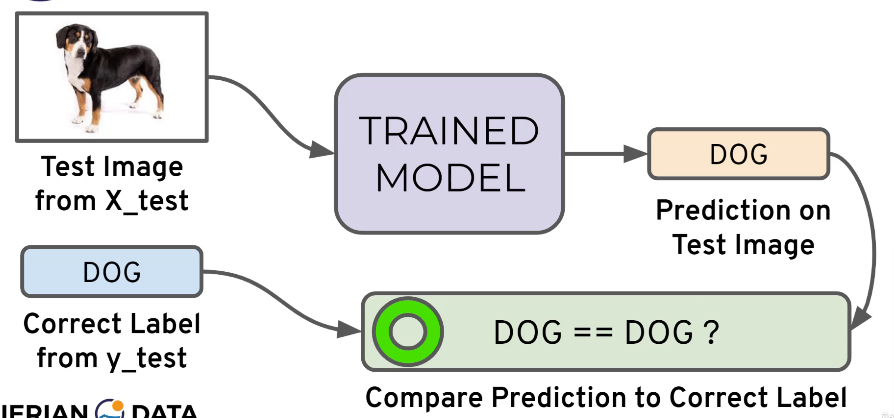
# Classification Metrics

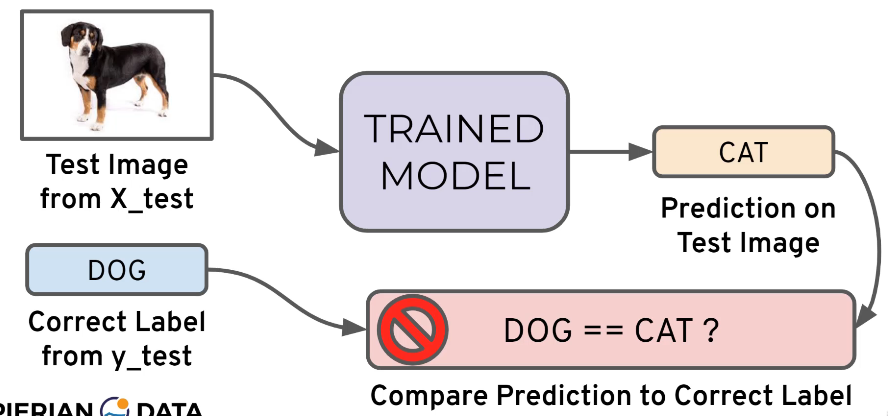
The key classification metrics are:

1. Accuracy
2. Recall
3. Precision
4. F1-Score

## Reason For classification metrics

* Typically, in any classification task the model can only achieve two results:  
  1. Either the model was correct in its prediction  
  2. Or the model was incorrect in its prediction
* Fortunately, incorrect vs correct expands to situations where there are multiple classes
* For the purposes of explaining the metrics, lets imagine a binary classification situation, where there are only two methods available
* In this example, we will attempt to predict if an image is a dog or a cat.
* Since this is a supervised learning, we will first **fit/train** a model on training data, then **test** the model on **testing data.**
* Once we have the model’s predictions from the **X\_test** data, we compare it to the **true y values** (the correct labels).





* In real world, a single metric won’t tell the complete story
* To understand all of this, lets bring back the 4 metrics mentioned and see how they are calculated.
* We could organize our predicted values compared to the real values in a **confusion matrix.**

### Accuracy

* Accuracy in classification problems is the **number of correct predictions** made by the model divided by the **total number of predictions**.
* For example, if the X\_set was 100 images and our model **correctly** predicted 80 images, then we have 80/100 i.e., **0.8 or 80% accuracy**.
* Accuracy is useful when the target classes are well balanced.
* In our example, we would have roughly the same amount of cat images as we have dog images.
* Accuracy is not a good choice with unbalanced classes
* Imagine we had 99 images of dogs and only 1 image of a cat.
* If our model was simply a line that always predicted dog, we would get 99% accuracy. However, since our model is made to predict both dogs and cats, the accuracy in real world would be different.

In this situation we will want to know more about recall and precision.

### Recall

* Ability of a model to find all the relevant cases within a dataset.
* The precise definition of recall is the number of true positives divided by the number of true positives plus the number of false negatives.

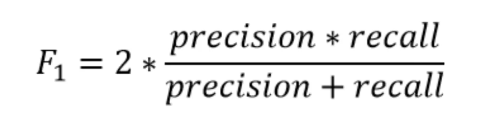
### Precision

* Ability of a classification model to identify only the relevant data points.
* Precision is defined as the number of true positives divided by the number of true positives plus the number of false positives.

#### Recall and Precision

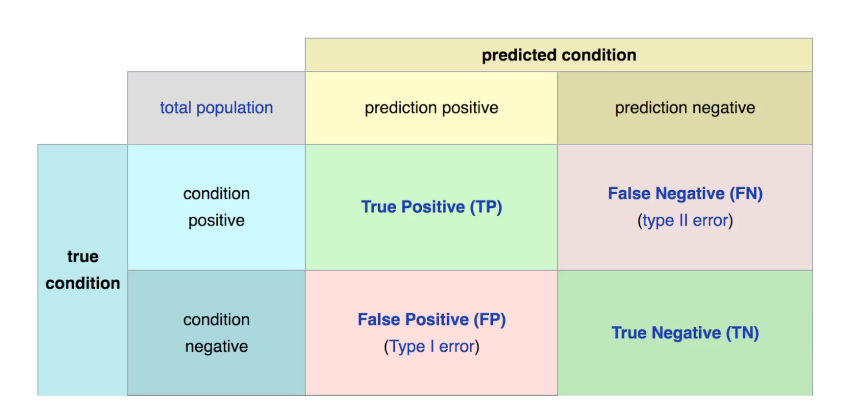
* Often there is a trade-off between Recall and Precision
* While recall expresses the ability to find all the relevant instances in a dataset, precision expresses the proportion of the data points our model says was relevant and actually were relevant.

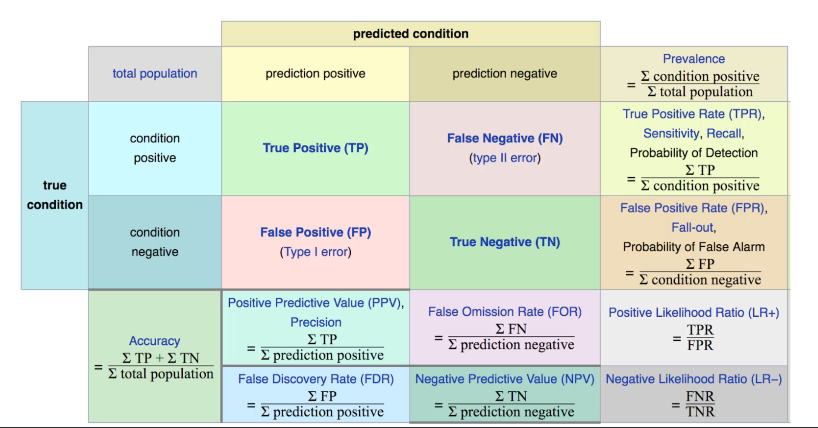
### F1 Score

* In cases where we want to find an optimal blend of precision and recall we can combine the two metrics using what is called the F1 Score.
* The F1 score is the harmonic mean of precision and recall taking both metrics into account in the following equation:  
  
* The reason we use the harmonic mean instead of a simple average is because it punishes the extreme values.
* A classifier with a precision of 1.0 and a recall of 0.0 has a simple average of 0.5 but an F1 score of 0.

## Confusion Matrix

We can also view all correctly classified versus incorrectly classified images in the form of a confusion matrix.





* The main point to remember with the confusion matrix and the various calculated metrics is that they are all fundamentally ways of comparing the predicted values versus the true values.
* What constitutes a “good” metrics will really depend on the specific situation.
* The good enough accuracy depends on the context of the situation.
* Often models are used as quick diagnostic tests to have before having a more invasive test
* Often, we have a precision/recall trade off. We need to decide if the model should focus on fixing the False Positives vs the False Negatives.
* In disease diagnosis, it is probably better to go in the direction of False Positive, so we make sure we correctly classify as many cases of disease as possible.
* All of this is to say, machine learning is not performed in a "vacuum", but instead a collaborative process where we should consult with experts in the domain (e.g., medical doctors)

## Evaluating Regression

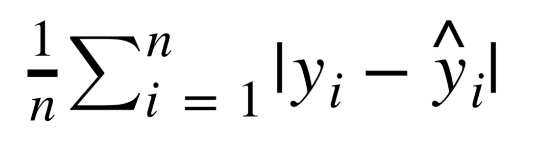
* Regression is a task when a model attempts to predict continuous values (unlike categorical values, which is classification).
* The previous metrics are not much useful for regression problems. We need metrics designed for continuous values.
* For example, attempting to predict the price of a house given its features is a regression task.
* Attempting to predict the country a house is in given its features would be a classification task instead.

Most common evaluation metrics for regression:

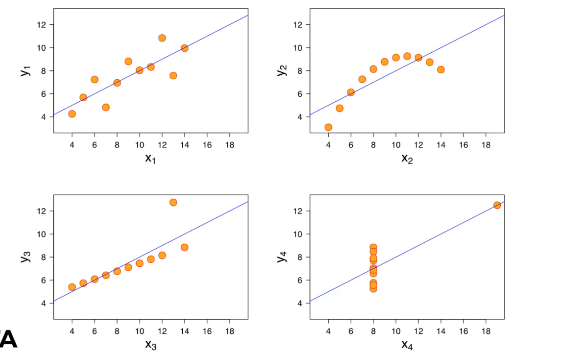
* Mean Absolute Error
* Mean Squared Error
* Root Mean Square Error

### Mean Absolute Error (MAE)

* This is the mean of the absolute value of errors.
* It is easy to understand.

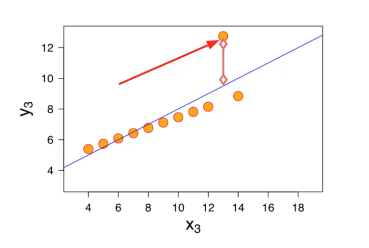


#### MAE won't punish large errors

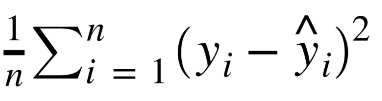


Although the data are varied, the line is the same.

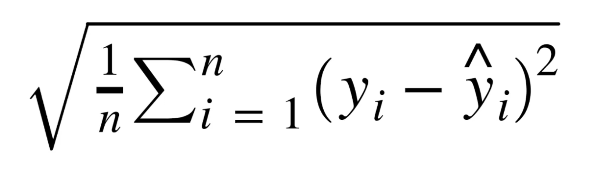
* We want our error metrics to account for cases like the following:



### Mean Square Error (MSE)

* This is the mean of the squared errors
* Larger errors are noted more than that with MAE, making MSE more popular.  
    
  
* One issue that should be considered is that the units also get squared.

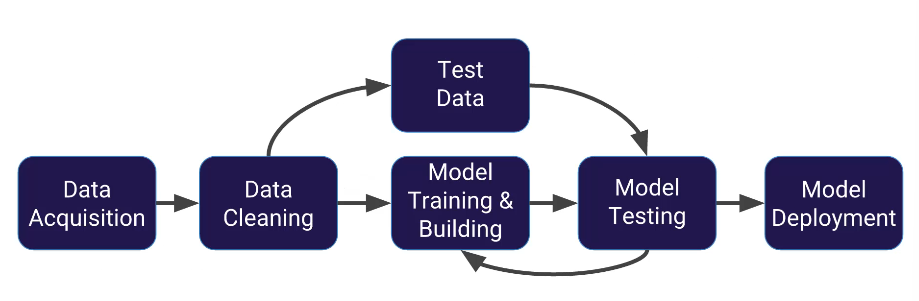
### Root Mean Square Error (RMSE)

* This is the root of the mean of the squared errors.
* This is the most popular method utilizing the advantage of MSE while also keeping units in check.  
    
  
* Once again, the value of RMSE being good or not is context specific.
* A RMSE of $10 is a fantastic for predicting the price of a house, but horrible for predicting the price of a candy bar.

So, compare the error metric to the average value of the label in the data set to try to get an intuition of its overall performance. Domain knowledge also plays an important role here.

# Machine Learning

## Machine Learning Process



## Process of Using Scikit Learn

Every algorithm is exposed in scikit-learn via an "Estimator"

First, you'll import the model, the general form is:

from sklearn.family import Model

For example:

from sklearn.linear\_model import LinearRegression

### Estimator parameters:

All the parameters of an estimator can be set when it is instantiated, and have suitable default values.

You can use Shift+tab in jupyter to check the possible parameters.

#### For example:

model = LinearRegression(normaIize=True)

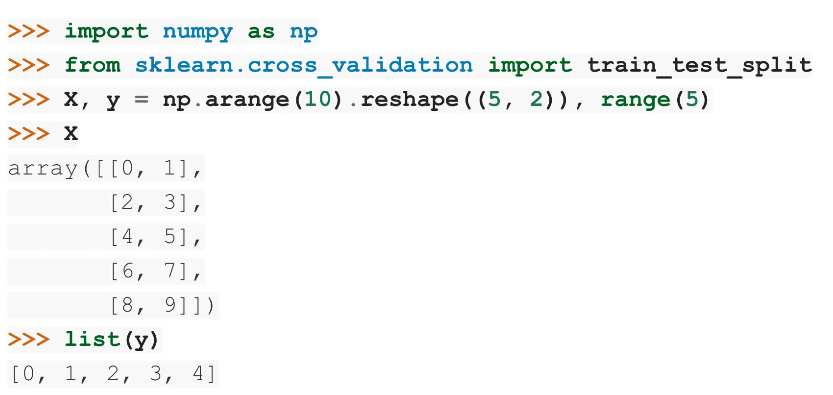
print(model)

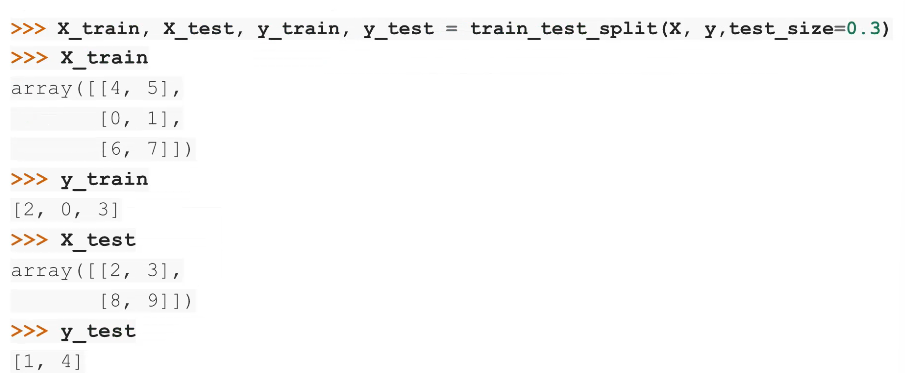
LinearRegression(copy\_X=True, fit \_ intercept-True,

normalize=True)

Once you have your model created with your parameters, it is time to fit your model on some data!

But remember, we should split this data into a training set and a test set.





Now that we have split the data, we can train/fit our model on the training data.

This is done through the model.fit() method:

model.fit(X\_train,y\_train)

* Now the model has been fit and trained on the training data.
* The model is ready to predict labels or values on the test set!

We get predicted values using the predict method:

predictions = model.predict(X\_test)

We can then evaluate our model by comparing our predictions to the correct values.

The evaluation method depends on what sort of machine learning algorithm we are using (e.g., Regression, Classification, Clustering, etc.)

Scikit-learn strives to have a uniform interface across all methods, and we'll see examples of these below.

Given a scikit-learn estimator object named model, the following methods are available...

### Estimators

#### Available in all Estimators:

* model.fit() : fit training data.
* For supervised learning applications, this accepts two arguments: the data X and the labels y (e.g., model.fit(X, y)).
* For unsupervised learning applications, this accepts only a single argument, the data X (e.g., model.fit(X)).

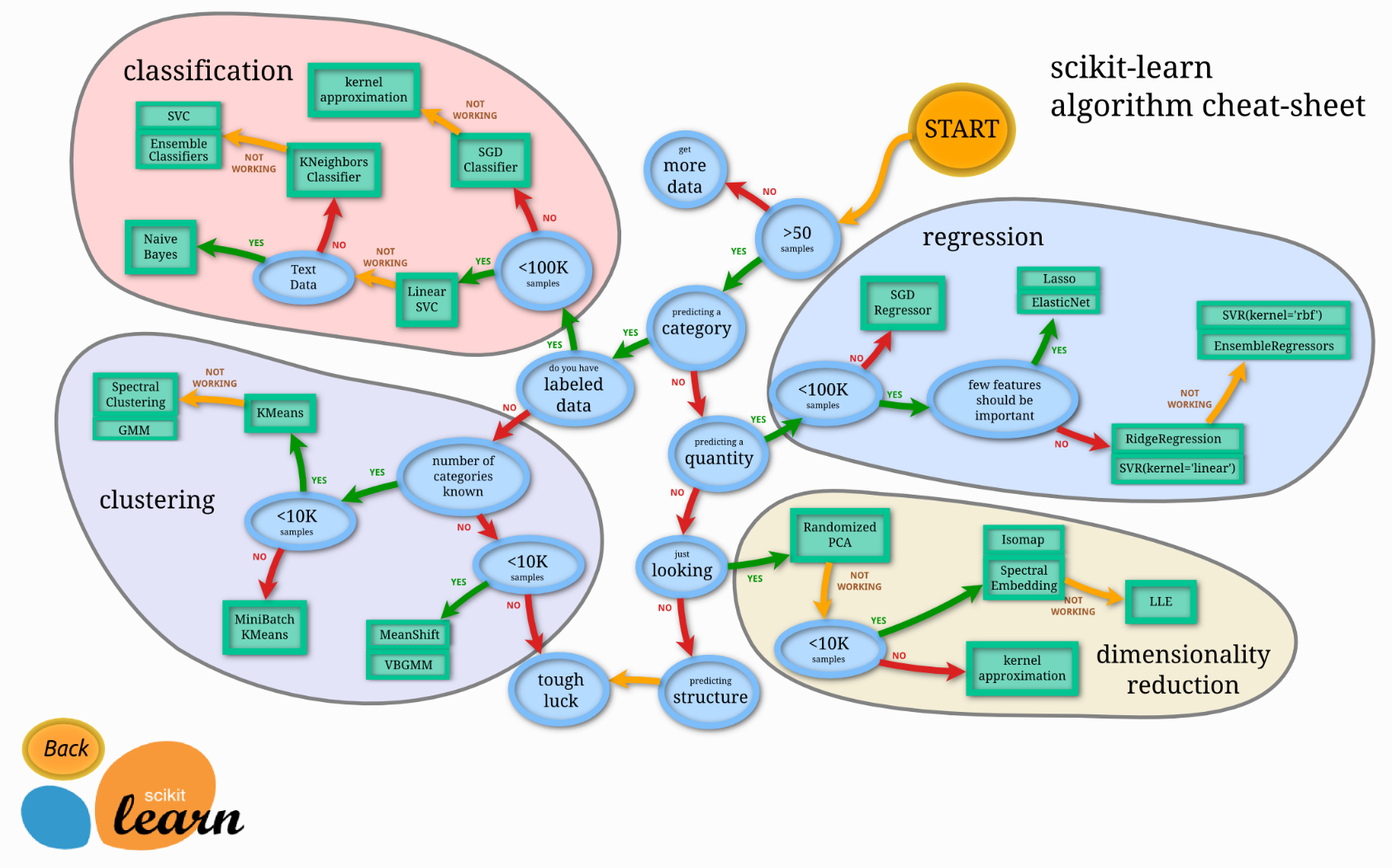
#### Available in Supervised Estimators

* model.predict() : given a trained model, predict the label of a new set of data. This method accepts one argument, the new data X\_new (e.g., model.predict(X\_new)), and returns the learned label for each object in the array.
* model.predict\_proba() : For classification problems, some estimators also provide this method, which returns the probability that a new observation has each categorical label. In this case, the label with the highest probability is returned by model.predict().
* model.score() : for classification or regression problems, most estimators implement a score method. Scores are between 0 and 1, with a larger score indicating a better fit.

#### Available in Unsupervised Estimators

* model.predict() : predict labels in clustering algorithms.
* model.transform() : given an unsupervised model, transform new data into the new basis. This also accepts one argument X\_new, and returns the new representation of the data based on the unsupervised model.
* model.fit\_transform() : some estimators implement this method, which more efficiently performs a fit and a transform on the same input data.

## Choosing an Algorithm



# Linear Regression

## History

This all started in the 1800s with a guy named Francis Galton. Galton was studying the relationship between parents and their children. In particular, he investigated the relationship between the heights of fathers and their sons.

What he discovered that a man’s son tended to be roughly as tall as his father.

However, Galton’s breakthrough was that the son’s height tended to be closer to the overall average height of all people.

Let's take Shaquille O'Neal as an example. Shaq is really tall:7ft 1in (2.2 meters).

If Shaq has a son, chances are he'll be pretty tall too. However, Shaq is such an anomaly that there is also a very good chance that his son will be not be as tall as Shaq.

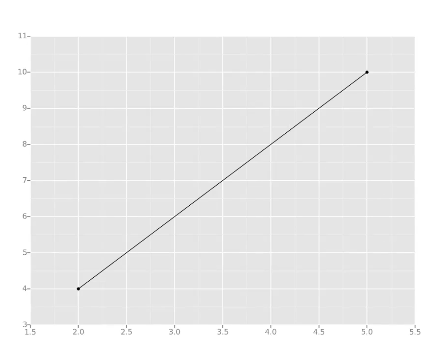
Turns out this is the case:

Shaq's son is pretty tall (6 ft 7 in), but not nearly as tall as his dad.

Galton called this phenomenon regression, as in "A father's son's height tends to regress (or drift towards) the mean (average) height."

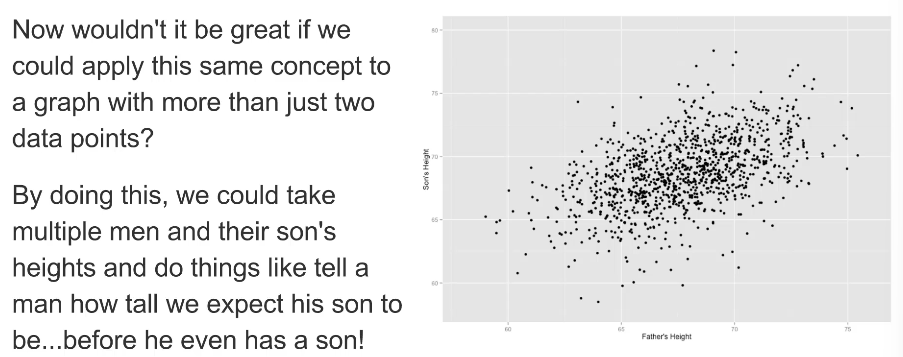
## Let's take the simplest possible example:

Calculating a regression with only 2 data points.

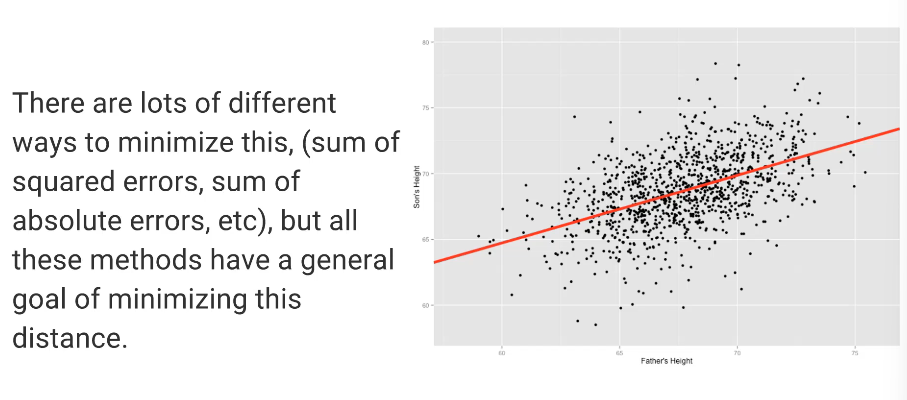


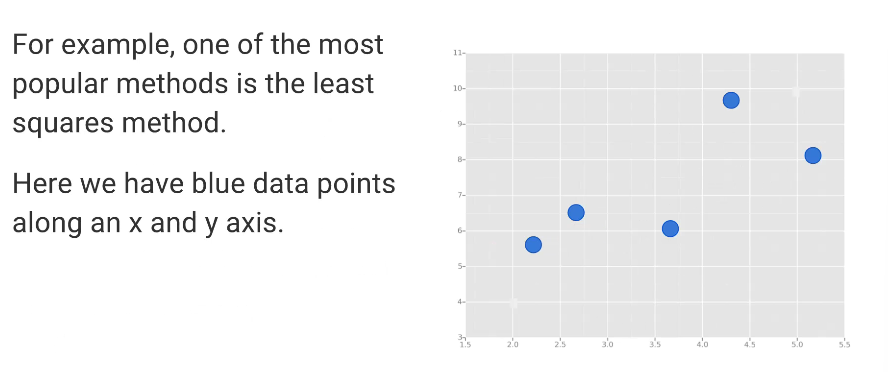
All we're trying to do when we calculate our regression line is draw a line that's as close to every dot as possible.

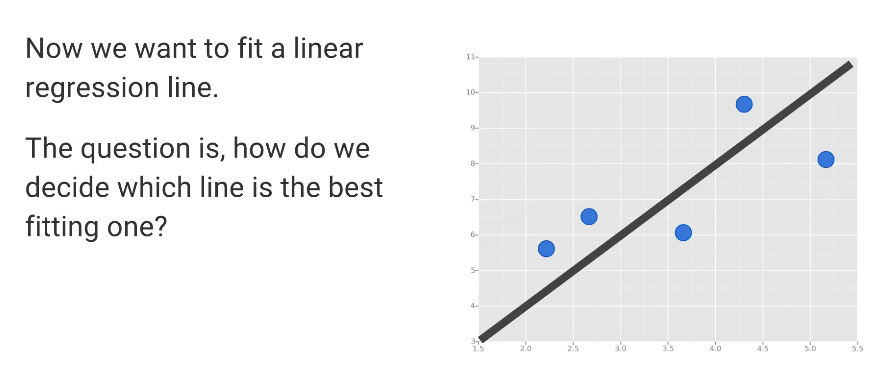
For classic linear regression, or "Least Squares Method", you only measure the closeness in the "up and down" direction

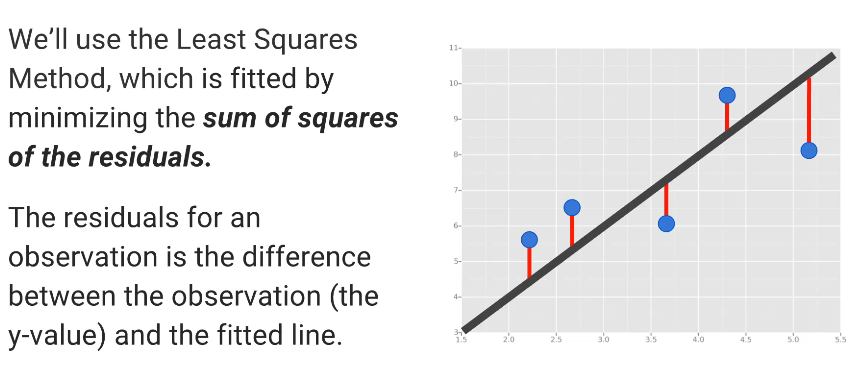








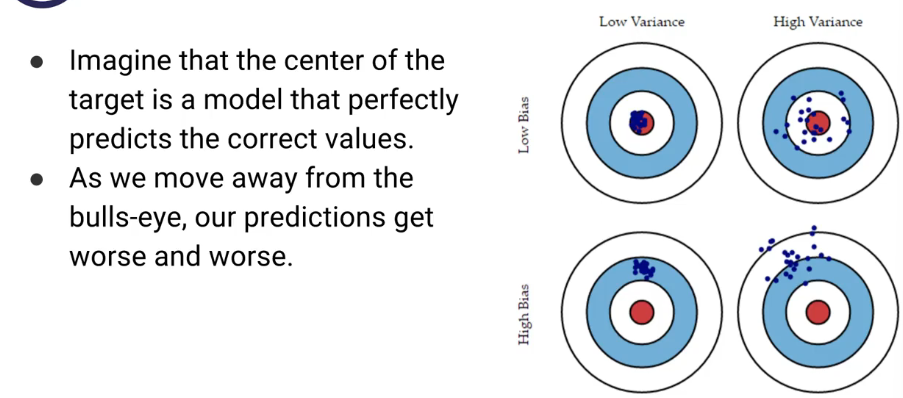


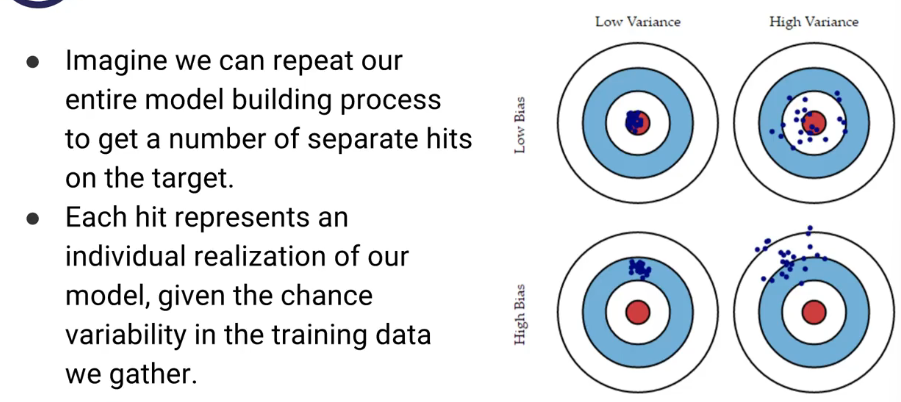


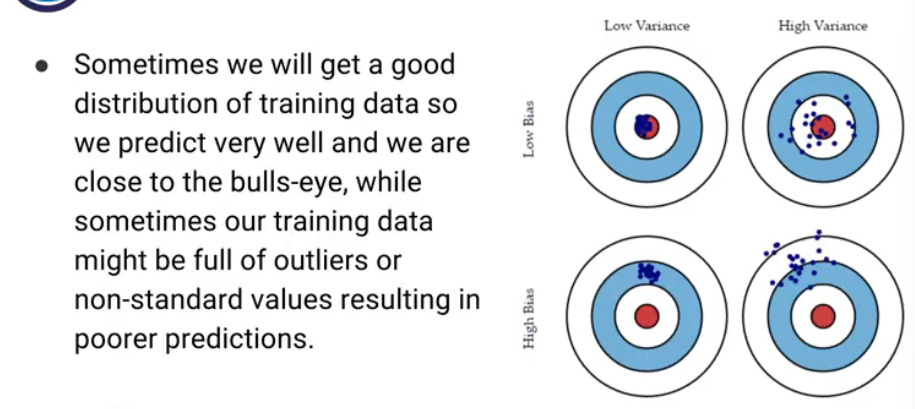
# Cross Validation and Bias-Variance Trade-Off

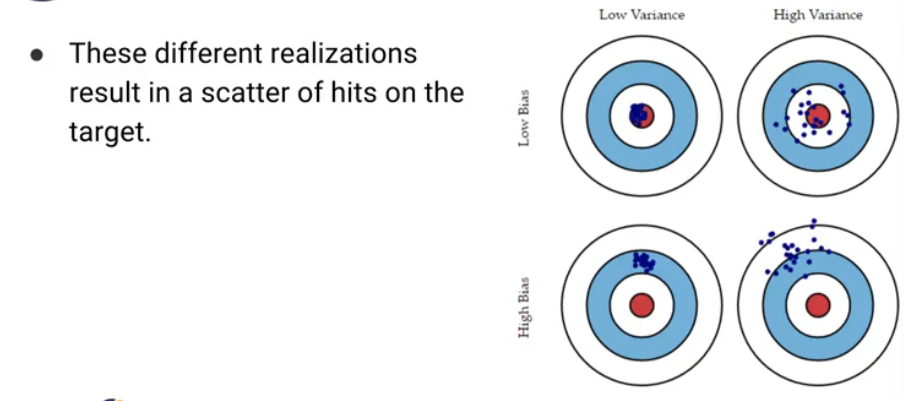
## Bias Variance Trade-Off

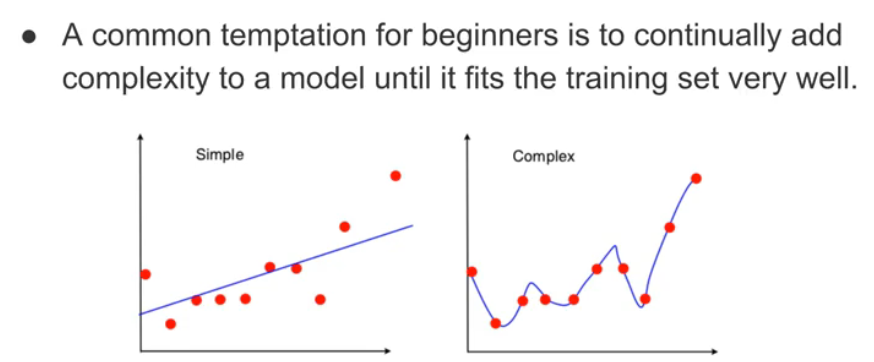
* The bias-variance trade-off is the point where we are adding just noise by adding model complexity (flexibility).
* The training error goes down as it has to, but the test error is starting to go up.
* The model after the bias trade-off begins to overfit.

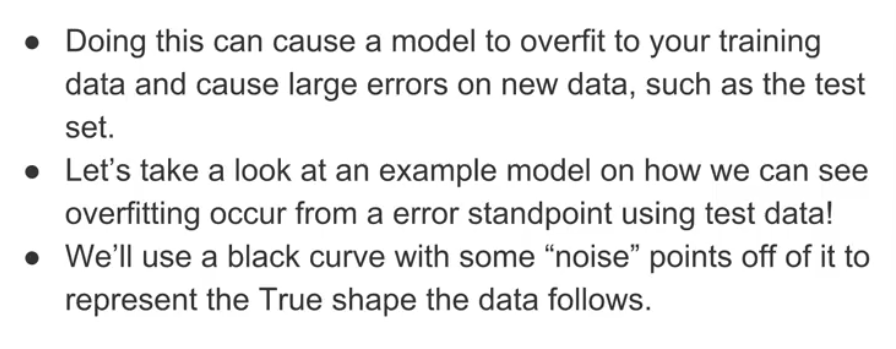


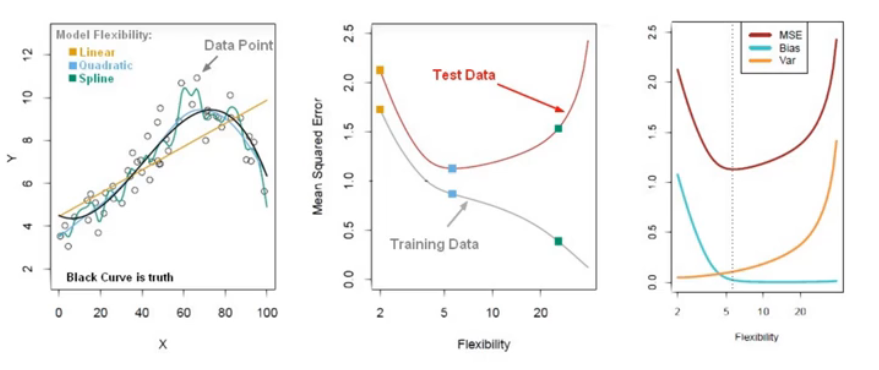


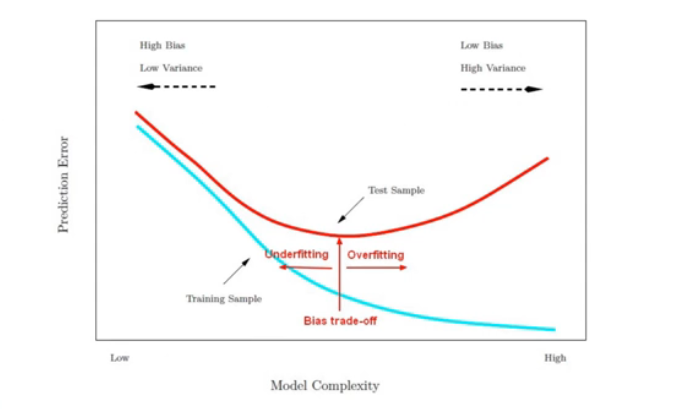




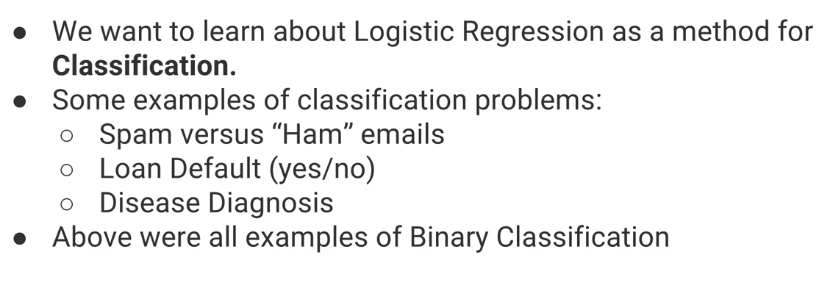


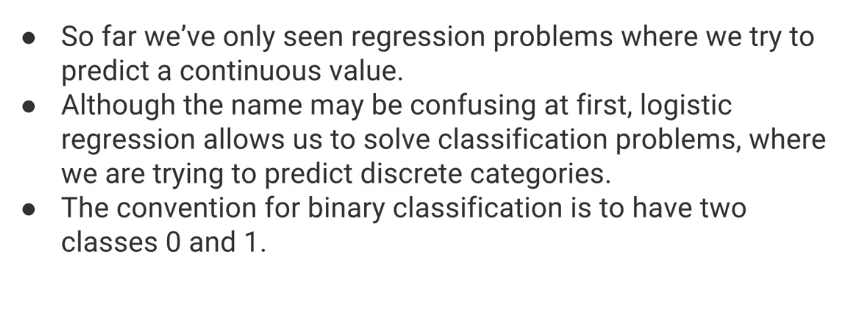


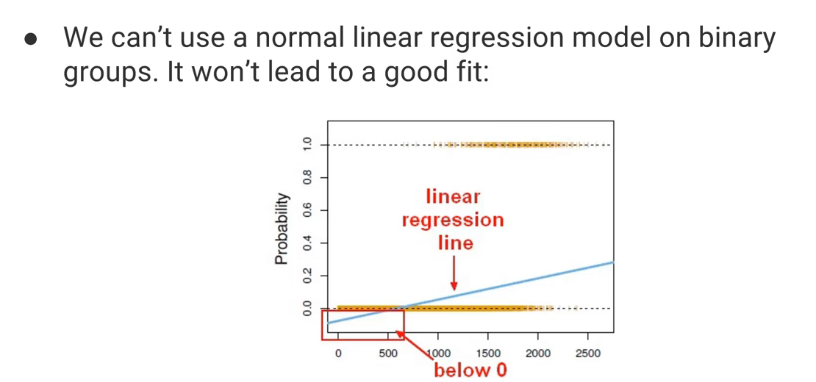


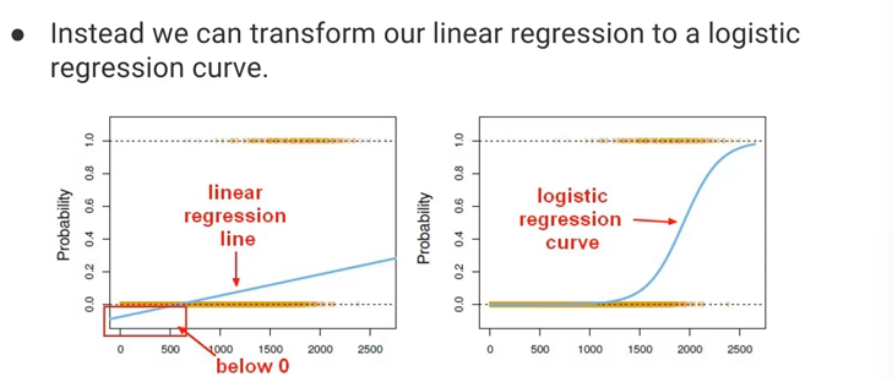


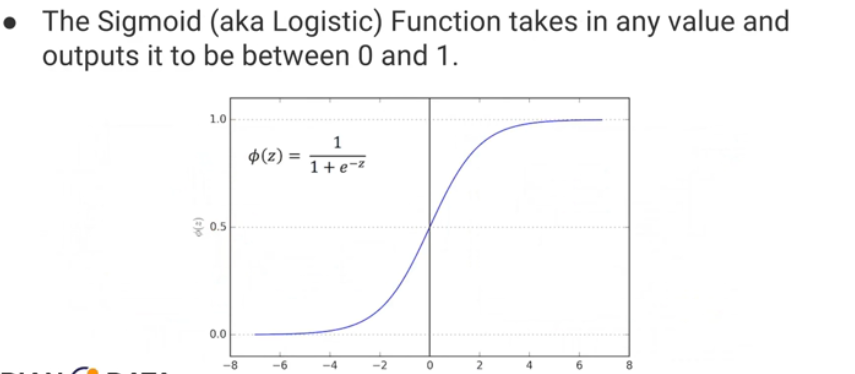
# Logistic Regression

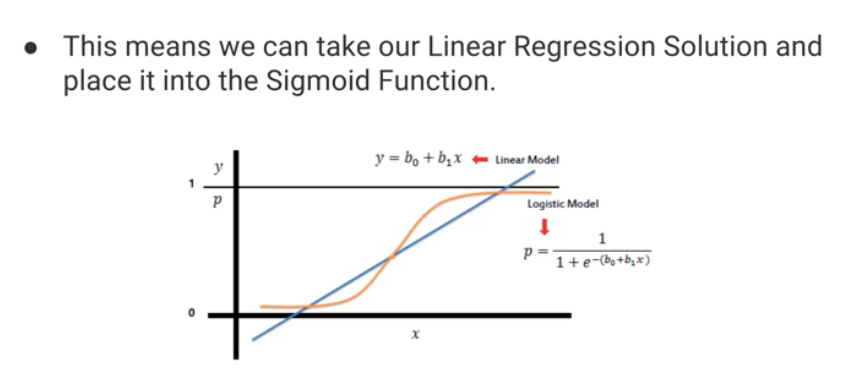


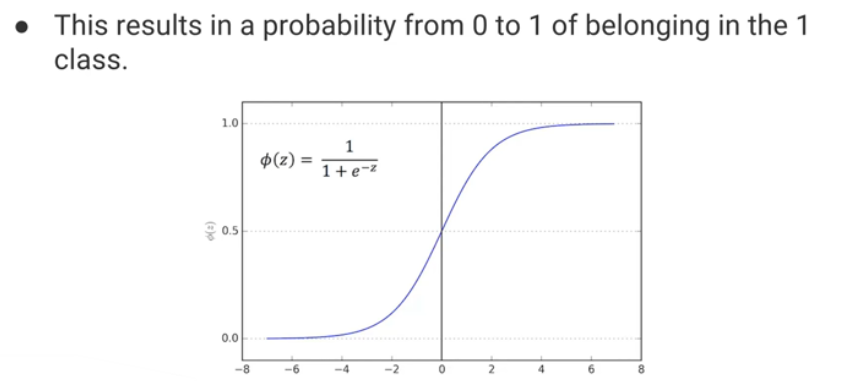


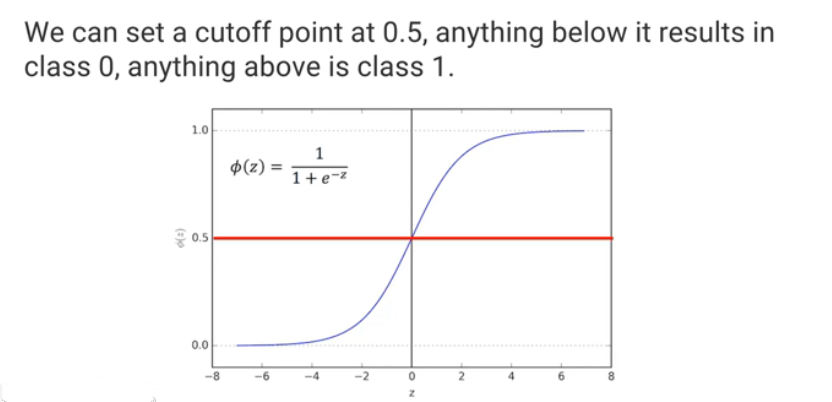


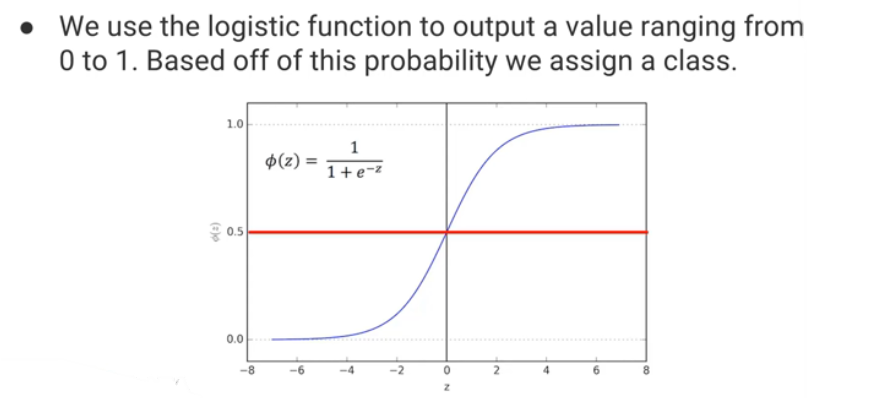


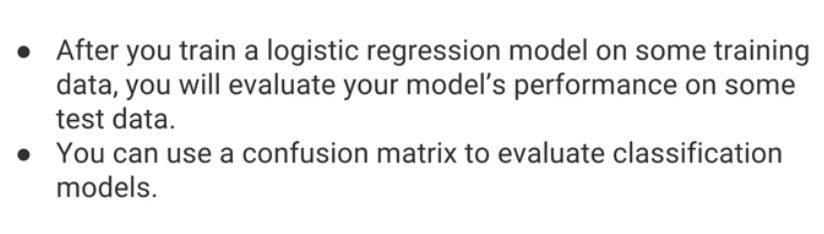


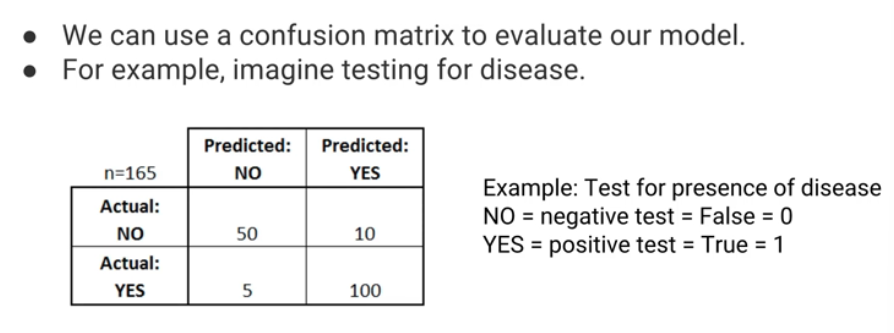


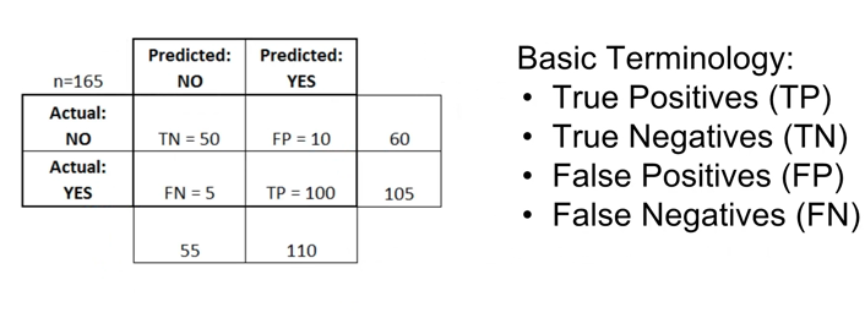


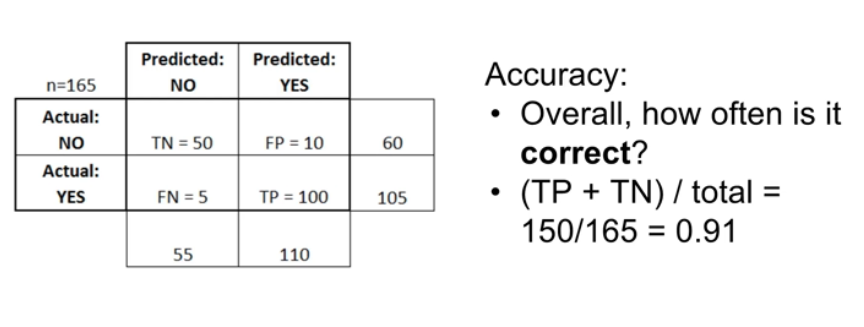


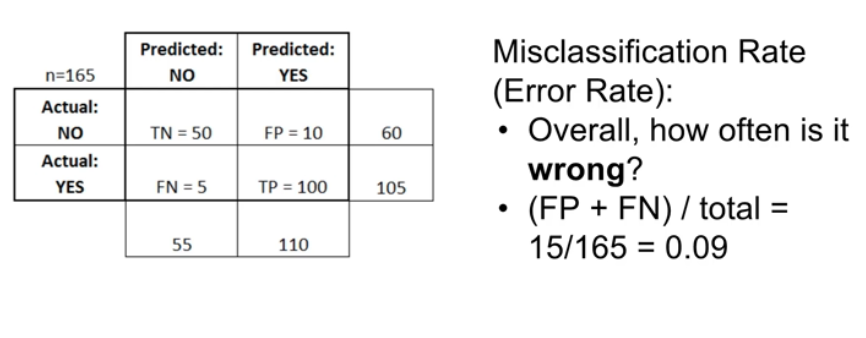


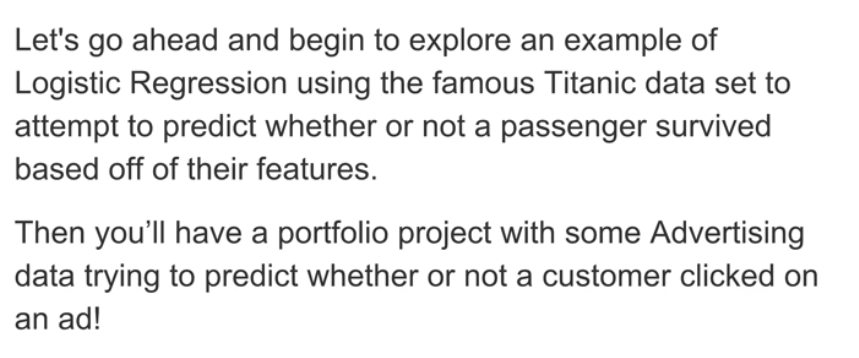








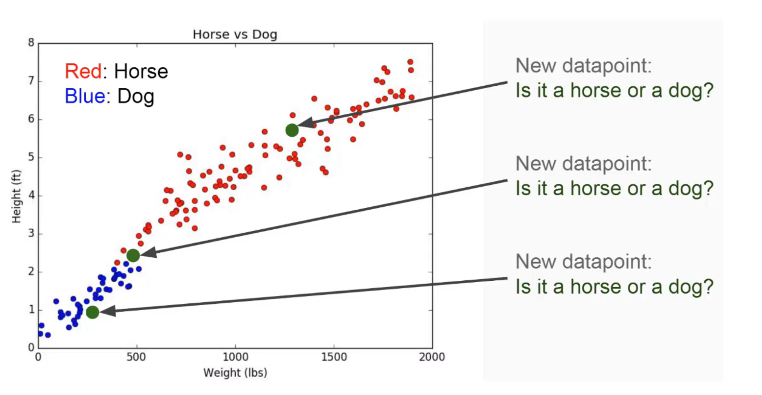




# K Nearest Neighbors (KNN)

K Nearest Neighbors is a classification algorithm that operates on a very simple principle. It is best shown through example!

Imagine we had some imaginary data on Dogs and Horses, with heights and weights.



## Training Algorithm:

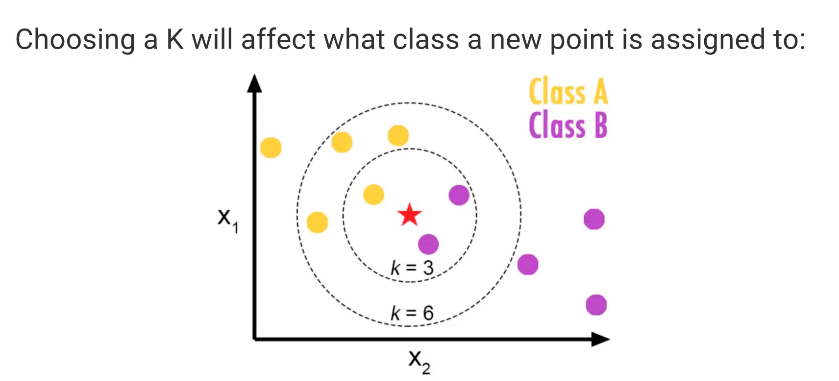
1. Store all the Data

## Prediction Algorithm:

1. Calculate the distance from x to all points in your data

2. Sort the points in your data by increasing distance from x

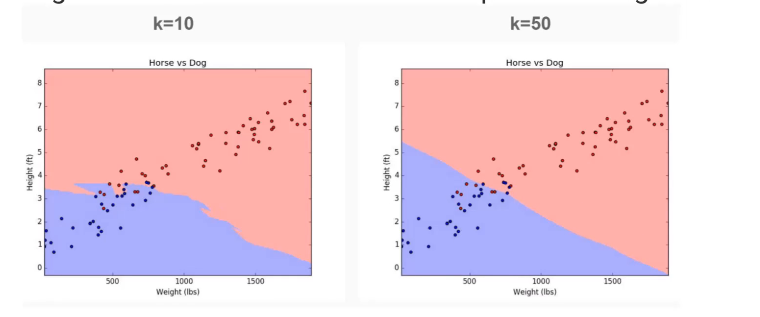
3. Predict the majority label of the "k" closest points



If K=3, the class assigned will be class B as out of the 3 neighboring points, 2 are class B and 1 is class A.

However, if K=6, the class assigned will be class A as out of the 6 neighboring points, 2 are class B and 4 are class A.





## Pros:

* Very simple
* Training is trivial
* Works with any number of classes
* Easy to add more data
* Few parameters

1. K
2. Distance Metric

## Cons:

• High Prediction Cost (worse for large data sets)

• Not good with high dimensional data

• Categorical Features don't work well

A common interview task for a data scientist position is to be given anonymized data and attempt to classify it, without knowing the context of the data.

We're going to simulate a similar scenario by giving you some "classified" data, where what the columns represent is not known, but you have to use KNN to classify it!