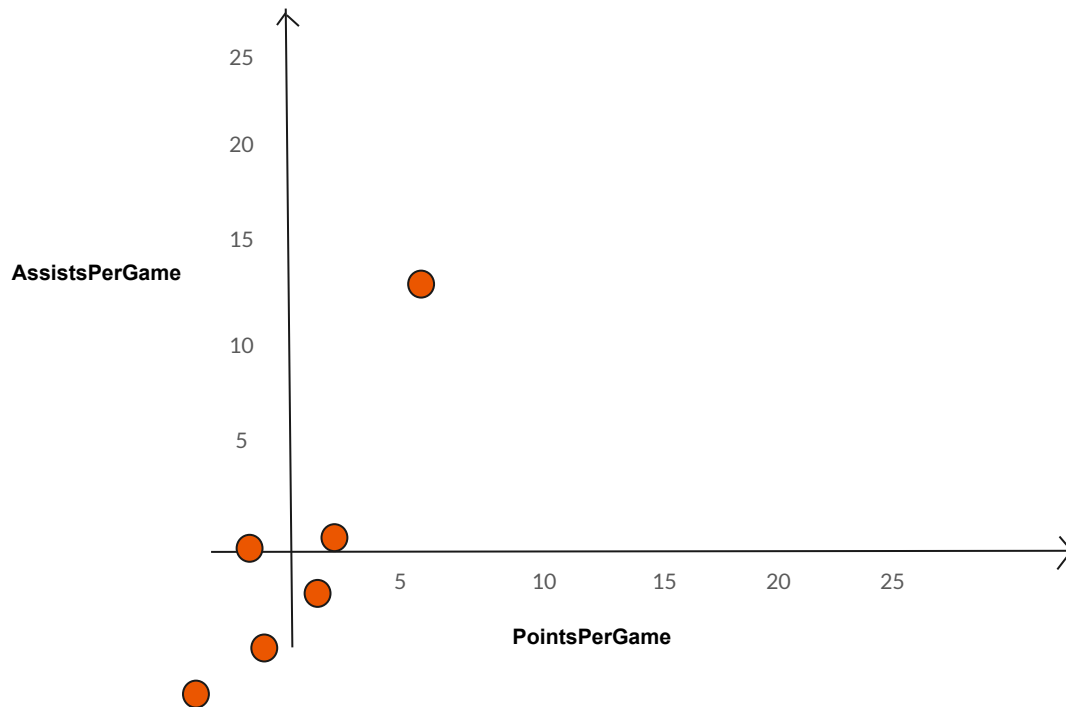
Along our diagonal we have **variance** for PPG & APG. In the corners of our matrix, we have the covariance of both **APG & PPG (19.79)**.

Next, we will calculate the eigenvectors of this covariance matrix.

PCA Step 3 - Eigenvalues & Eigenvectors

$team1 = \begin{bmatrix} -1.43, -3.91 \\ 2.73, 0.68 \\ -0.08, -2.18 \\ 4.85, 12.71 \\ -1.53, -1.17 \\ -4.56, -6.12 \end{bmatrix}$

Covariance matrix =
 $\begin{bmatrix} 11.23, 19.79, \\ 19.79, 44.18 \end{bmatrix}$

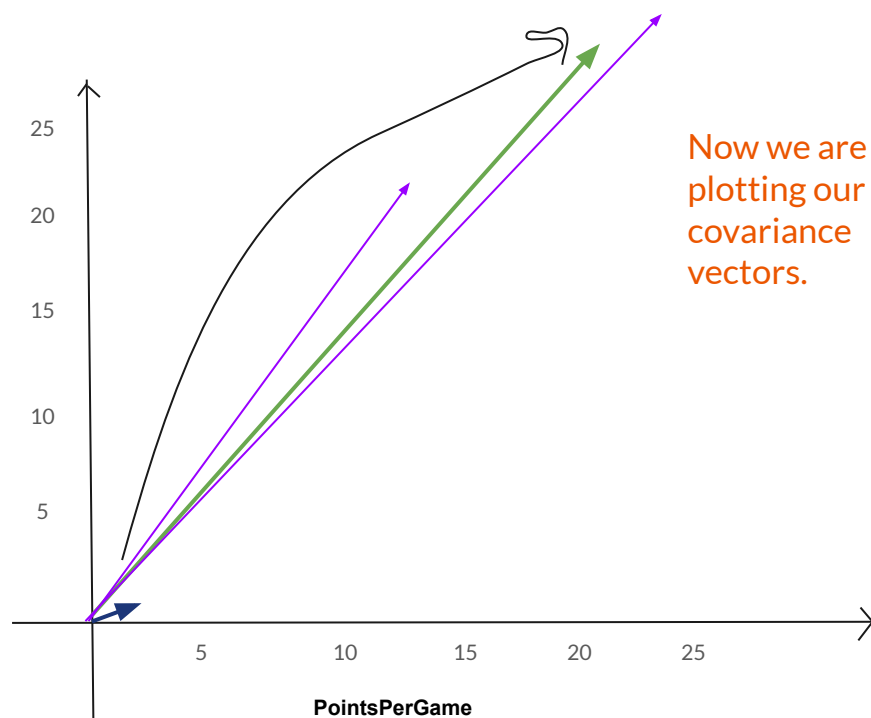


We use eigenvectors to answer: what are the main directions of variance in performance? (i.e. does PointsPerGame explain AssistsPerGame, or does AssistsPerGame explain PointsPerGame)?

Covariance matrix =
[11.23, 19.79,
19.79, 44.18]

$\text{Covariance} * [1, 1] = [31.02, 63.97]$

AssistsPerGame



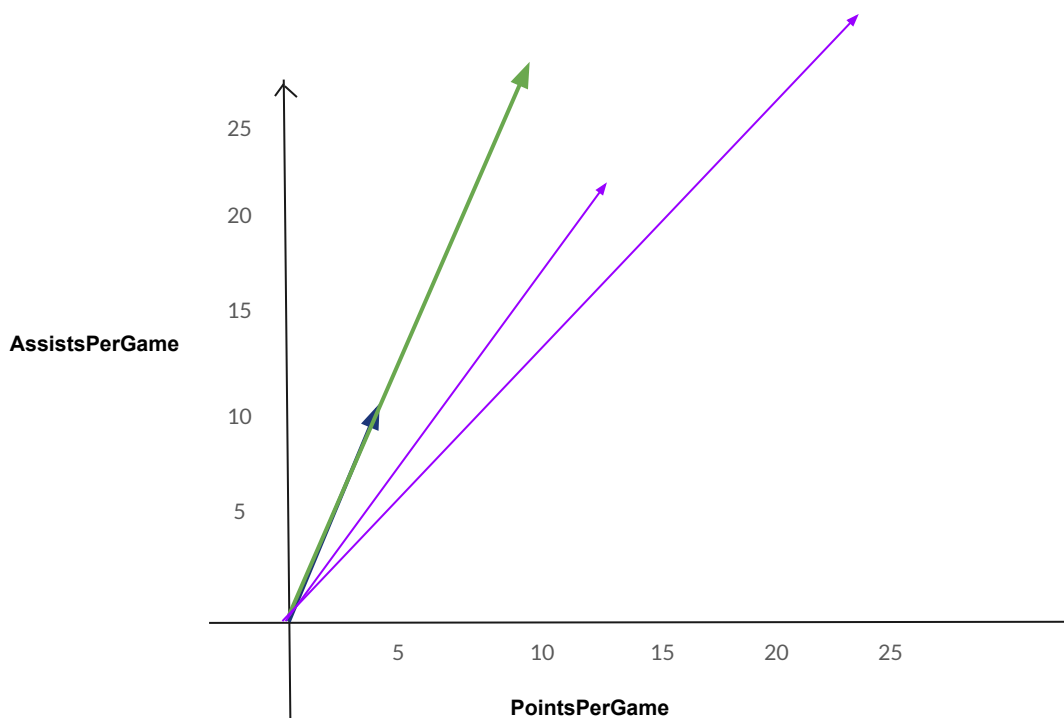
First, we must understand matrices as **linear transformations** of coordinate space.

For example, if we were to take a **vector** and calculate its dot product with this **matrix**, this would result in the coordinate moving to another point in our cartesian plot.

Covariance matrix =
[11.23, 19.79,
19.79, 44.18]

$\text{Covariance} * \mathbf{v1} = \mathbf{v2}$

Where $\mathbf{v2}$ has a cosine similarity of 0 with $\mathbf{v1}$



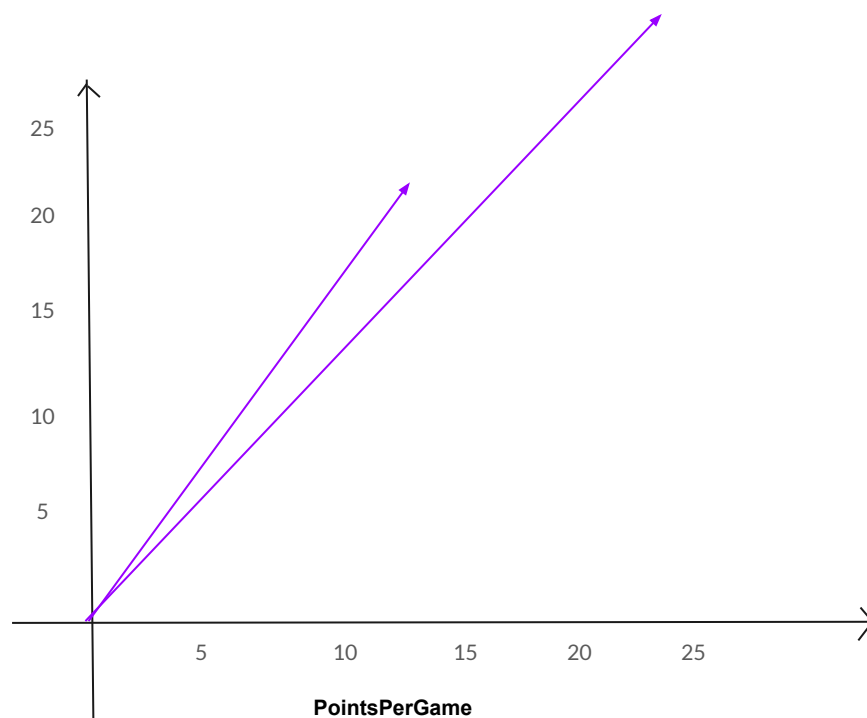
However, what about the case where our vector **does not experience any change in direction** after **these linear transformations**?

This vector would instead describe the **most meaningful axis or direction of rotation**

Covariance matrix =
[11.23, 19.79,
19.79, 44.18]

$$A\vec{v} = \lambda\vec{v}$$

AssistsPerGame



For *most* datasets, there exists **vectors** that **keep their direction** after this matrix multiplication.

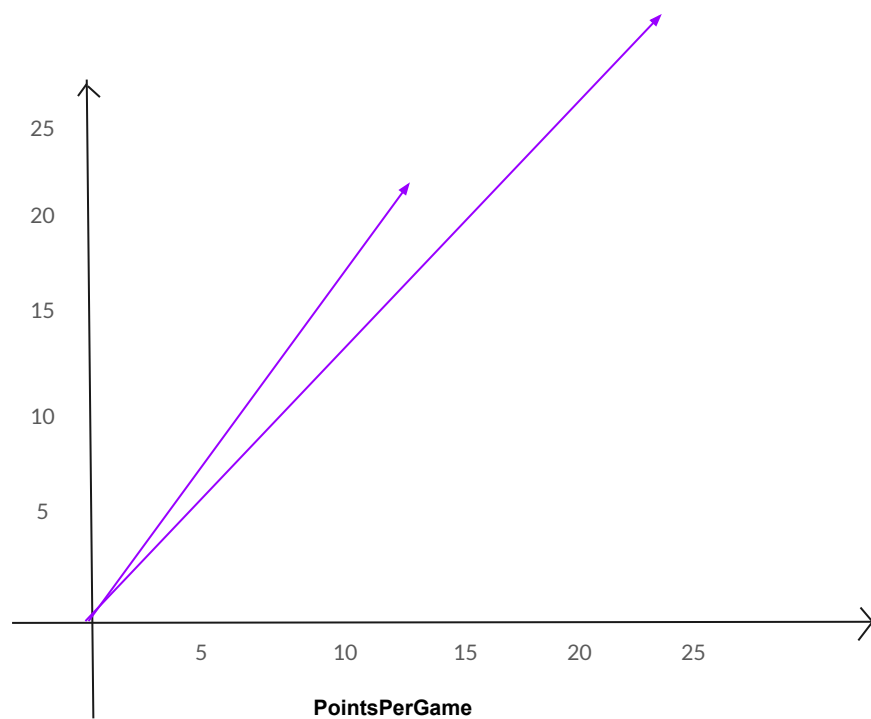
We call these vectors **eigenvectors** and their scale of transformation **eigenvalues**. We signify these components via the terminology above, where **"A"** is our matrix (dataset of samples).

Covariance matrix =
 [11.23, 19.79,
 19.79, 44.18]

$$A\vec{v} = \lambda\vec{v}$$

Original Matrix	=	Eigenvectors Matrix	Eigenvalues Matrix	Inverse of Eigenvectors Matrix
$\begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix}$	=	$\begin{bmatrix} -1 & -1 \\ 2 & 1 \end{bmatrix}$	$\begin{bmatrix} -2 & 0 \\ 0 & -1 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 \\ -2 & -1 \end{bmatrix}$

AssistsPerGame



Discovering the eigenvectors and eigenvalues of a matrix is **not** a trivial operation, so we will skip this step and only focus on the calculation of eigenvalues in order to discover the most important components of a dataset.

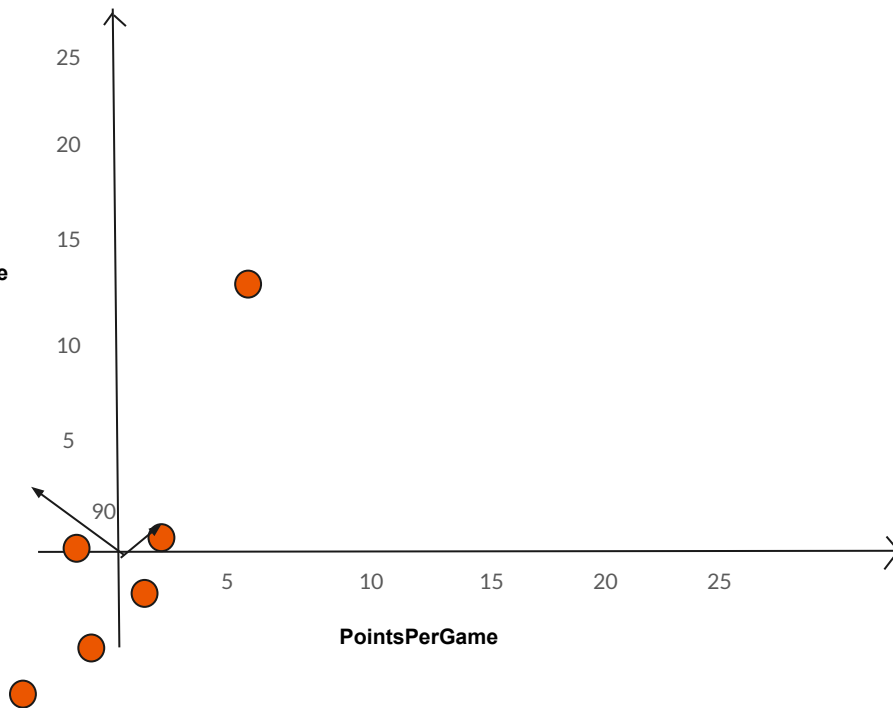
$$\begin{pmatrix} \text{Cov}(PPG, PPG) & \text{Cov}(PPG, APG) \\ \text{Cov}(APG, PPG) & \text{Cov}(APG, APG) \end{pmatrix} = \begin{bmatrix} 11.23 & 19.79 \\ 19.79 & 44.18 \end{bmatrix}$$

eigenvalue 1 = 53.45
eigenvector 1 = [-2.13, 1]

eigenvalue 2 = 0.632
eigenvector 2 = [0.46, 1]

We will leave this calculation as a mystery

AssistsPerGame



We calculate the eigenvalues and eigenvectors of this covariance matrix using our eigen-decomposition algorithm, which give us the following vectors and values.

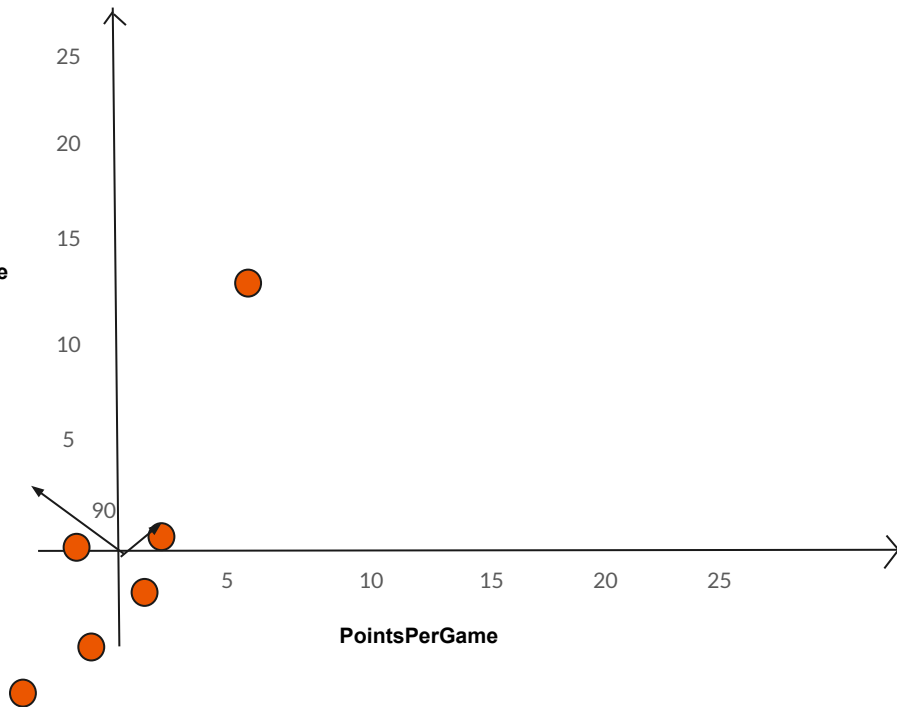
What do you notice about our eigenvectors? What kind of vectors are they in relation to one another?

$$\begin{pmatrix} \text{Cov}(PPG, PPG) & \text{Cov}(PPG, APG) \\ \text{Cov}(APG, PPG) & \text{Cov}(APG, APG) \end{pmatrix} = \begin{bmatrix} 11.23 & 19.79 \\ 19.79 & 44.18 \end{bmatrix}$$

eigenvalue 1 = 53.45
eigenvector 1 = [-2.13, 1]

eigenvalue 2 = 0.632
eigenvector 2 = [0.46, 1]

AssistsPerGame



These are **orthogonal vectors**. While these aren't the principal components themselves, we've successfully captured the unique directions of this dataset. This will allow us to calculate new uncorrelated dimensions.

$$\begin{pmatrix} \text{Cov}(PPG, PPG) & \text{Cov}(PPG, APG) \\ \text{Cov}(APG, PPG) & \text{Cov}(APG, APG) \end{pmatrix} = \begin{bmatrix} 11.23 & 19.79 \\ 19.79 & 44.18 \end{bmatrix}$$

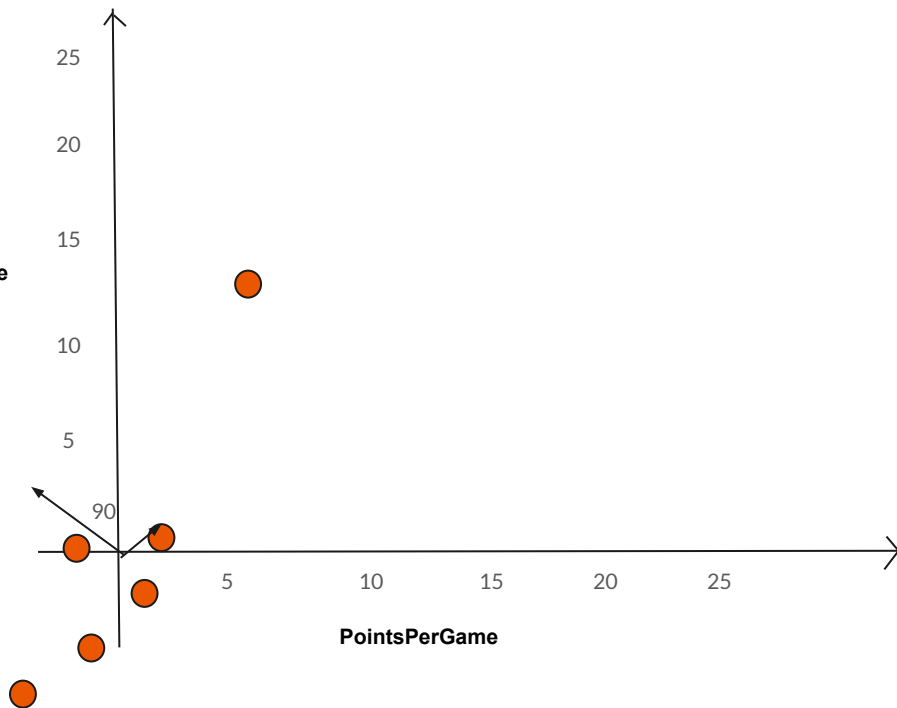
eigenvalue 1 = 53.45
eigenvector 1 = [-2.13, 1]

variance =
 53.45/(53.45+0.632)

eigenvalue 2 = 0.632
eigenvector 2 = [0.46, 1]

variance =
 0.632/(53.45+0.632)

AssistsPerGame



Furthermore we calculate how **much variance** is explained by each eigenvector by calculating the ratio of their **respective eigenvalues**.

The higher the value (from 0 to 1) the more **important** this dimension is.

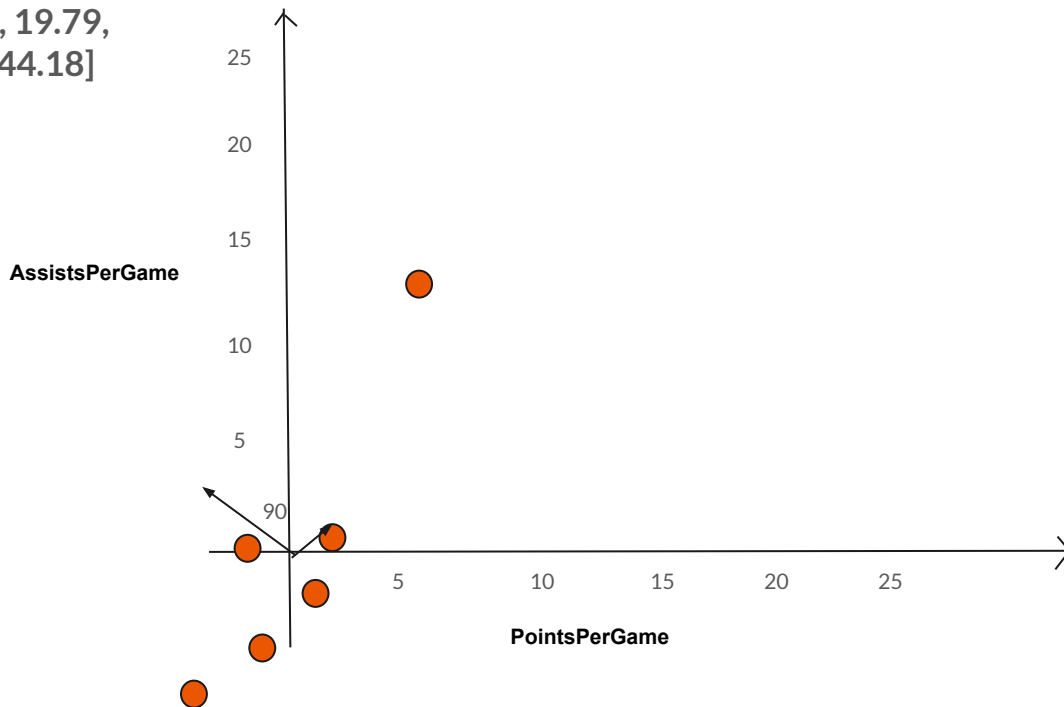
$$\begin{pmatrix} \text{Cov}(PPG, PPG) & \text{Cov}(PPG, APG) \\ \text{Cov}(APG, PPG) & \text{Cov}(APG, APG) \end{pmatrix} = \begin{bmatrix} 11.23 & 19.79 \\ 19.79 & 44.18 \end{bmatrix}$$

eigenvalue 1 = 53.45
eigenvector 1 = [-2.13, 1]

variance =
0.9883

eigenvalue 2 = 0.632
eigenvector 2 = [0.46, 1]

variance =
0.0116

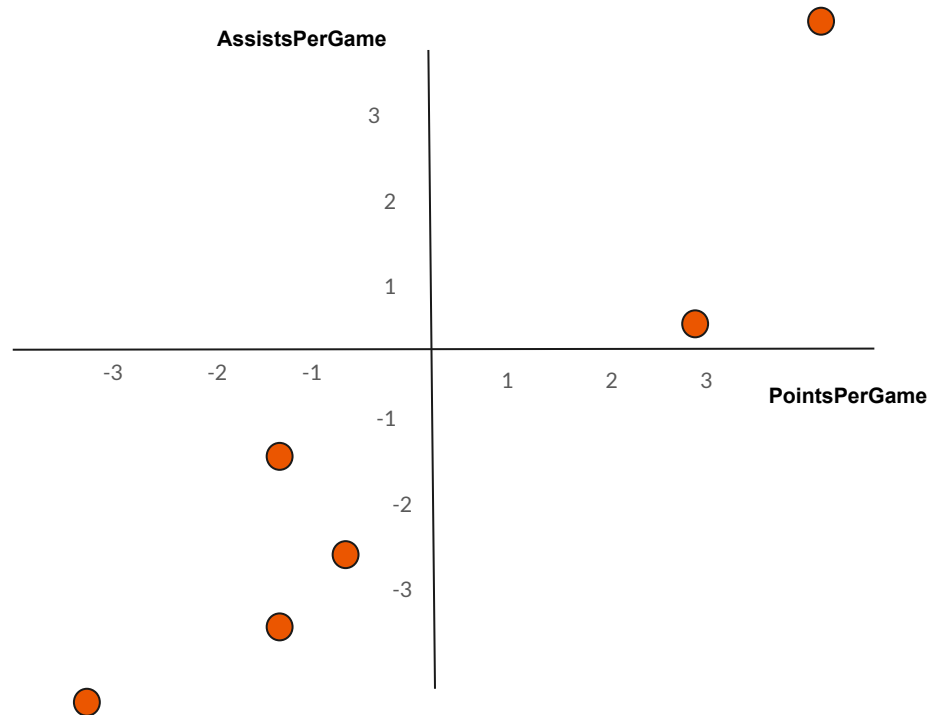


We can see that **eigenvector 1** accounts for roughly 98% of variance in our data. That is, we can explain 98% of movement in our dataset using this single dimension.

$team1 = [-1.43, -3.91$
2.73, 0.68
-0.08, -2.18
4.85, 12.71
-1.53, -1.17
-4.56, -6.12]

$eigenvalue\ 1 = 53.45$
 $eigenvector\ 1 = [-2.13, 1]$

$eigenvalue\ 2 = 0.632$
 $eigenvector\ 2 = [0.46, 1]$



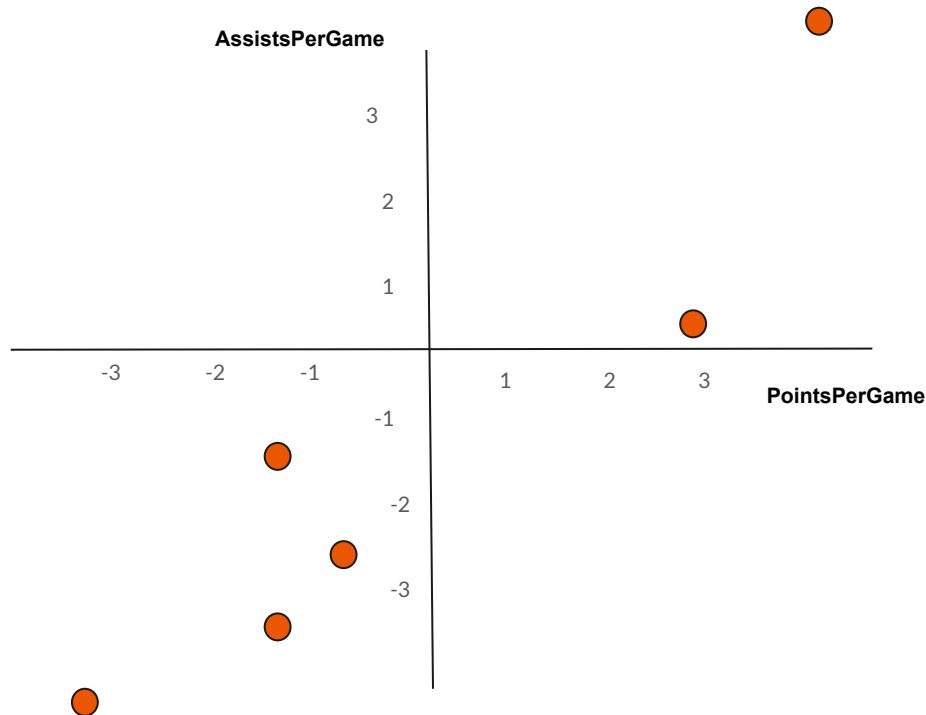
We will now center our graph around 0 to better visualize the PCA transformation.

PCA Step 4 - Mapping Values to Lower Dimensions

$V =$
 $\begin{bmatrix} -2.13 & 0.46 \\ 1 & 1 \end{bmatrix}$

$team1 = \begin{bmatrix} -1.43, -3.91 \\ 2.73, 0.68 \\ -0.08, -2.18 \\ 4.85, 12.71 \\ -1.53, -1.17 \\ -4.56, -6.12 \end{bmatrix}$

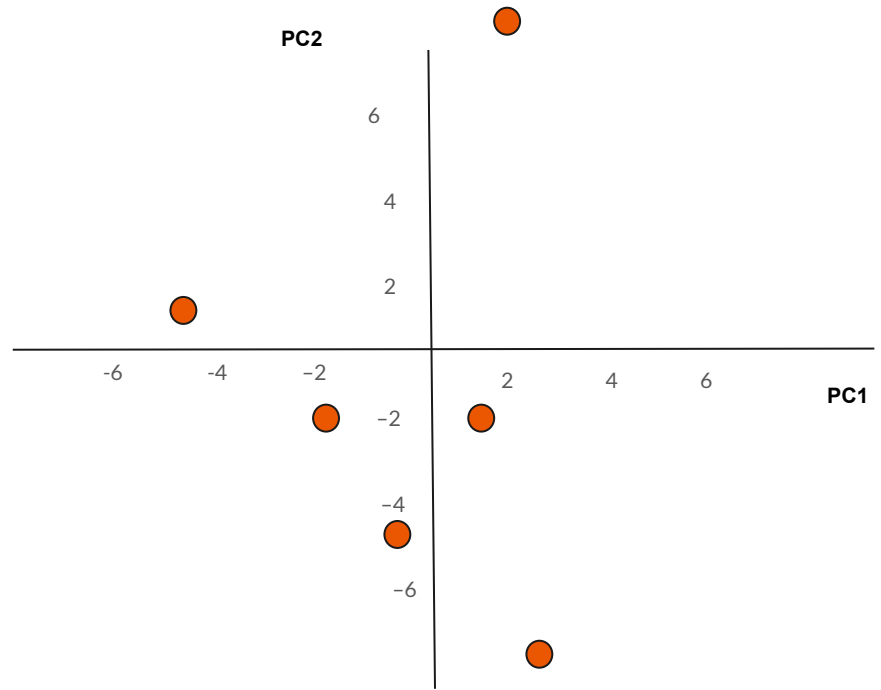
$team1 * V = \dots$



We can multiply our combination of eigenvectors (V) with our standardized matrix ($team1$) to **create new features that express our principal components.**

$V =$
 $\begin{bmatrix} -2.13 & 0.46 \\ 1 & 1 \end{bmatrix}$
 $team1 = \begin{bmatrix} -1.43, -3.91 \\ 2.73, 0.68 \\ -0.08, -2.18 \\ 4.85, 12.71 \\ -1.53, -1.17 \\ -4.56, -6.12 \end{bmatrix}$

$team1 * V = \begin{bmatrix} -0.86, -4.56 \\ -5.13, 1.94 \\ -2.01, -2.22 \\ 2.38, 14.94 \\ 2.09, -1.87 \\ 3.59, -8.22 \end{bmatrix}$



Remember the main purpose of PCA is **reduce our number of variables**, currently it seems that we still have 2 variables. However, let's recall how did we **determine the importance of each eigenvector**?

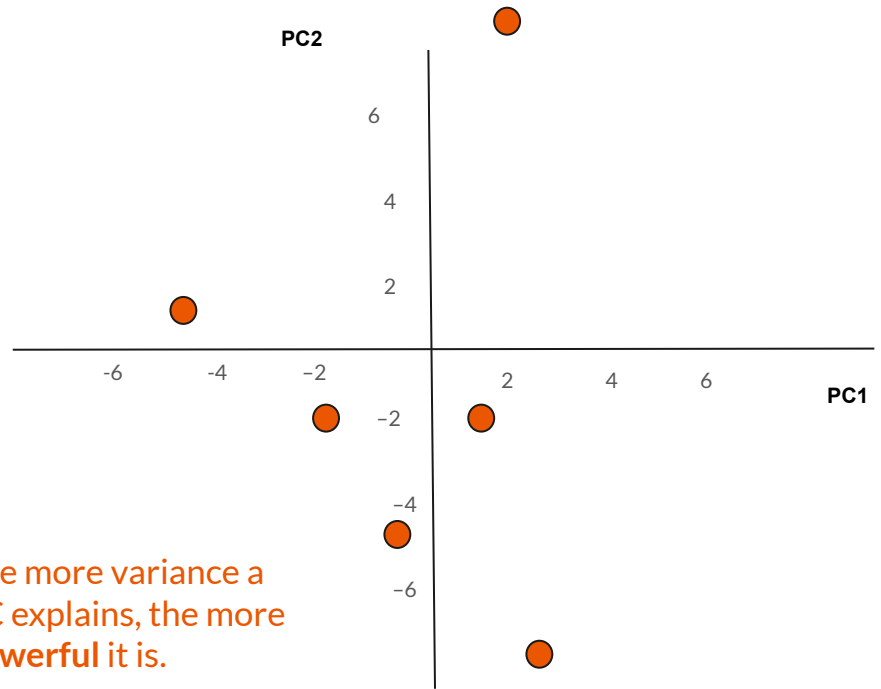
$V = \begin{bmatrix} -2.13 & 0.46 \\ 1 & 1 \end{bmatrix}$
 $team1 = \begin{bmatrix} -1.43, -3.91 \\ 2.73, 0.68 \\ -0.08, -2.18 \\ 4.85, 12.71 \\ -1.53, -1.17 \\ -4.56, -6.12 \end{bmatrix}$

$team1 * V = \begin{bmatrix} -0.86, -4.56 \\ -5.13, 1.94 \\ -2.01, -2.22 \\ 2.38, 14.94 \\ 2.09, -1.87 \\ 3.59, -8.22 \end{bmatrix}$

PC1 variance = 0.9883

PC2 variance = 0.0116

The more variance a PC explains, the more powerful it is.



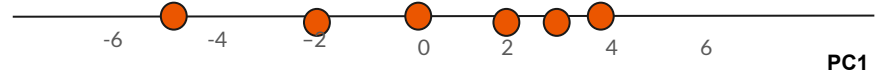
We utilize the **proportion of variance explained** to determine the importance of principal components. The general rule of thumb is to utilize the **smallest number of components** that explain 80%-90% of variance. Which PC explains almost 100% of variance?

$$V = \begin{bmatrix} -2.13 & 0.46 \\ 1 & 1 \end{bmatrix}$$

$$team1 = \begin{bmatrix} -1.43 & -3.91 \\ 2.73 & 0.68 \\ -0.08 & -2.18 \\ 4.85 & 12.71 \\ -1.53 & -1.17 \\ -4.56 & -6.12 \end{bmatrix}$$

$$team1 * V = \begin{bmatrix} -0.86, \\ -5.13, \\ -2.01, \\ 2.38, \\ 2.09, \\ 3.59 \end{bmatrix}$$

PC1 variance =
0.9883



You've just reduced your dataset to 1 dimension
and kept 98% of original variance!

PC1. Therefore we can simply discard PC2 and use PC1 in our ML algorithm to get the least noisy predictions. Remember, your unseen (test) data must also be multiplied by your calculated eigenvectors.



PCA Summary

For repetition, the **Principal Component Analysis (PCA)** entails the following steps:

1. **Standardize** the data via the mean
2. Calculate the **covariance** matrix
3. Calculate the **eigenvalues and eigenvectors** of the covariance matrix
4. Utilize eigenvectors to calculate your new **principal components**.

Additional Implementations of Dimensionality Reduction



Additional Dimensionality Reduction Techniques

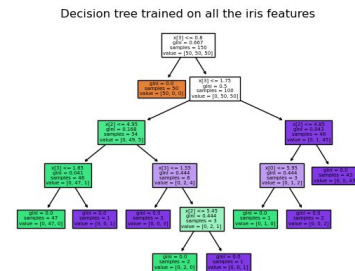
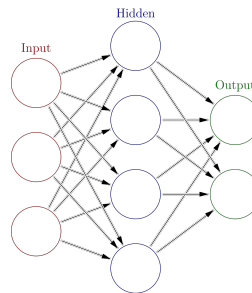
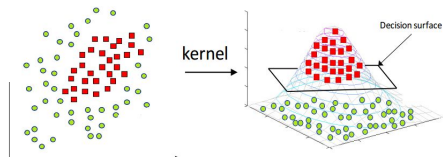
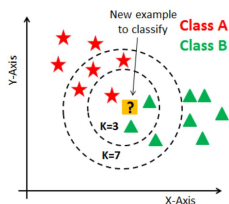
While PCA is the first dimensionality reduction algorithm almost all ML engineers learn, it is by no means is the only technique. There is also:

- t-SNE (manifold learning):
<https://www.datacamp.com/tutorial/introduction-t-sne>
- Independent Component Analysis
- Linear Discriminant Analysis
- Low Variance Filter
- **Transformers**

PCA Lab

PCA Lab

Complete the `pca.ipynb` notebook.

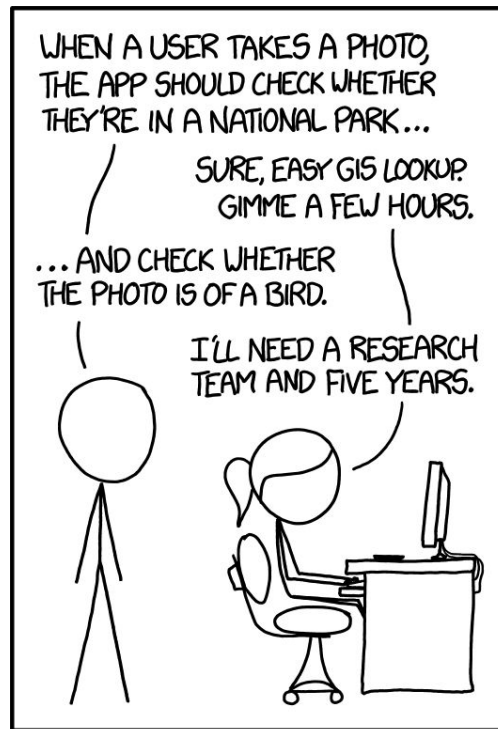


Wrap Up

Thursday

ML Review

- Review ML concepts from the week
- Work on TLAB #2



IN CS, IT CAN BE HARD TO EXPLAIN THE DIFFERENCE BETWEEN THE EASY AND THE VIRTUALLY IMPOSSIBLE.



Lab (Due 04/22)



You are a data scientist working at an up & coming music platform startup that just secured its Series B financing.

With this infusion of new cash flow, you are tasked with building an **unsupervised recommendation algorithm** that will recommend users new songs based on their previous listening history. As this is a brand new project, you will have to build this project from **scratch**.

Submit a link to your GitHub repository by **4/22**.