

# Machine Learning Overview

# After this video you will be able to..

- Explain what machine learning is
- List three applications of machine learning encountered in everyday life

# What is Machine Learning?



# Machine Learning is...

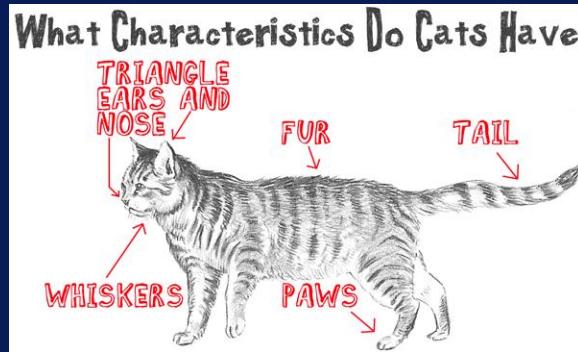
...learning from data



# Machine Learning is...

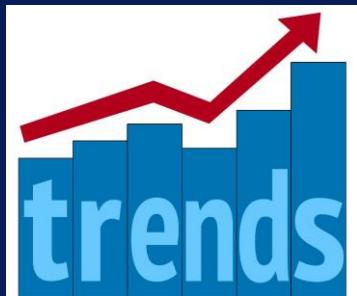
... learning from data

... no explicit programming



# Machine Learning is...

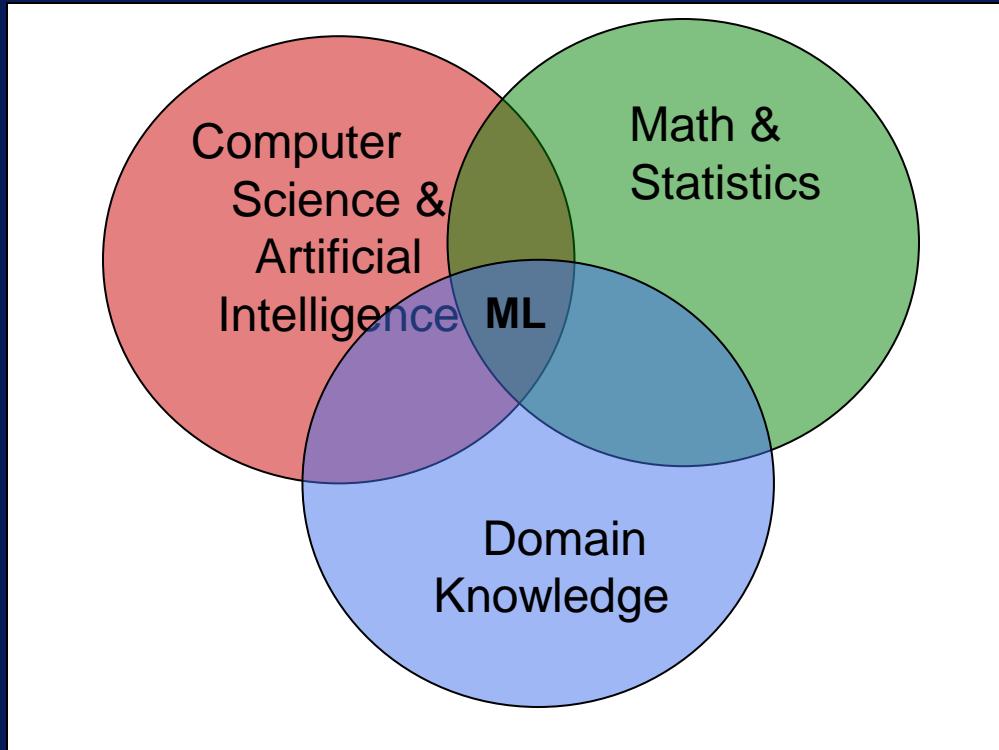
- ... learning from data
- ... no explicit programming
- ... discovering hidden patterns



# Machine Learning is...

- ... learning from data
- ... no explicit programming
- ... discovering hidden patterns
- ... data-driven decisions

# Machine Learning (ML) is an Interdisciplinary Field



# Example Application of Machine Learning

- Credit card fraud detection



# Example Application of Machine Learning

- Handwritten digit recognition

4	→ 4	2	→ 2	3	→ 3
4	→ 4	9	→ 9	0	→ 0
5	→ 5	7	→ 7	1	→ 1
9	→ 9	0	→ 0	3	→ 3
6	→ 6	7	→ 7	4	→ 4

# Example Application of Machine Learning

- Recommendations on websites



# More Applications of Machine Learning

- Targeted ads on mobile apps
- Sentiment analysis
- Climate monitoring
- Crime pattern detection
- Drug effectiveness analysis

# What's in a Name?

*Machine learning*

*Data mining*

*Predictive analytics*

*Data science*

# Machine Learning Models

- Learn from data
- Discover patterns and trends
- Allow for data-driven decisions
- Used in many different applications



# Categories of Machine Learning Techniques

# After this video you will be able to..

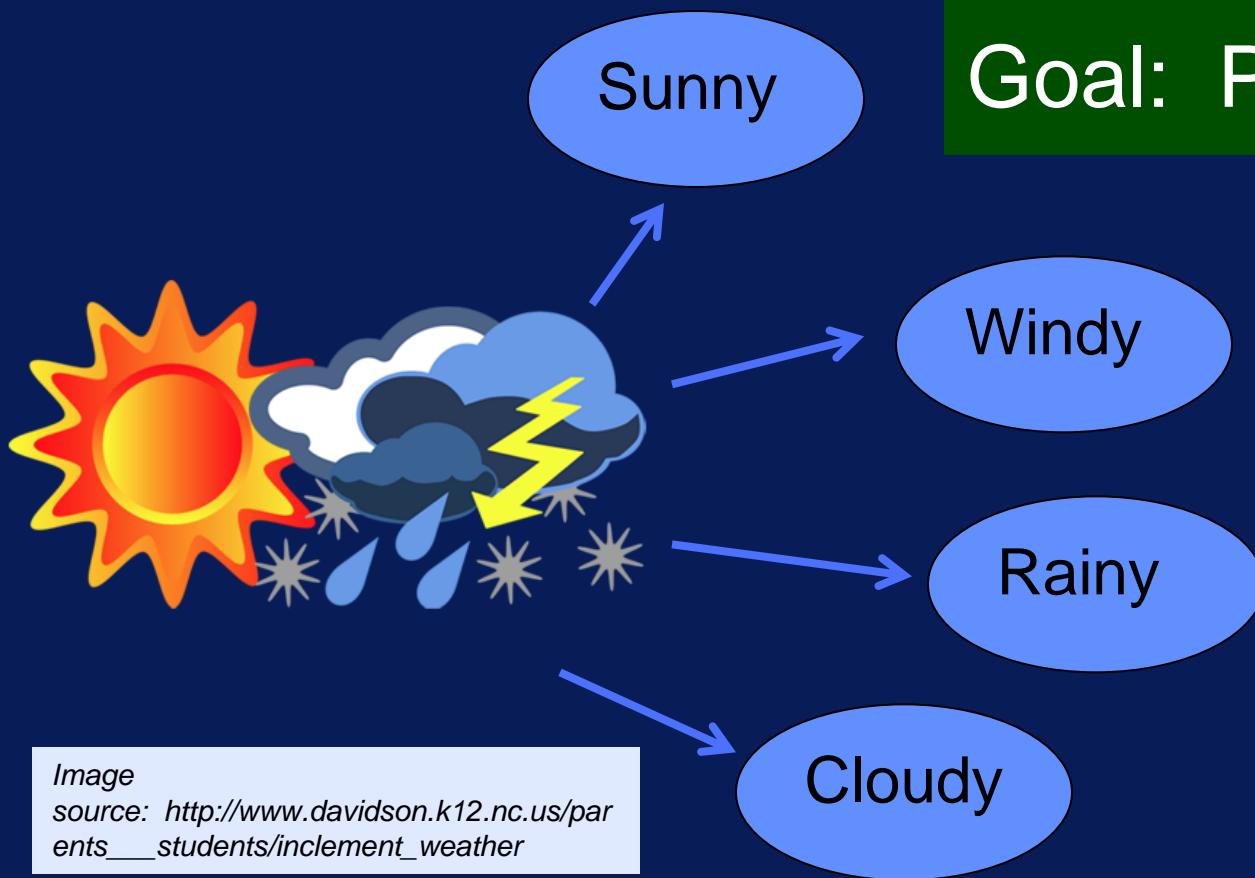
- Describe the main categories of machine learning techniques
- Summarize how supervised learning differs from unsupervised learning

# Categories of Machine Learning Techniques

- Classification
- Regression
- Cluster Analysis
- Association Analysis

# Classification

Goal: Predict category



Image

source: [http://www.davidson.k12.nc.us/parents\\_students/inclement\\_weather](http://www.davidson.k12.nc.us/parents_students/inclement_weather)

# Classification Examples

- Classify tumor as benign or malignant
- Predict if it will rain tomorrow
- Determine if loan application is high-, medium-, or low-risk
- Identify sentiment as positive, negative, or neutral

# Regression

Goal: Predict numeric value



# Regression Examples

- Estimate demand for a product based on time of year
- Predict score on a test
- Determine likelihood of drug effectiveness for patient
- Predict amount of rain

# Cluster Analysis

Goal: Organize similar items into groups.

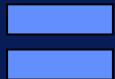


# Cluster Analysis Examples

- Identify areas of similar topography (desert, grass, etc.)
- Categorize different types of tissues from medical images
- Determine different groups of weather patterns
- Discover crime hot spots

*Presenter*

# Association Analysis



Goal: Find rules to capture associations between items.

# Association Analysis Examples

- Recommend items based on purchase/browsing history
- Have sales on related items often purchased together
- Identify web pages accessed together

# Categories of Machine Learning Techniques

Classification

Cluster  
Analysis

Regression

Association  
Analysis

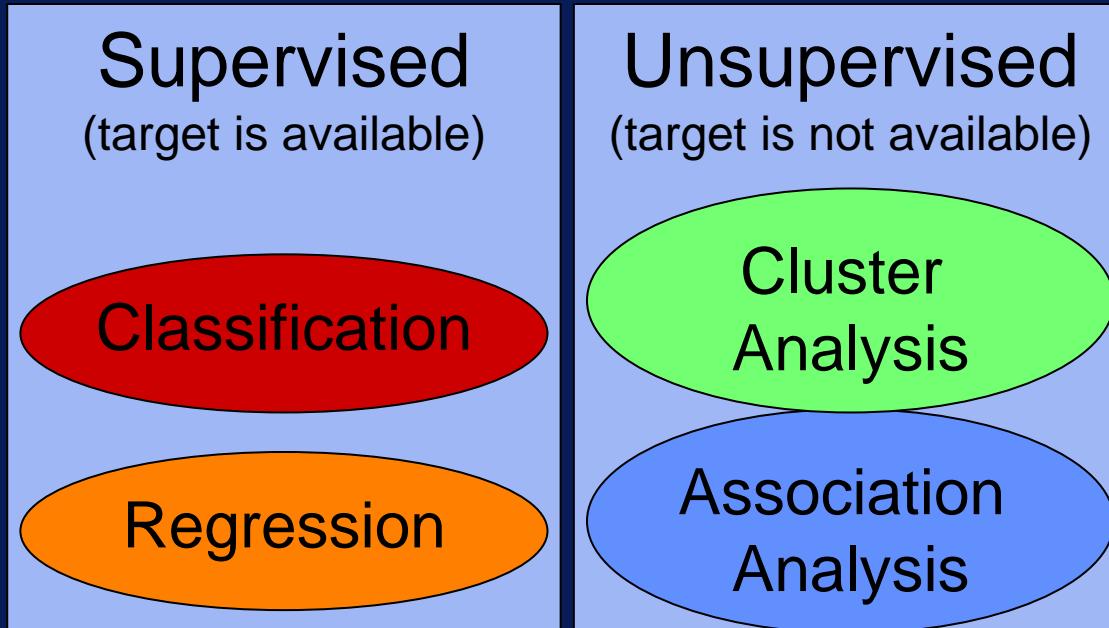
# Supervised vs. Unsupervised

- **Supervised Approaches**
  - Target (what model is predicting) is provided
  - ‘Labeled’ data
  - Classification & regression are supervised.

# Supervised vs. Unsupervised

- **Unsupervised Approaches**
  - Target is unknown or unavailable
  - ‘unlabeled’ data
  - Cluster analysis & association analysis are unsupervised.

# Categories of Machine Learning Techniques



# Machine Learning Process

# After this video you will be able to..

- Identify the steps in the machine learning process
- Discuss why the machine learning process is iterative

ACQUIRE

PREPARE

ANALYZE

REPORT

ACT

PURPOSE

ACQUIRE

PREPARE

ANALYZE

REPORT

ACT

# Step 1: Acquire Data



Identify data sources

Collect data

Integrate data

ACQUIRE

**PREPARE**

ANALYZE

REPORT

ACT

## Step 2: Prepare Data

Step 2-A: Explore

Step 2-B: Pre-process



ACQUIRE

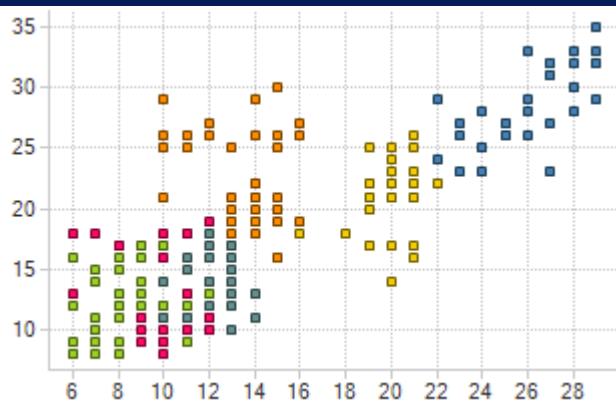
PREPARE

ANALYZE

REPORT

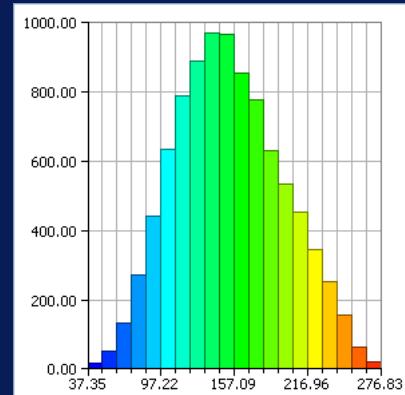
ACT

## Step 2-A: Explore Data



Understand  
nature of data

Preliminary  
analysis



ACQUIRE

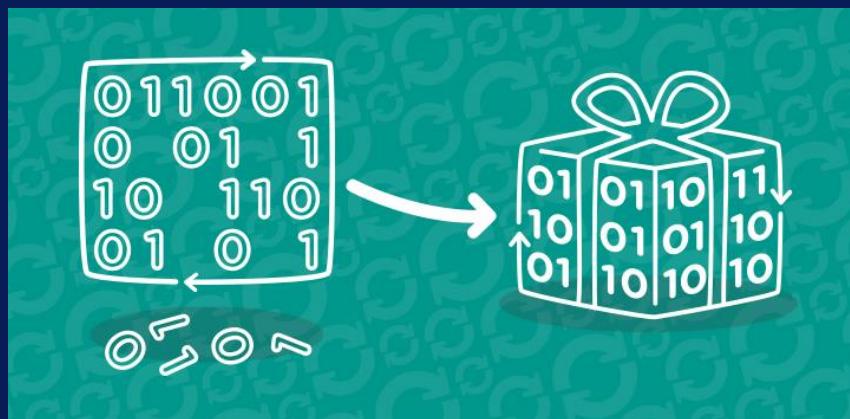
**PREPARE**

ANALYZE

REPORT

ACT

## Step 2-B: Pre-process Data



Clean

Select

Transform

ACQUIRE

PREPARE

**ANALYZE**

REPORT

ACT

# Step 3: Analyze Data



Select analytical techniques

Build models

Assess results

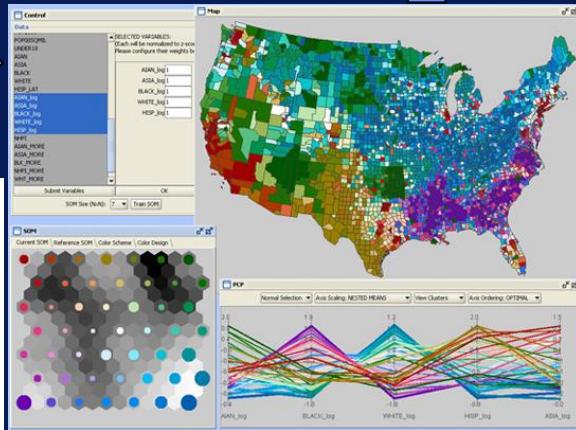
# ACQUIRE

# PREPARE

# REPORT

ACT

# Step 4: Communicate Results



ACQUIRE

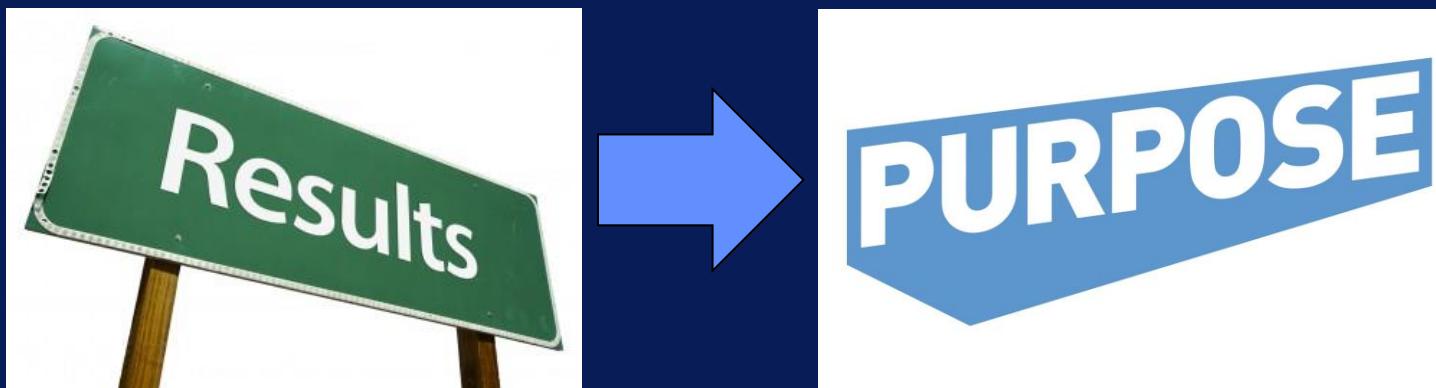
PREPARE

ANALYZE

REPORT

ACT

## Step 5: Apply Results



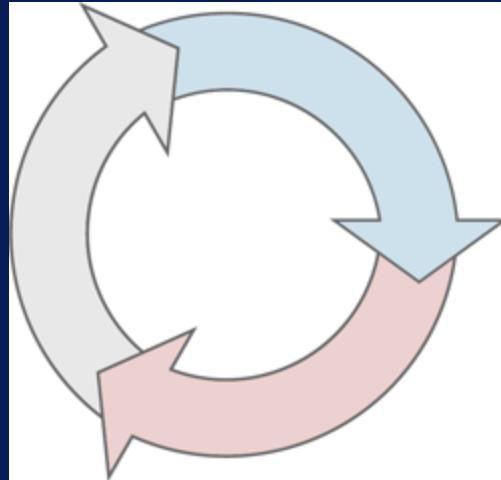
ACQUIRE

PREPARE

ANALYZE

REPORT

ACT



**Iterative process**

# Goals and Activities in the Machine Learning Process

# After this video you will be able to...

- Explain the goals of each step in the machine learning process
- List key activities in each step in the process

ACQUIRE

PREPARE

ANALYZE

REPORT

ACT



ACQUIRE

PREPARE

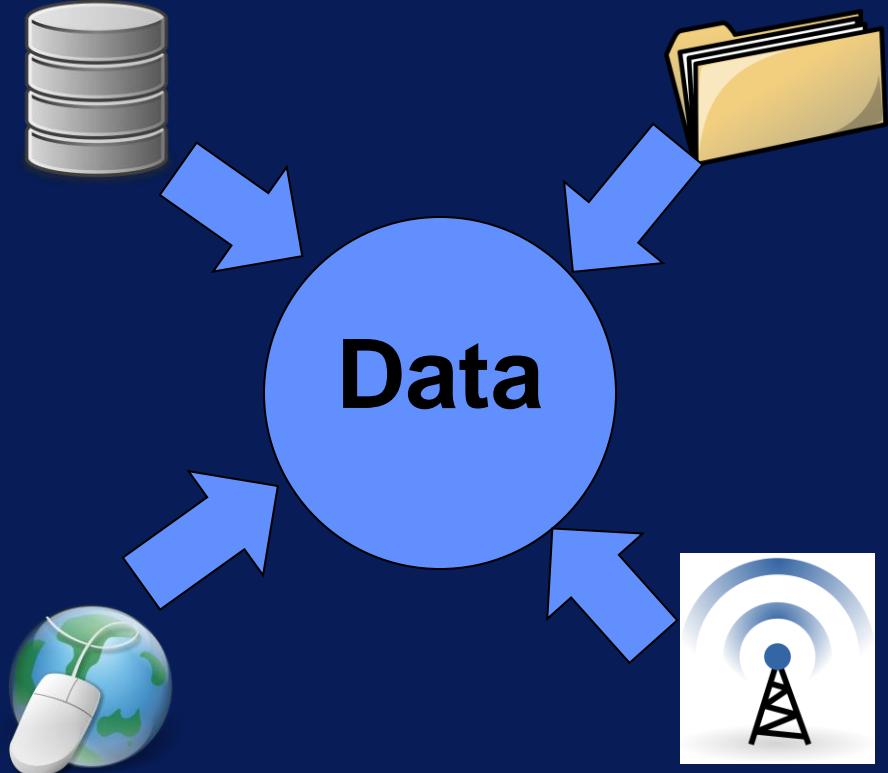
ANALYZE

REPORT

ACT

Goal: Identify and obtain all data  
related to problem

# Acquire Data



Identify data sources  
Collect data  
Integrate data

ACQUIRE

**PREPARE**

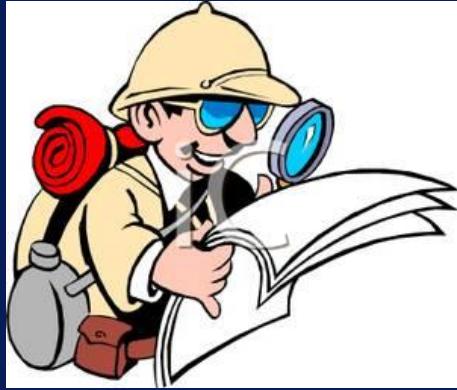
ANALYZE

REPORT

ACT

Step 2-A: Explore

Step 2-B: Pre-process



# Why Explore?

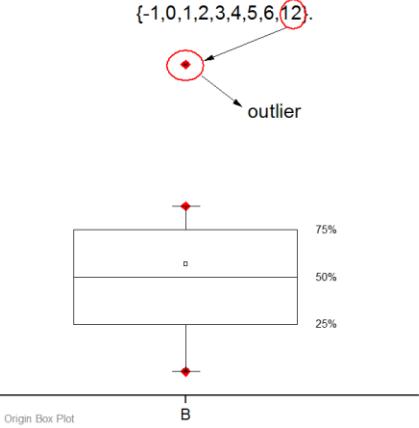
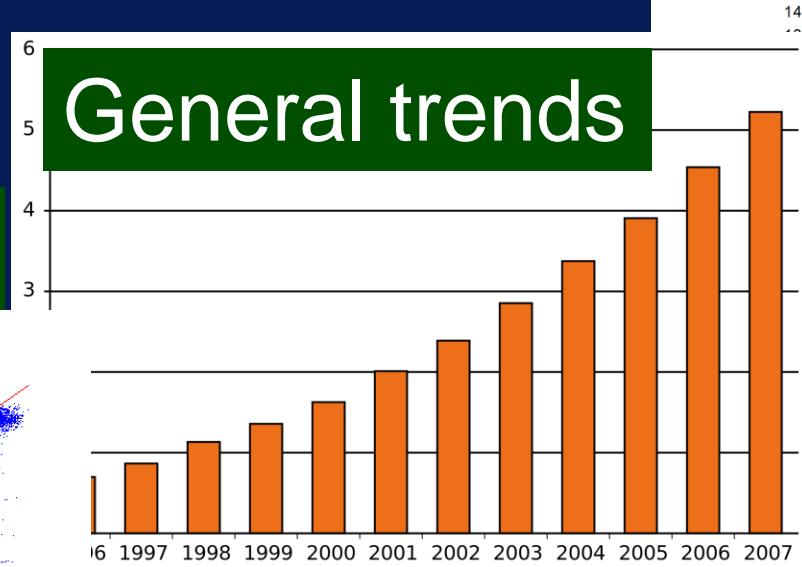
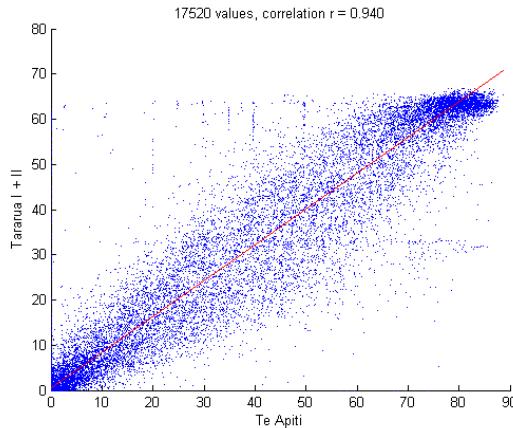
Goal: Understand your data



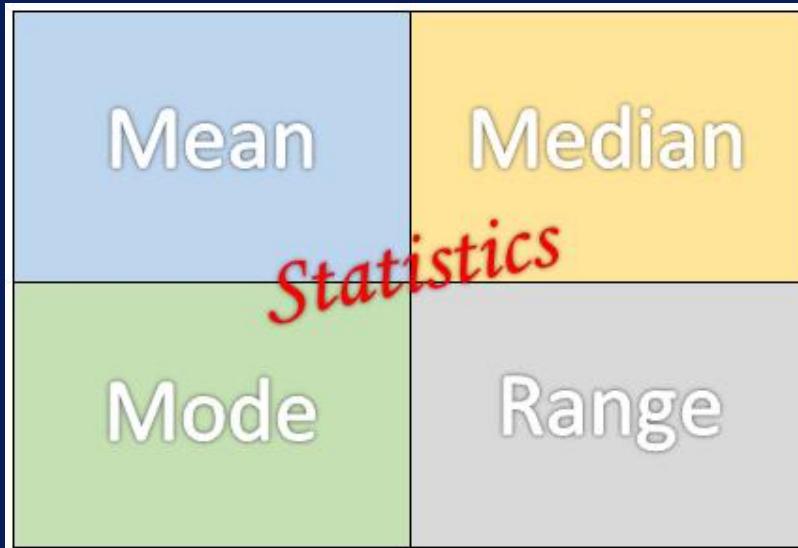
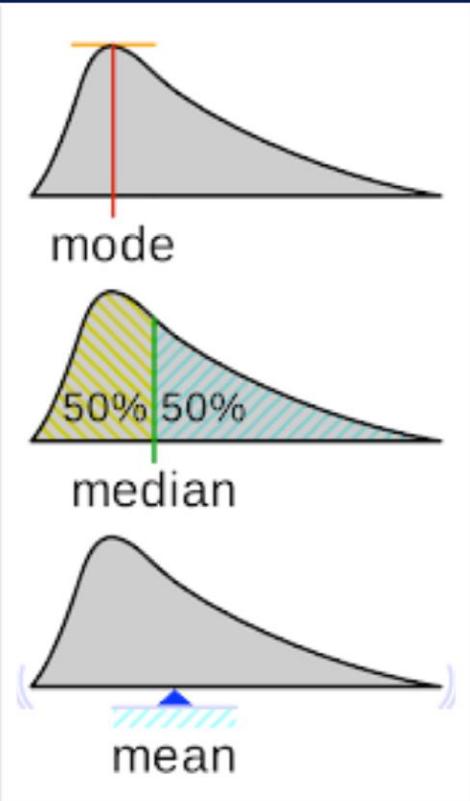
# Why Explore?

Outliers

Correlations

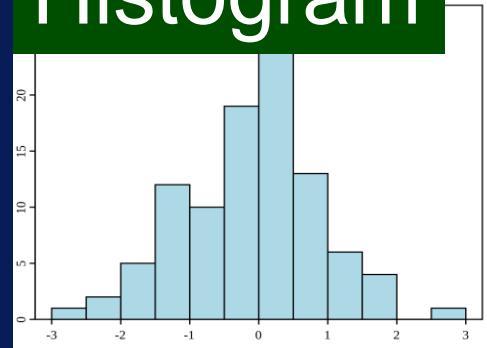


# Describe Your Data

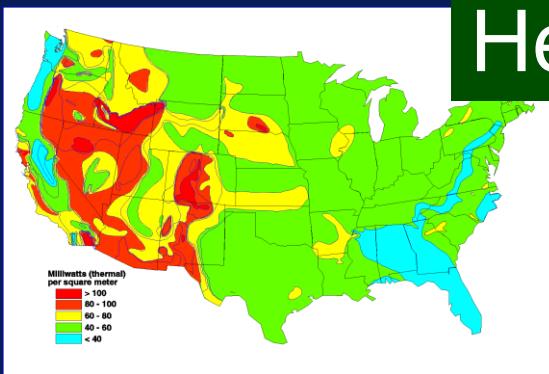


# Visualize Your Data

## Histogram



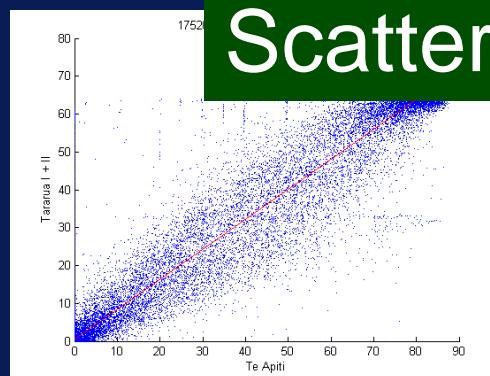
## Heat map



## Line plot



## Scatter plot



ACQUIRE

**PREPARE**

ANALYZE

REPORT

ACT

Step 2-A: Explore

Step 2-B: Pre-process

ACQUIRE

**PREPARE**

ANALYZE

REPORT

ACT

Step 2-A: Explore

Goal: Create data  
for analysis

Step 2-B: Pre-process

Clean

Select

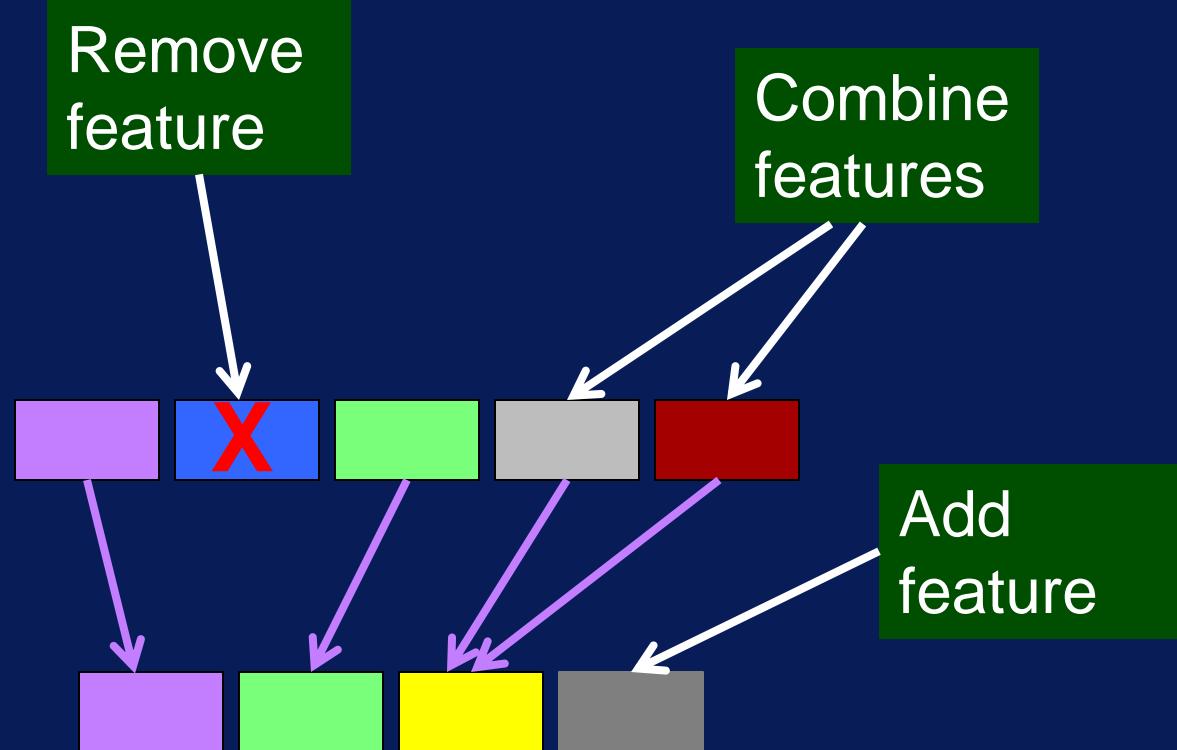
Transform

# Data Cleaning

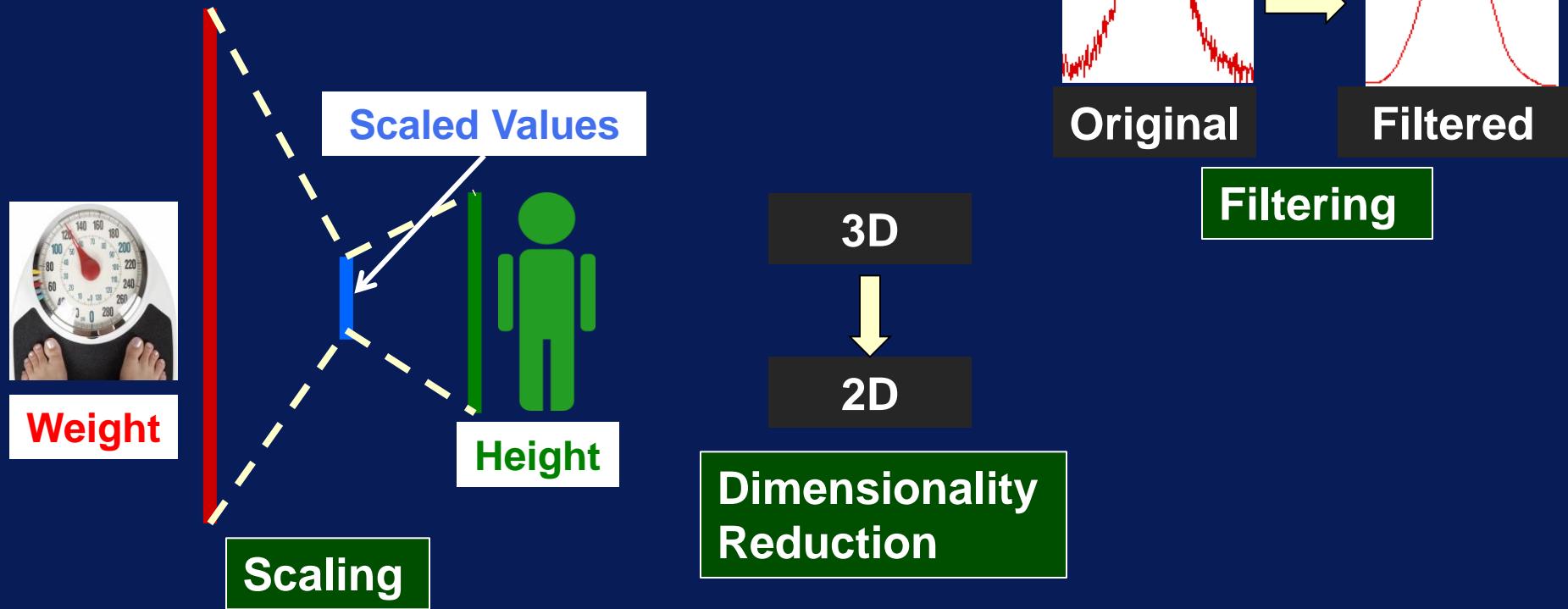
- Missing values
- Duplicate data
- Inconsistent data
- Noise
- Outliers



# Feature Selection



# Feature Transformation





ACQUIRE

PREPARE

**ANALYZE**

REPORT

ACT

## Goals:

- Build model
- Evaluate results

# Analyze

Select technique

Build model

Evaluate

Classification  
Regression  
Cluster Analysis  
Association  
Analysis



# Analyze

Select technique



Build model



Evaluate results



ACQUIRE

PREPARE

ANALYZE

REPORT

ACT

Goal: Communicate results and recommend actions

# Present

# Report



## with



## using

+ + + + + a b | e a u





ACQUIRE

PREPARE

ANALYZE

REPORT

ACT

Goal: Determine actions based on insights

# Act

Determine action



Implement



Assess impact

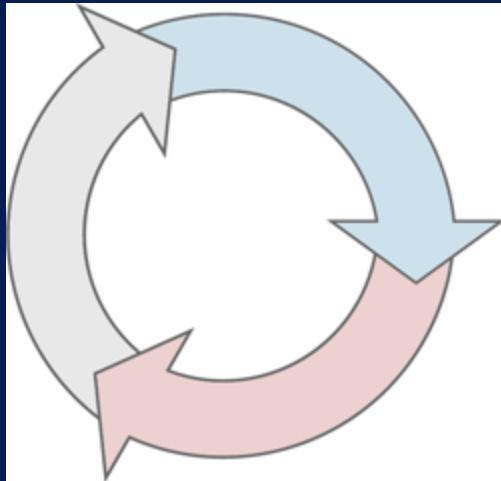
ACQUIRE

PREPARE

ANALYZE

REPORT

ACT



**Iterative process**

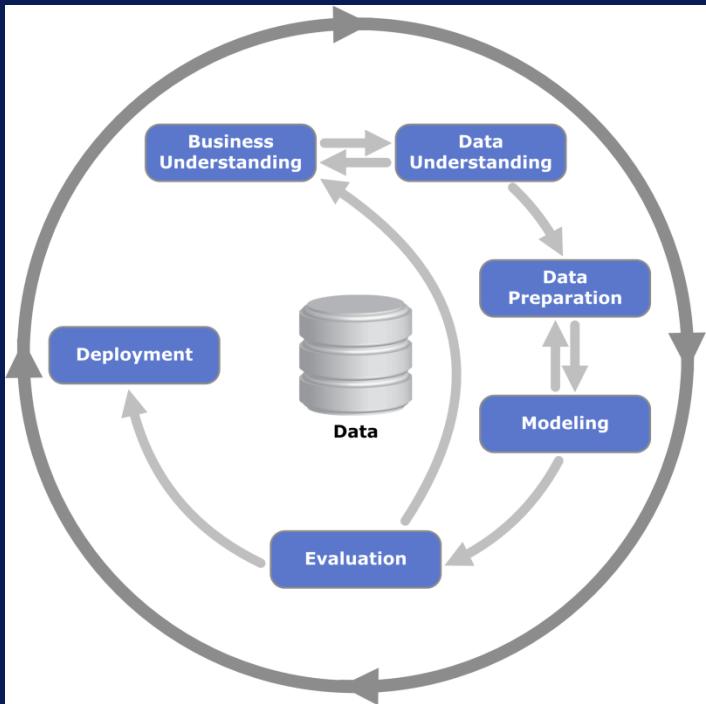
# Cross Industry Standard Process for Data Mining (CRISP-DM)

# After this video you will be able to..

- Summarize what CRISP-DM is
- List the phases of CRISP-DM
- Describe the goals of each phase

# CRISP-DM

# CRoss Industry Standard Process for Data Mining



# CRISP-DM Phases

- Business Understanding
- Data Understanding
- Data Preparation
- Modeling
- Evaluation
- Deployment

# Phase 1 – Business Understanding

- Define problem or opportunity
- Assess situation
- Formulate goals



# Phase 2 – Data Understanding

- **Data acquisition**
- **Data exploration**



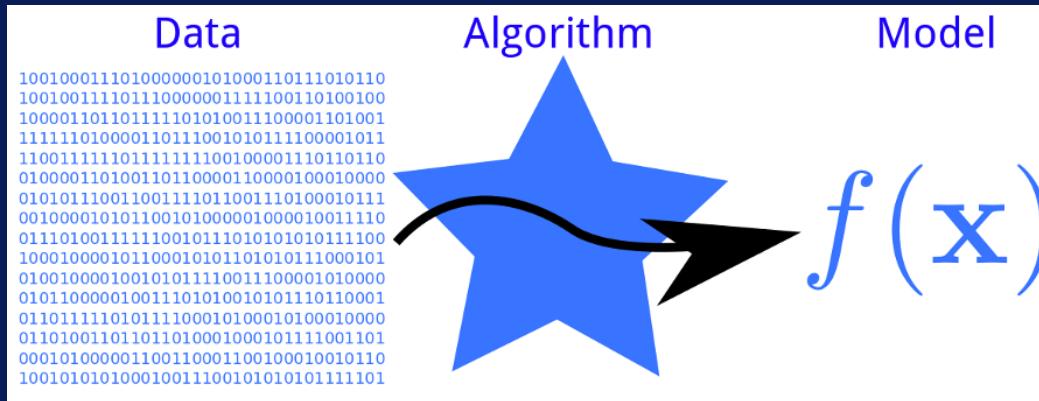
# Phase 3 – Data Preparation

- Prepare data for modeling
- Address quality issues, select features to use, process data for modeling



# Phase 4 – Modeling

- Determine type of problem
- Select modeling technique(s) to use
- Build model



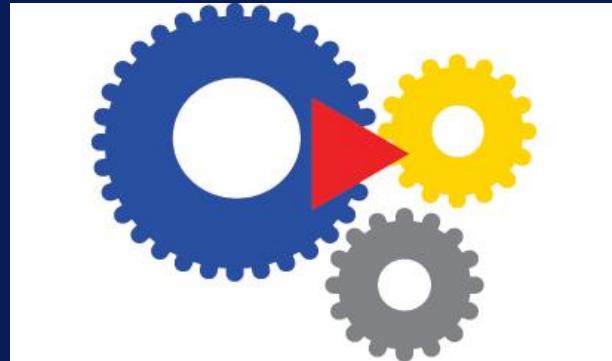
# Phase 5 – Evaluation

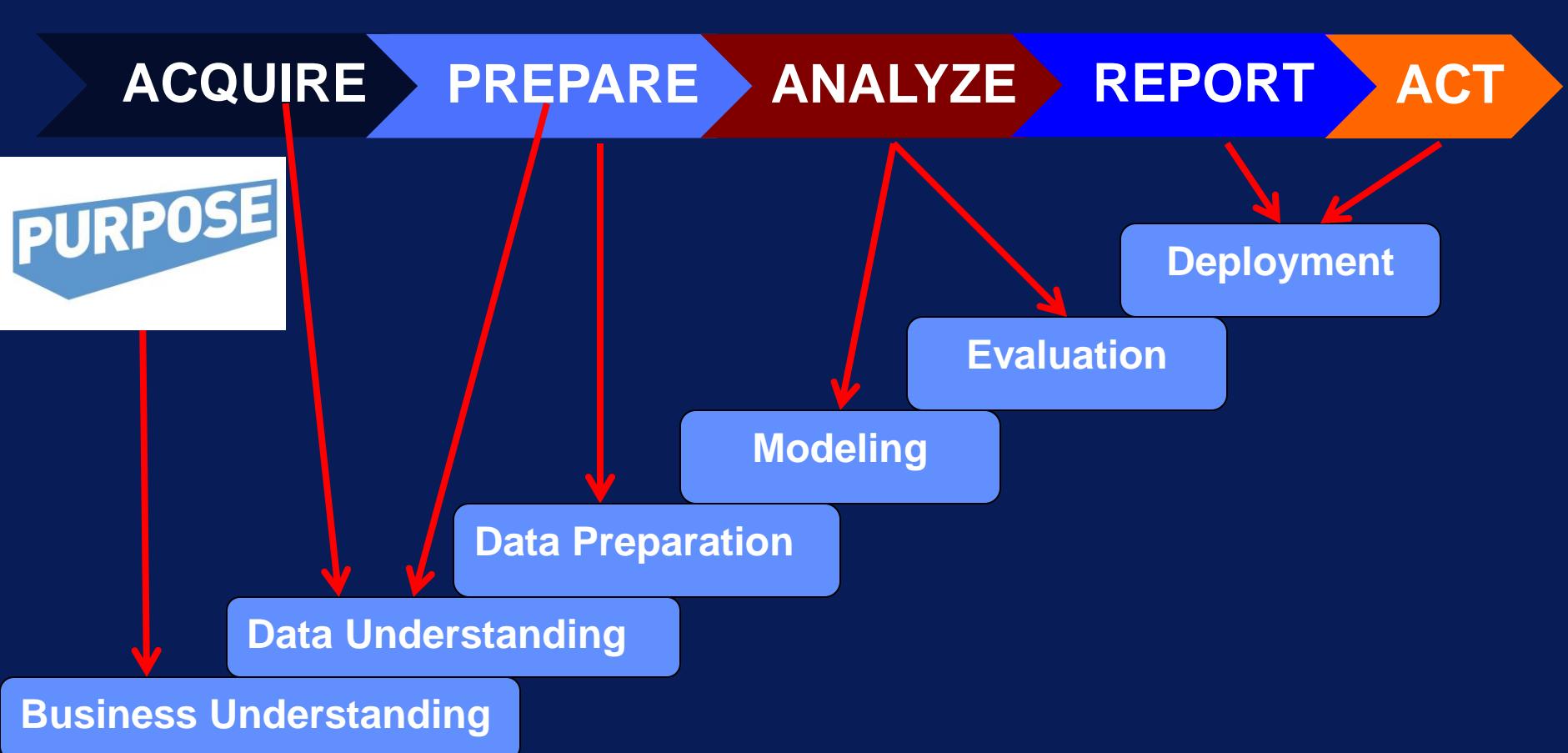
- Assess model performance
- Evaluate model results with respect to success criteria



# Phase 6 – Deployment

- Produce final report
- Deploy model
- Monitor model





# Tools Used in This Course

# After this video you will be able to..

- Describe what KNIME is
- Describe what Spark MLlib is
- Contrast KNIME and ML as machine learning tools

# Tools for This Course

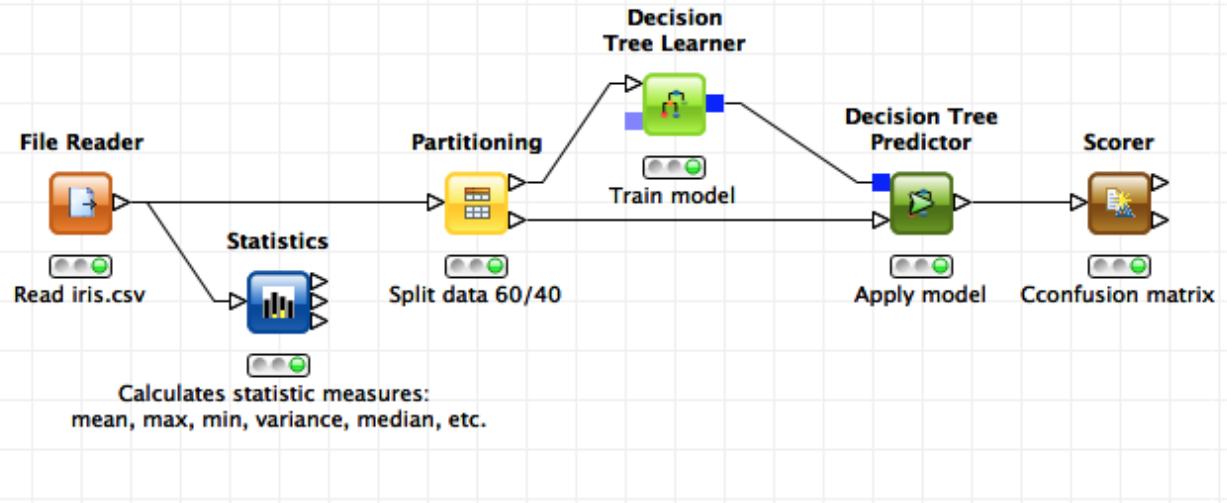


# KNIME Analytics Platform

- Platform for data analytics, reporting, and visualization
- GUI-based approach with drag-and-drop interface
- Nodes provide functionality
- Nodes are assembled to create workflows



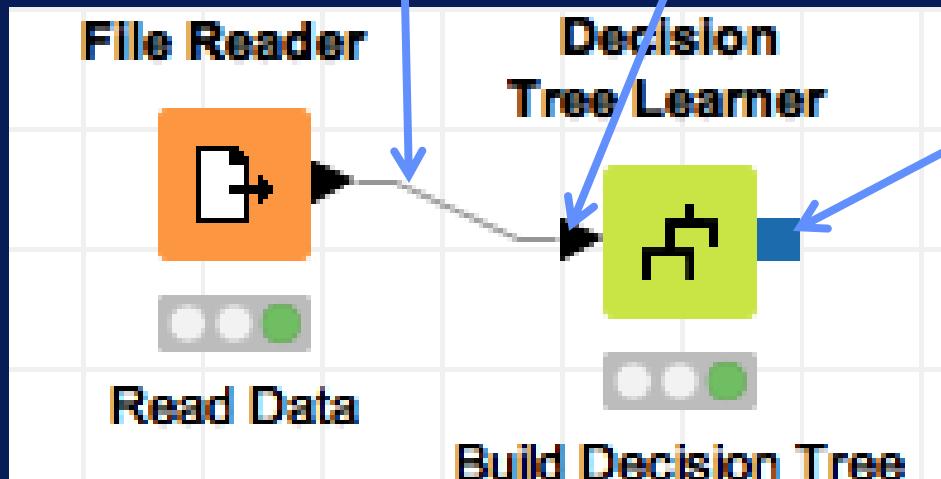
# KNIME Workflow

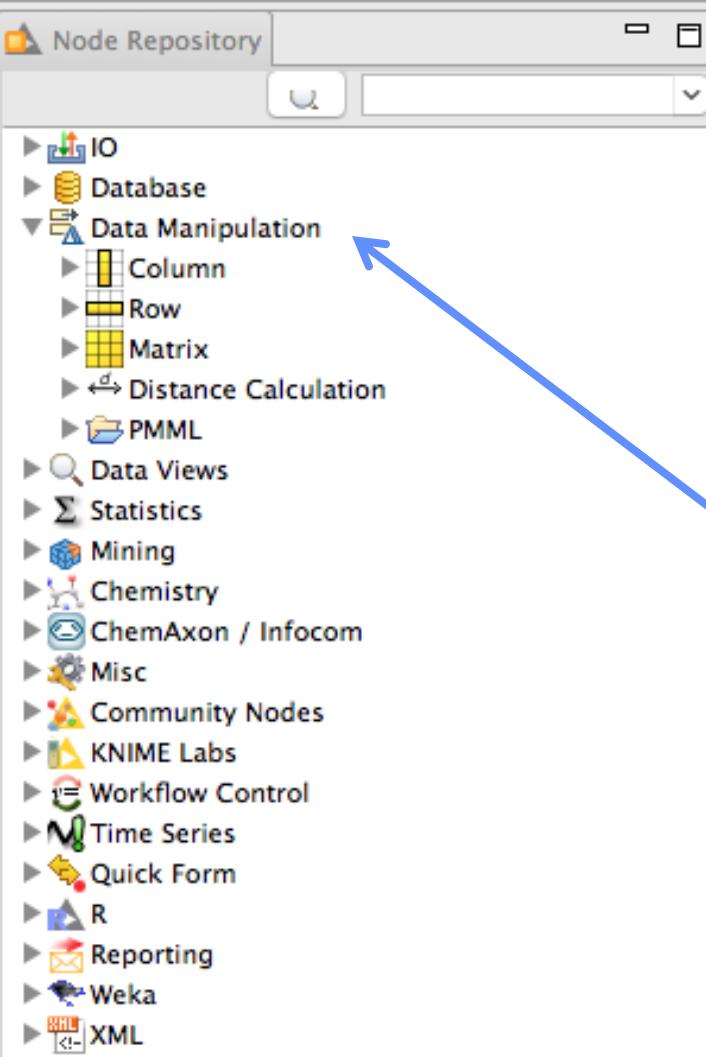


- **Visual representation of steps in analysis process**
- **Workflow is composed of nodes**

# KNIME Node

## Node implements specific operation





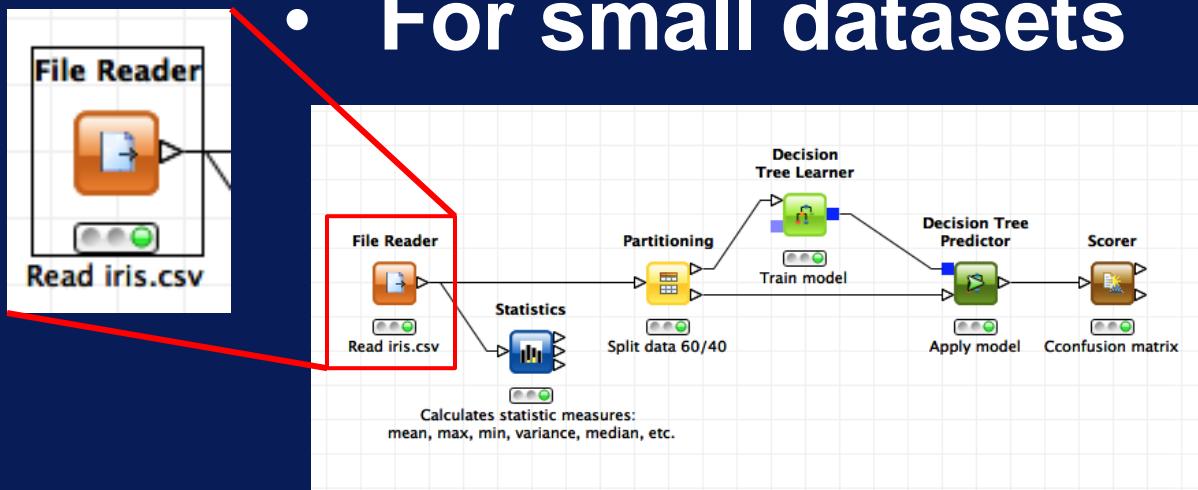
# Node Repository

Contains nodes organized by category

Expand category  
to see  
subcategories  
and nodes

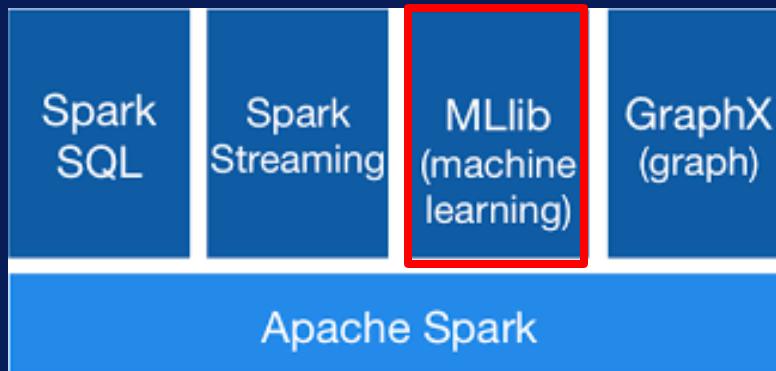
# KNIME

- GUI-based
- Drag-and-drop
- Interactive
- For small datasets



# Spark MLlib

- Scalable machine learning library
- Runs on Spark
  - Distributed computing platform



# Spark MLlib

- Write code to implement machine learning operations

## # Read and parse data

```
data = sc.textFile("data.txt")
```

## # Decision tree for classification

```
model = DecisionTree.trainClassifier  
        (parsedData, numClasses=2)  
print(model.toDebugString())  
model.save(sc, "decisionTreeModel")
```

# Spark MLlib

- Provides APIs for

Java

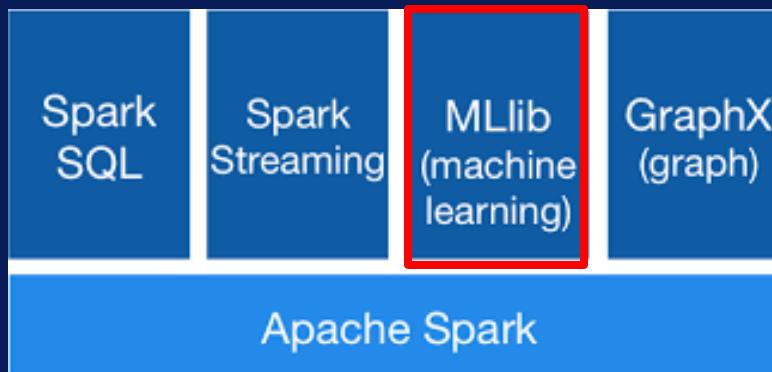
Scala

Python

R

# Spark MLlib

- Distributed platform
- Scalable algorithms & techniques
- For large datasets
- Requires coding



# KNIME & Spark MLlib



# Data Terminology

# After this video you will be able to..

- Describe what a feature is and how it relates to a sample
- Name some alternative terms for ‘feature’
- Summarize how a categorical feature differs from a numerical feature

# Terms to Describe Data

Variables

Samples

ID	Date	MinTemp	MaxTemp	Rainfall
1	2010-06-17	55	75	0.1
2	2010-06-18	52	78	0.0
3	2010-06-19	50	78	0.0
4	2010-06-20	54	77	0.0

# Terms to Describe Data

**Variables**

ID	Date	MinTemp	MaxTemp	Rainfall
1	2010-06-17	55	75	0.1
2	2010-06-18	52	78	0.0
3	2010-06-19	50	78	0.0
4	2010-06-20	54	77	0.0

Samples

# Terms to Describe Data

Variables

Samples

ID	Date	MinTemp	MaxTemp	Rainfall
1	2010-06-17	55	75	0.1
2	2010-06-18	52	78	0.0
3	2010-06-19	50	78	0.0
4	2010-06-20	54	77	0.0

# Other Names for ‘Sample’

sample

row

instance

record

observation

example

ID	Date	MinTemp	MaxTemp	Rainfall
1	2010-06-17	55	75	0.1
2	2010-06-18	52	78	0.0
3	2010-06-19	50	78	0.0
4	2010-06-20	54	77	0.0

# Other Names for ‘Variable’

variable

feature

dimension

column

attribute

field

Variables

ID	Date	MinTemp	MaxTemp	Rainfall
1	2010-06-17	55	75	0.1
2	2010-06-18	52	78	0.0
3	2010-06-19	50	78	0.0
4	2010-06-20	54	77	0.0
...				

# Data Types

- Most common

Numeric

Categorical

- Others

String

Date

...

# Numeric Variables

- Values are numbers
- Also called ‘quantitative’

1

$7 \times 10^5$

163.92

-0.4902

# Examples of Numeric Variables

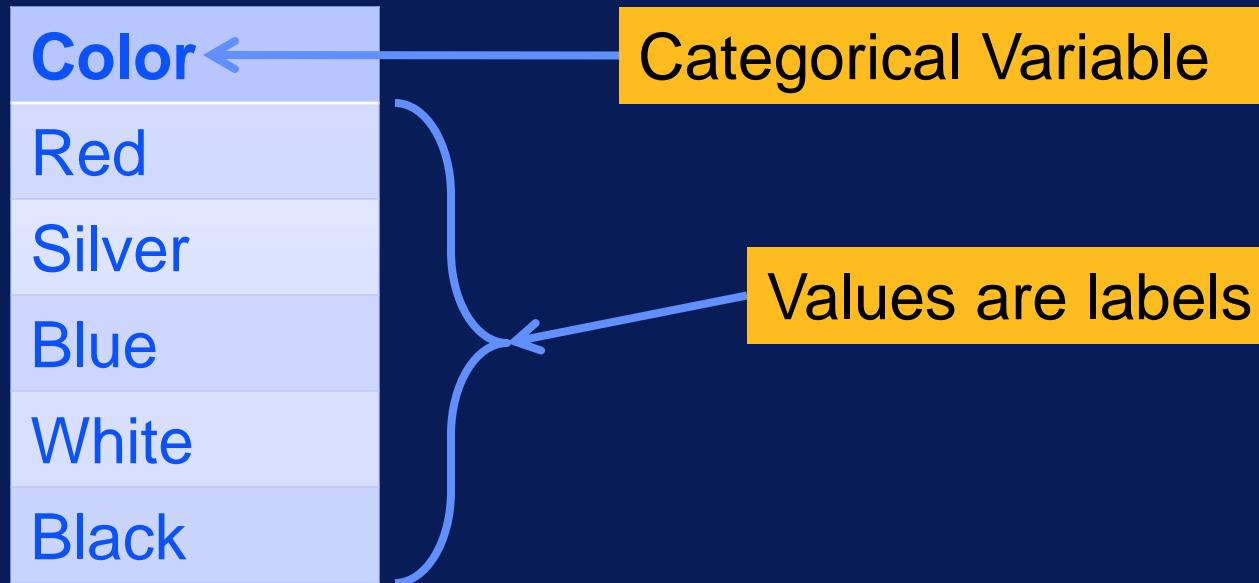
- Height
- Score on an exam
- Number of transactions per hour
- Change in stock price



01.230	0.472	-2.80%	N/A	0
61.8175	0.420	-1.53%	30.400	200
82.230	0.1325	-0.68%	N/A	0
16.370	1.250	-0.21%	N/A	0
39.500	0.340	-1.50%	N/A	0
62.748	0.340	-2.03%	16.310	600
1.570	0.412	-0.87%	38.900	16.380
1.440	4.300	-0.85%	N/A	3400
0.770	0.130	-0.96%	N/A	40.710
69	0.010	-0.80%	N/A	400
5	1.0331	-0.17%	8.080	0
5	0.7825	-1.55%	N/A	12000
5	0.190	-2.15%	N/A	8.090
		-1.06%	N/A	17700
			N/A	0
			N/A	7.770

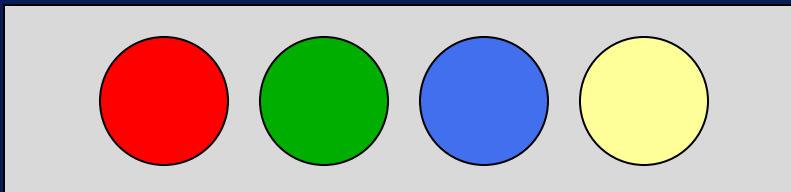
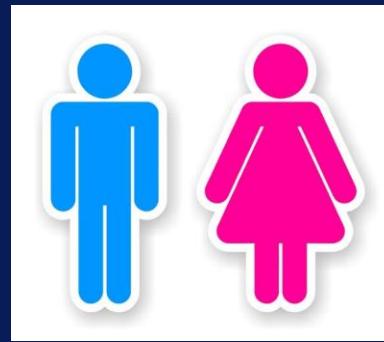
# Categorical Variables

- Values are labels, names, or categories
- Also called ‘qualitative’ or ‘nominal’



# Examples of Categorical Variables

- Gender
- Marital status
- Type of customer
- Product categories
- Color of an item



**Sample**



- Instance
- Record
- Row
- Observation
- ...

**Variable**

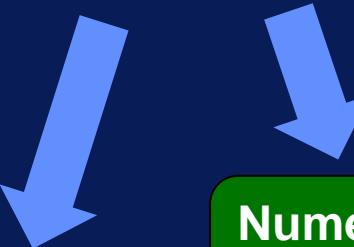


- Feature
- Field
- Column
- ...

Variables				
ID	Date	MinTemp	MaxTemp	Rainfall
1	2010-06-17	55	75	0.1
2	2010-06-18	52	78	0.0
3	2010-06-19	50	78	0.0
4	2010-06-20	54	77	0.0

Samples

**Categorical**  
*Qualitative*  
*Nominal*



**Numeric**  
*Quantitative*



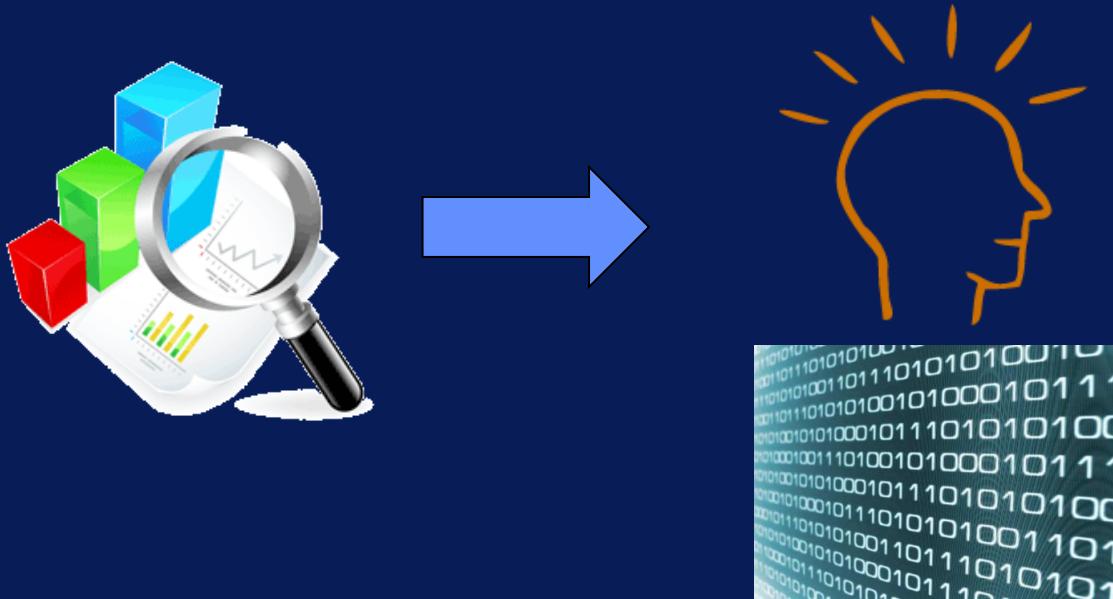
# Data Exploration

# After this video you will be able to..

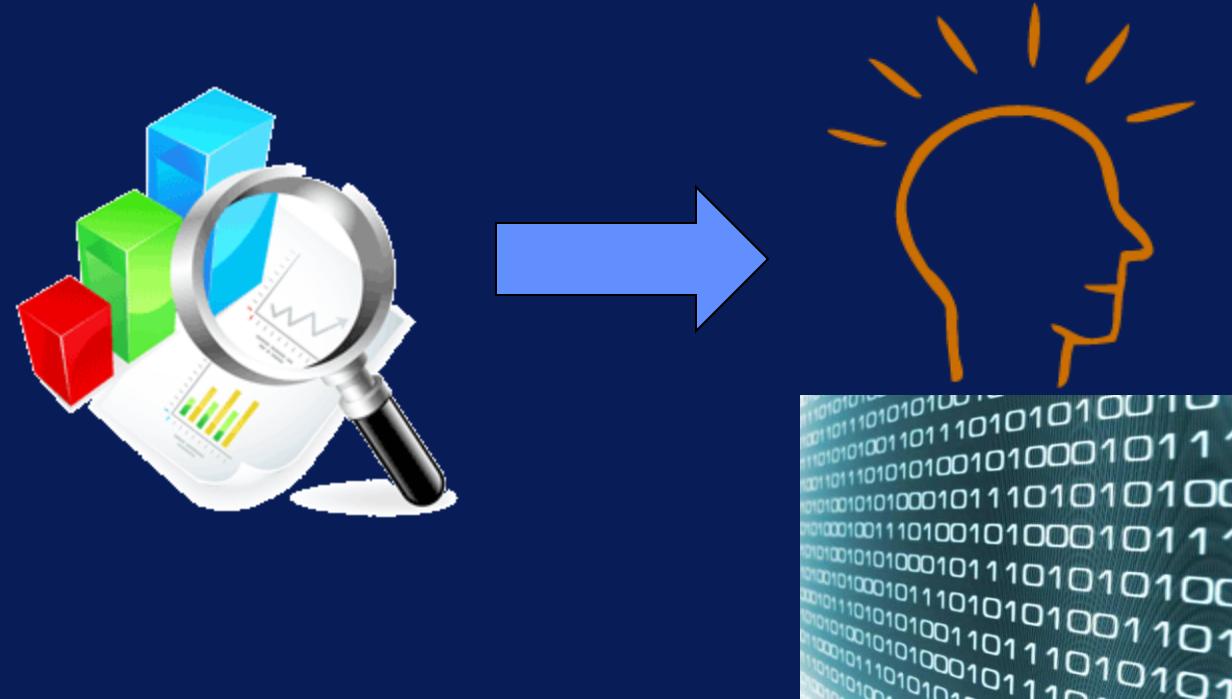
- Explain why data exploration is necessary
- Articulate the objectives for data exploration
- List the categories of techniques for exploring data

# Why Explore Data?

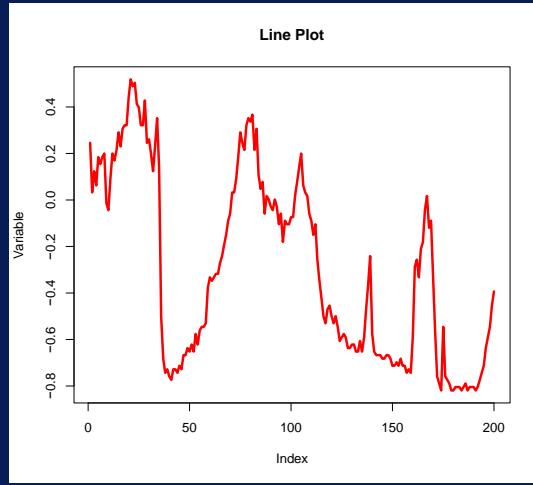
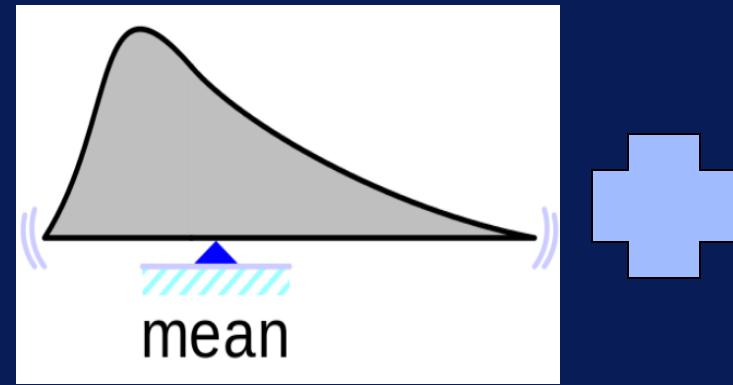
# Goal: To understand your data



# Exploratory Data Analysis (EDA)



# Ways to Explore Data

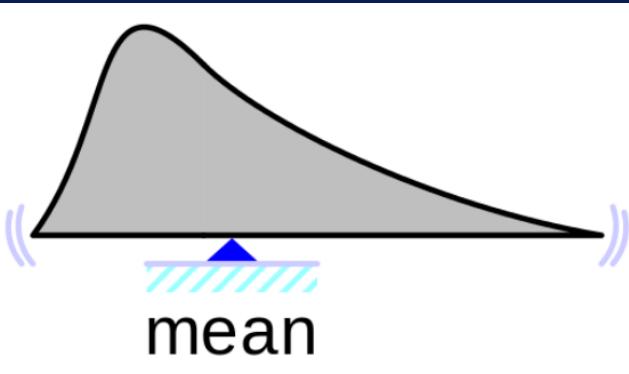
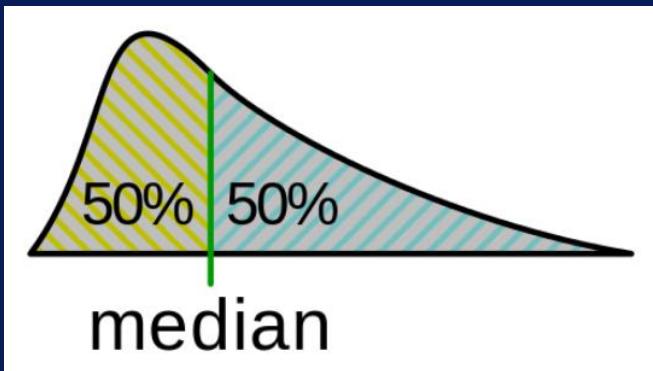
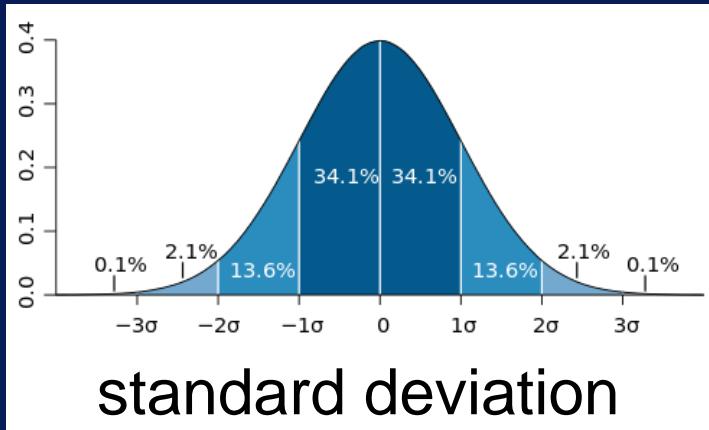


Summary  
Statistics

Visualization

# Summary Statistics

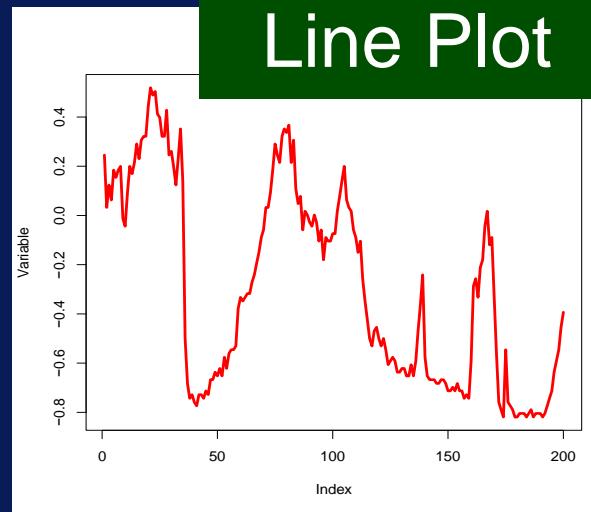
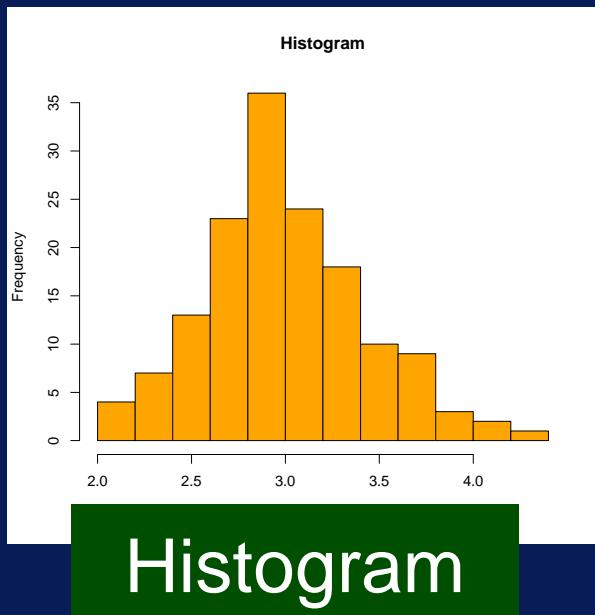
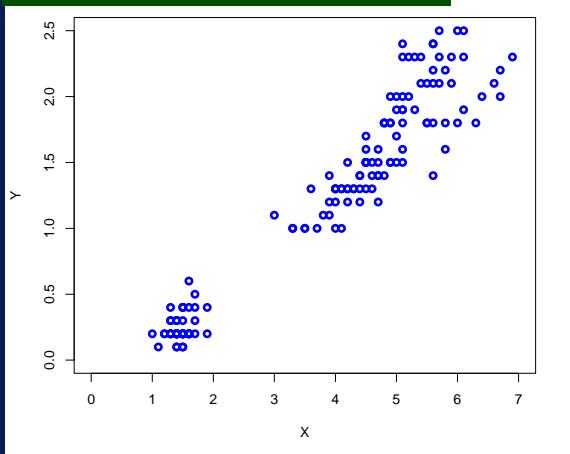
- Information that summarizes dataset



# Data Visualization

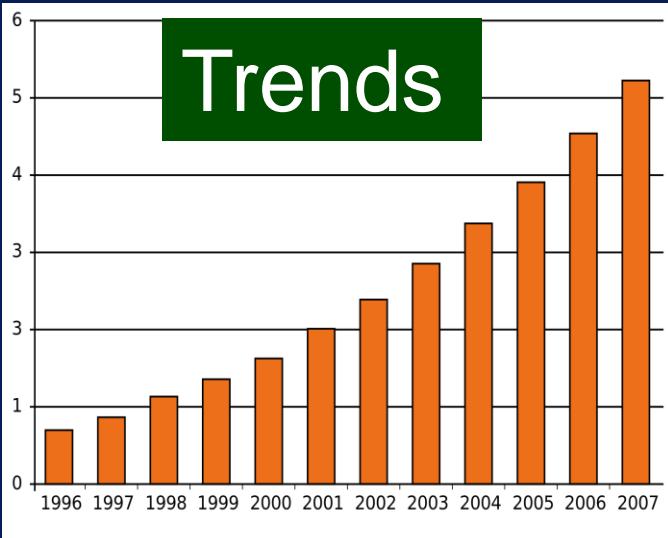
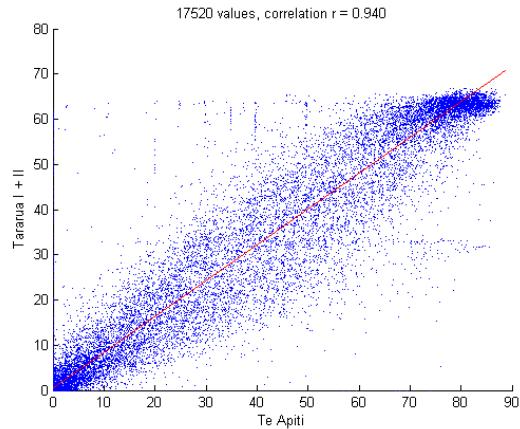
- Look at data graphically

## Scatter Plot

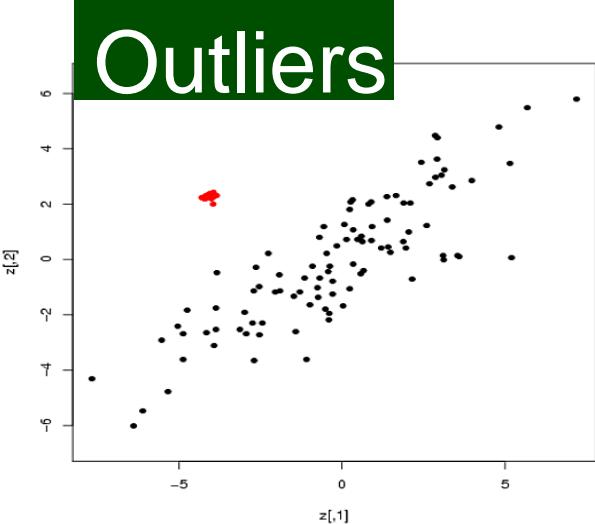


# Some Things to Look For

## Correlations



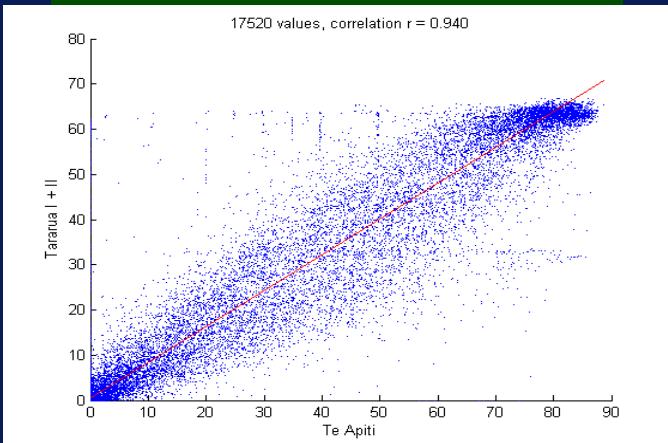
## Outliers



# Correlations

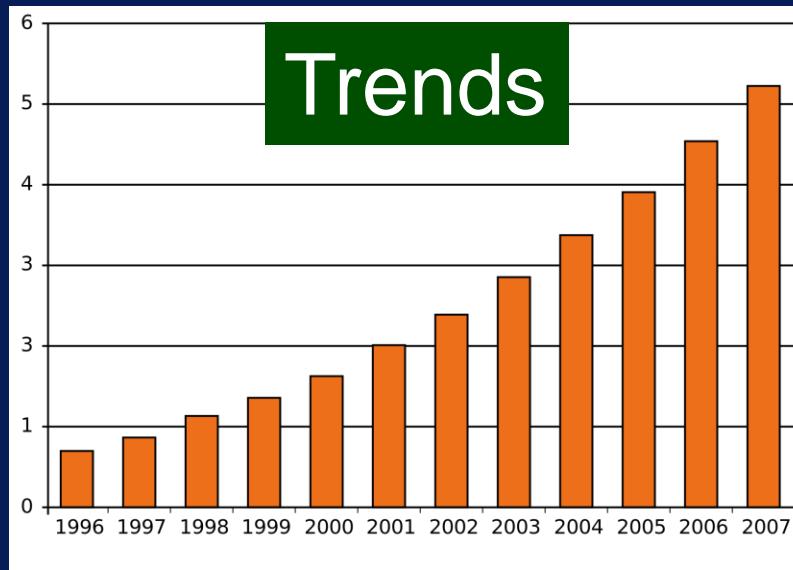
- Provide information about relationship between variables

## Correlations



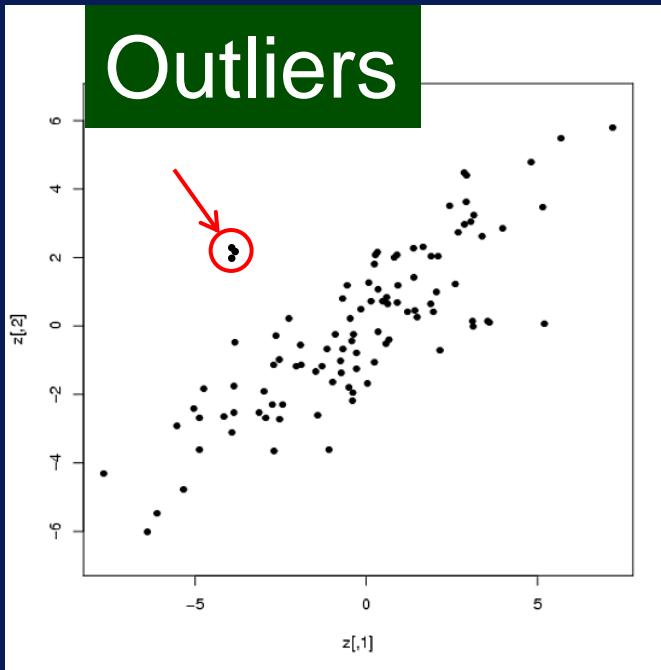
# Trends

- Indicate general characteristics of data

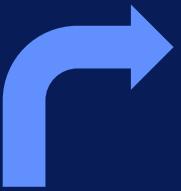


# Outliers

- Indicate potential problems with data



# Data Exploration



Informed  
Analysis

A thick, light blue curved arrow pointing from the top-right towards the bottom-left, indicating a feedback loop from Analysis back to Exploration.

Data  
Understanding

A thick, light blue curved arrow pointing from the bottom-left towards the top-right, indicating a flow or cycle between the stages.

Data  
Exploration

# Exploring Data through Summary Statistics

# After this video you will be able to..

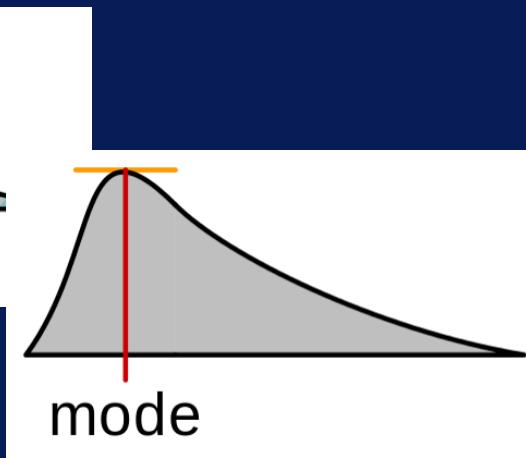
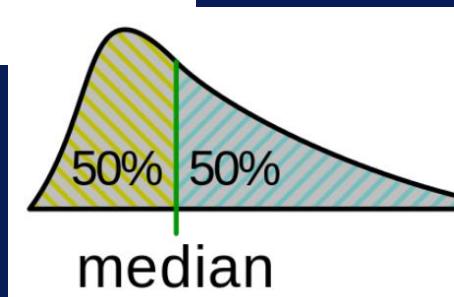
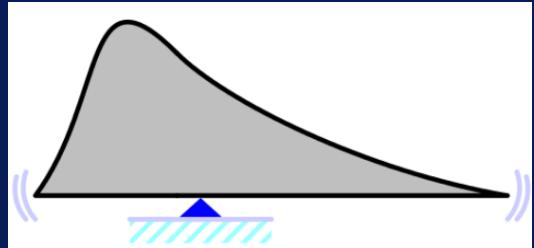
- Define what a summary statistic is
- List three common summary statistics
- Explain how summary statistics are useful in exploring data

# What are summary statistics?

- Quantities that summarize and describe a set of data values
- Measures of
  - Location: mean, median
  - Spread: standard deviation
  - Shape: skewness

# Measures of Location

Describe central or typical value of dataset



# Measures of Location - Example

Age	Age (sorted)
35	21
42	22
78	35
22	42
56	42
50	50
42	56
78	78
21	78
87	87

Mean = 51.1

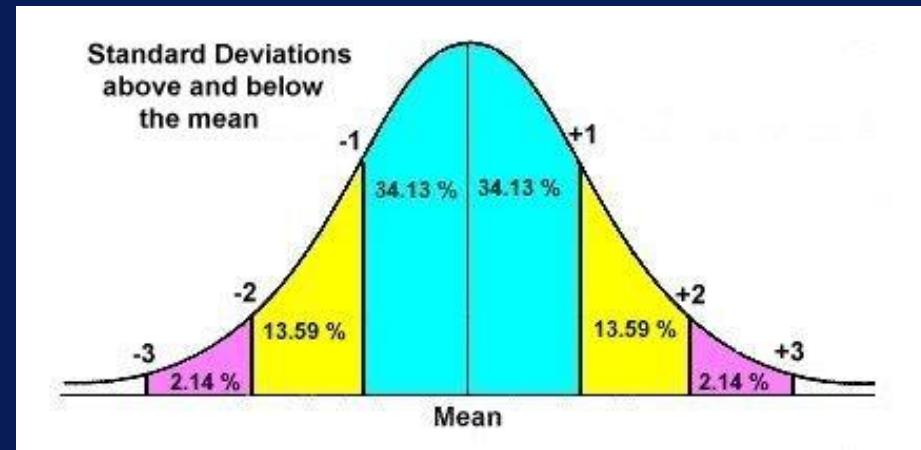
Median =  $(42+50)/2 = 46$

Mode = 42 & 78

# Measures of Spread

Describe how dispersed or varied data is

minimum      maximum  
standard variation  
deviation      range



# Measures of Spread – Example

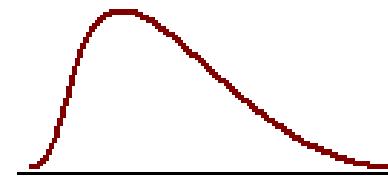
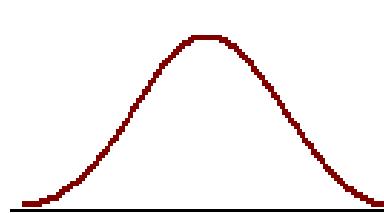
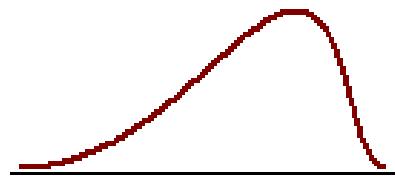
Age	Age (sorted)
35	21
42	22
78	35
22	42
56	42
50	50
42	56
78	78
21	78
87	87

$$\text{Range} = 87 - 21 = 66$$

$$\text{Variance} = 548.767$$

$$\text{Standard deviation} = 23.426$$

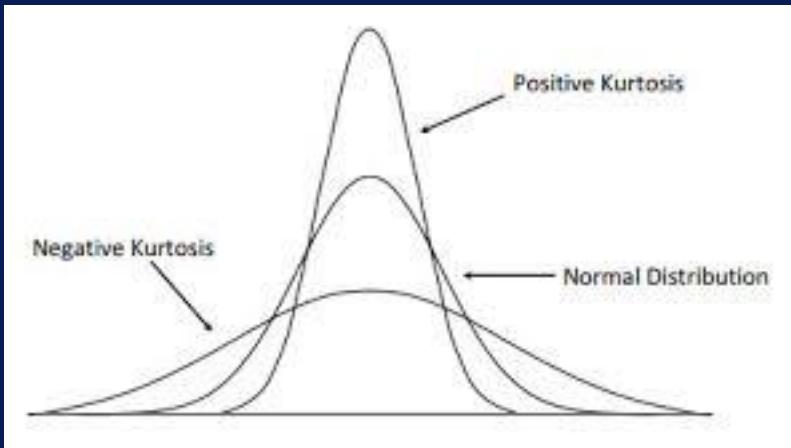
# Measures of Shape



Negatively skewed distribution  
or Skewed to the left  
Skewness < 0

Normal distribution  
Symmetrical  
Skewness = 0

Positively skewed distribution  
or Skewed to the right  
Skewness > 0



skewness

kurtosis

# Measures of Shape – Example

Age

35

42

78

22

56

50

42

78

21

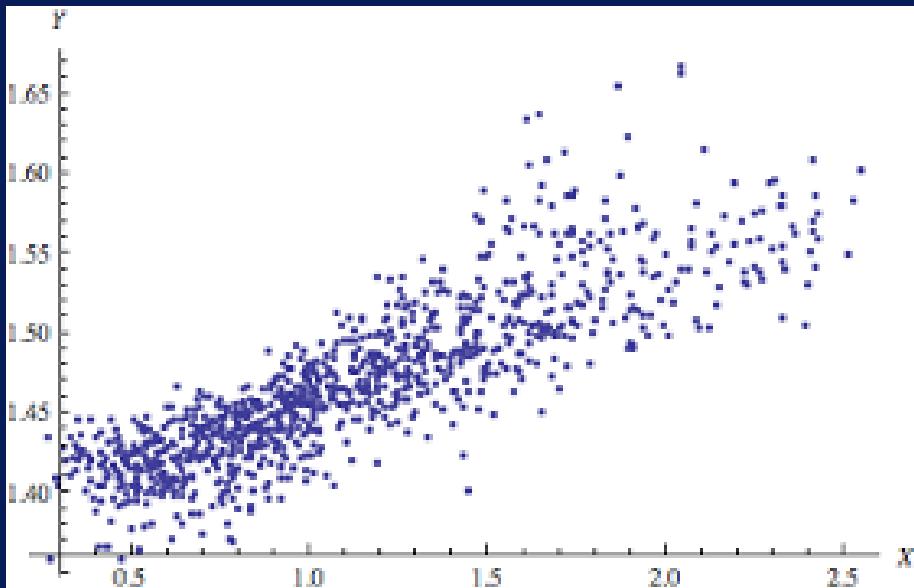
87

Skewness = 0.2995

Kurtosis = -1.2028

# Measures of Dependence

Describe relationship between variables



correlation

# Measures of Dependence – Example

Height	Weight
180	68
153	70
204	84
133	44
208	81
142	53
122	40
168	50
175	64
200	72

Correlation = 0.8906

# Statistics on Categorical Variables

Describe number of categories and frequency of each category

Color/Pet	White	Brown	Black	Orange	Total
Dog	34	44	32	0	110
Cat	25	2	43	0	70
Fish	1	0	5	33	39
<b>Total</b>	<b>60</b>	<b>46</b>	<b>80</b>	<b>33</b>	<b>219</b>

contingency table

# Contingency Table - Example

Color/Pet	White	Brown	Black	Orange	Total
Dog	34	44	32	0	110
Cat	25	2	43	0	70
Fish	1	0	5	33	39
<b>Total</b>	60	46	80	33	219

# Check Dimensions

- Check number of rows and columns

ID	Date	MinTemp	MaxTemp	Rainfall
1	2010-06-17	56	75	0.1
2	2016-06-18	52	78	0.0
3	2010-06-19	50	78	0.0
4	2010-06-20	54	77	0.0

# rows = # samples ?

# columns = # variables ?

# Check Values

- Check values in some samples

Should temperature values in F or C?

Is this correct?

ID	Date	MinTemp	MaxTemp	Rainfall
1	2010-06-17	56	24	0.1
2	2016-06-18	52	26	3,678.9
3	2010-06-19	50	26	0.0
4	2010-06-20	54	25	0.0

Is this date or timestamp?

# Check Missing Values

ID	Date	MinTemp	MaxTemp	Rainfall
1	2010-06-17	56	75	--
2	2016-06-18	52	78	--
3	2010-06-19	--	78	0.1
4	2010-06-20	54	77	--

Does feature  
have mostly  
missing values?

How many samples have  
missing values?

# Summary Statistics

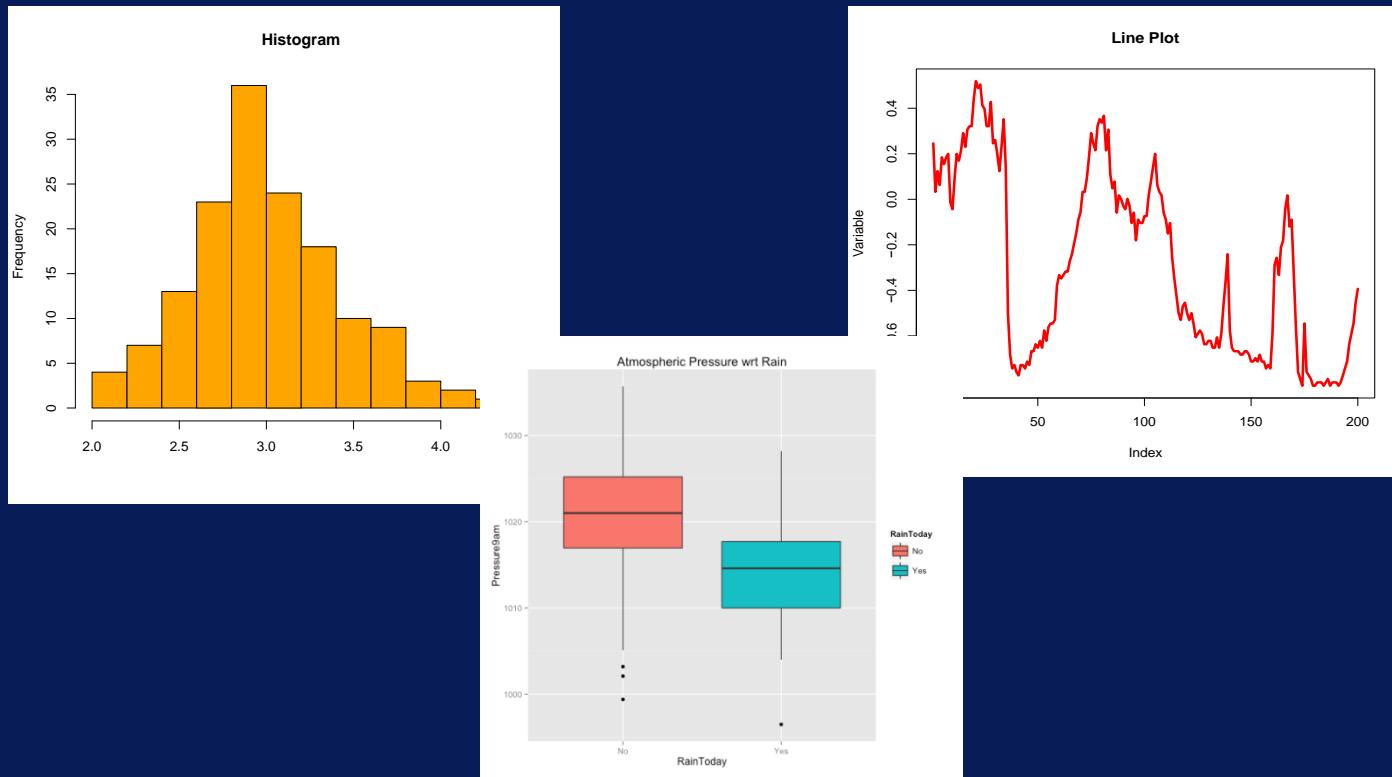
- **Measures of**
  - Location, spread, shape, dependence
- **Contingency table**
  - For categorical variables
- **Data validation**
  - Dimensions, missing values

# Exploring Data through Plots

# After this video you will be able to..

- Discuss how plots can be useful in exploring data
- Describe how you would use a scatter plot
- Summarize what a boxplot shows

# Visualizing Data

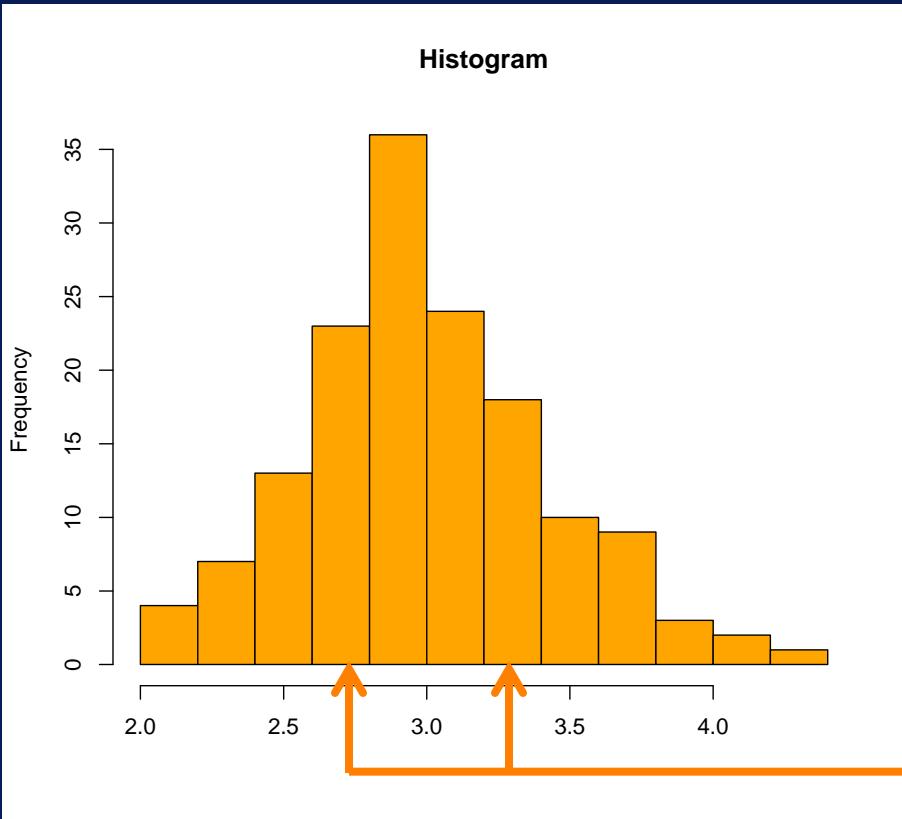


# Types of Plots

- Histogram
- Line plot
- Scatter plot
- Bar plot
- Box plot
- others

# Histogram

- Shows distribution of numeric variable

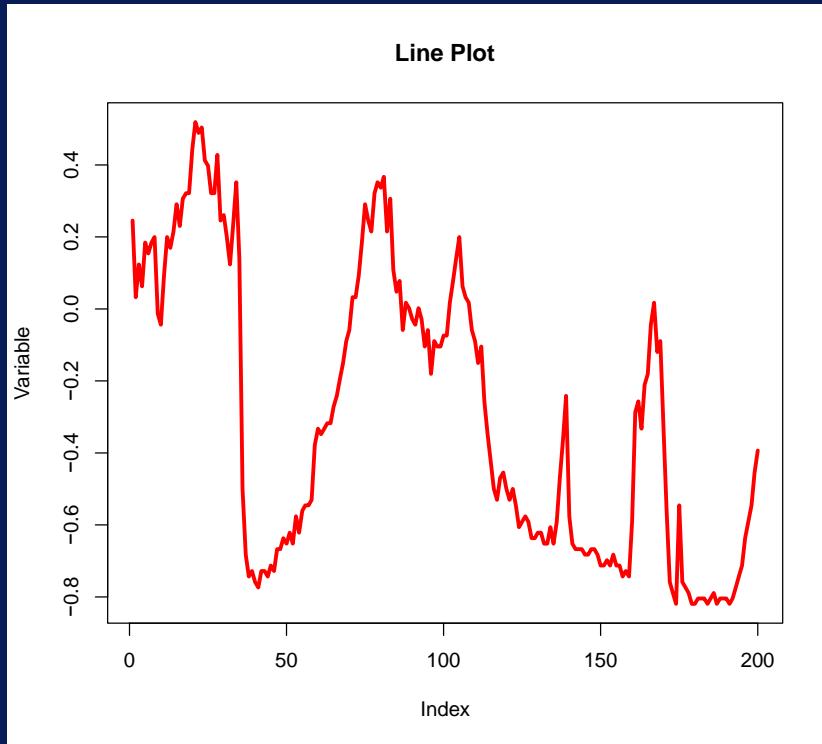


Bins

# What a Histogram Shows

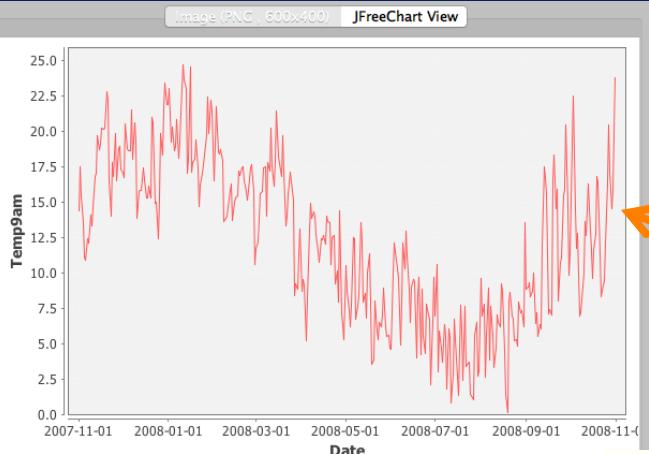


# Line Plot



- Shows change in data over time

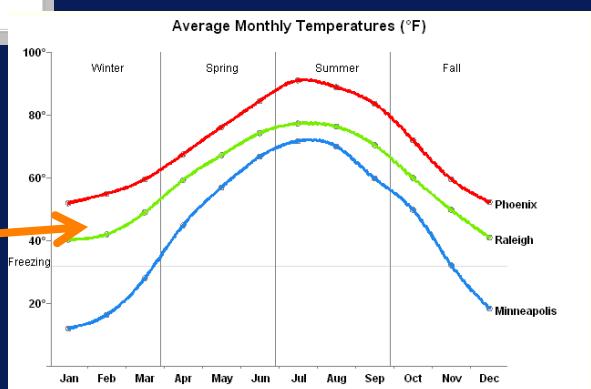
# What a Line Plot Shows



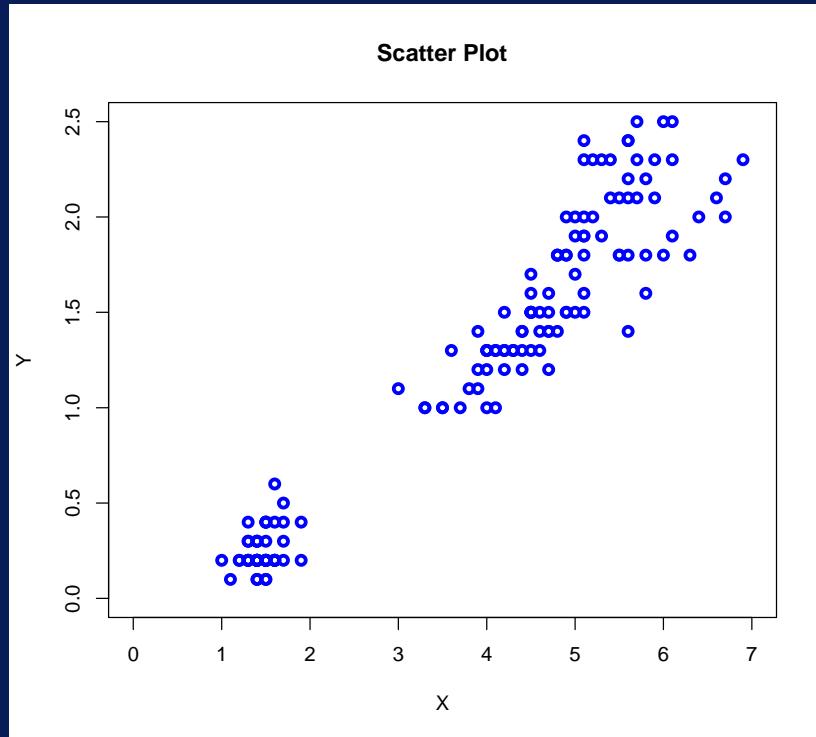
Trend  
Cyclical pattern



Compare variables



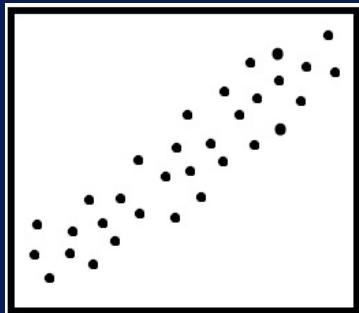
# Scatter Plot



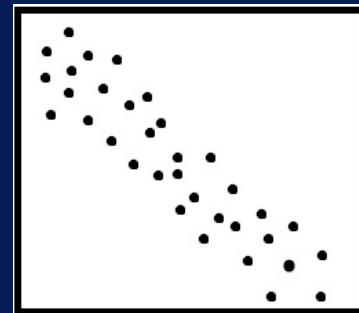
- Shows relationship between two variables

# What a Scatter Plot Shows

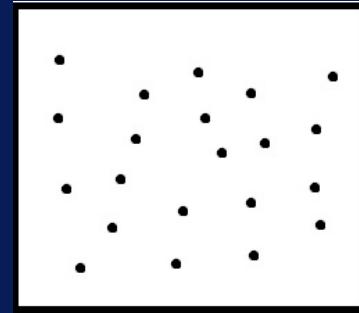
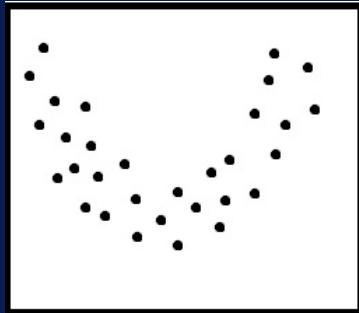
Positive  
Correlation



Negative  
Correlation

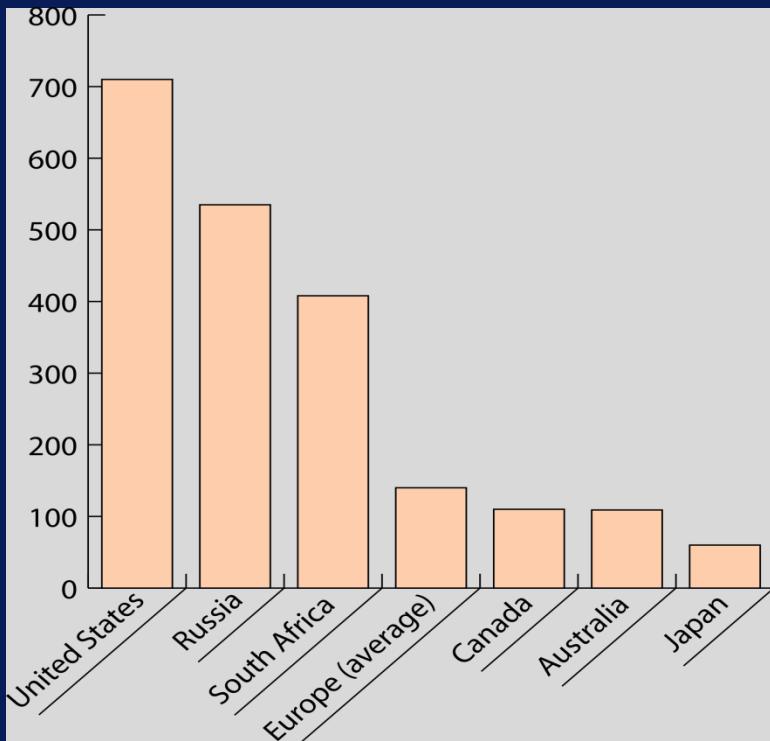


Non-  
Linear  
Correlation



No Correlation

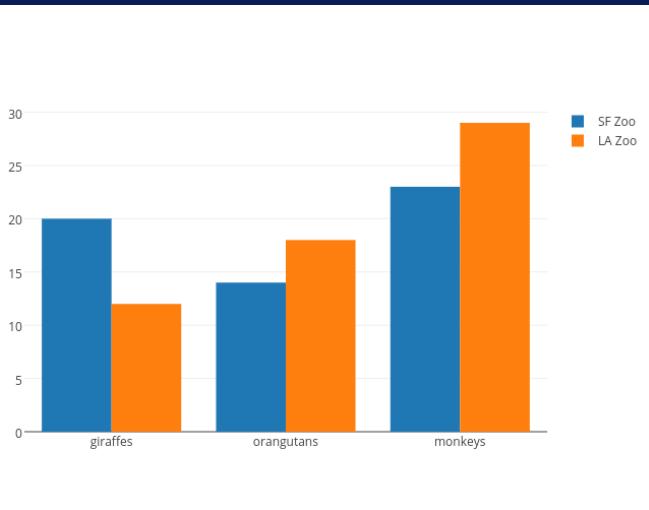
# Bar Plot



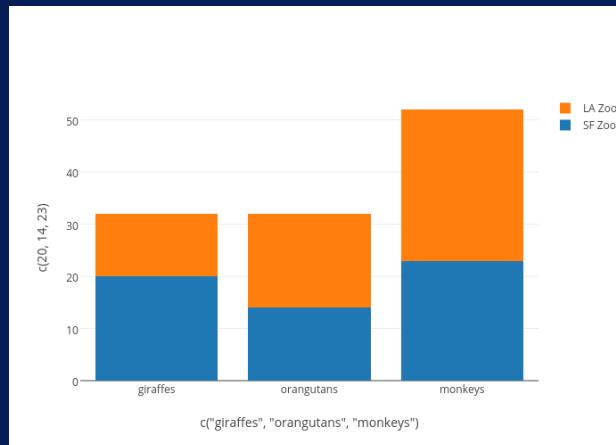
- Shows distribution of categorical variable

# What a Bar Plot Shows

## Grouped Bar Chart

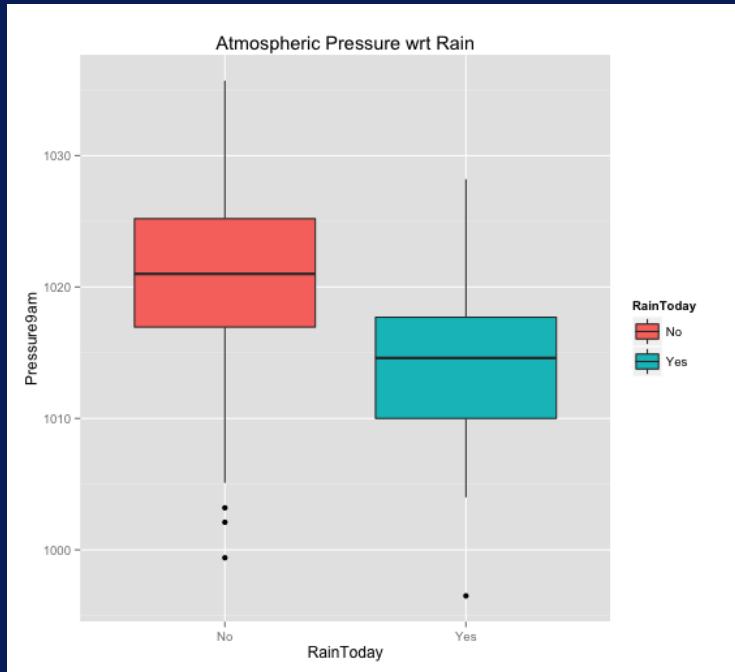


## Stacked Bar Chart

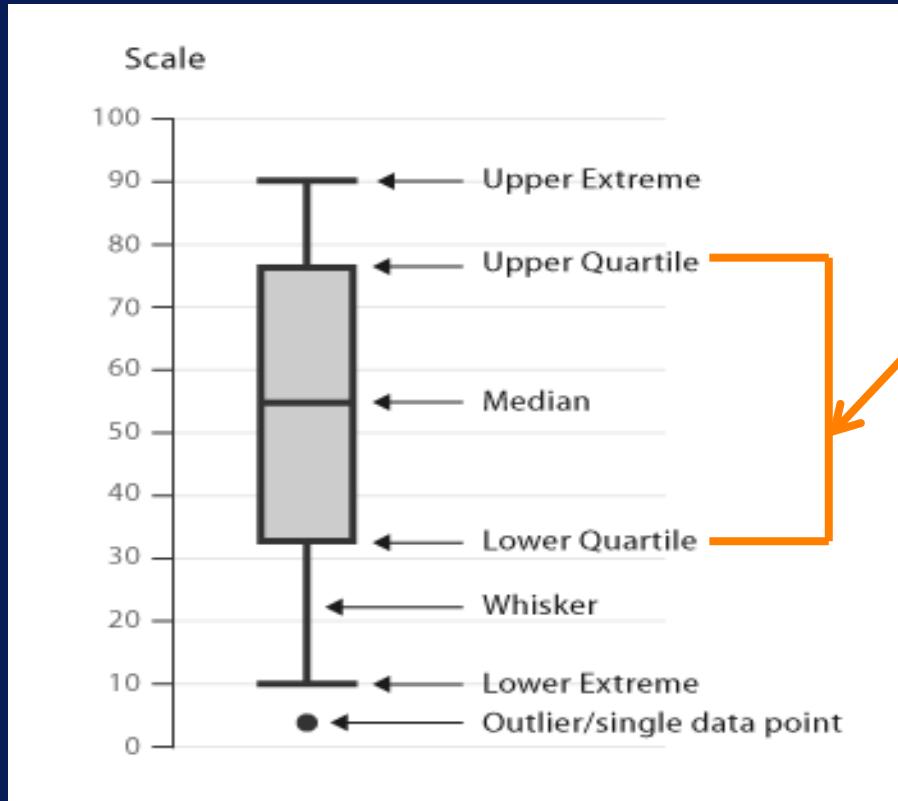


# Box Plot

- Compares distributions of variables

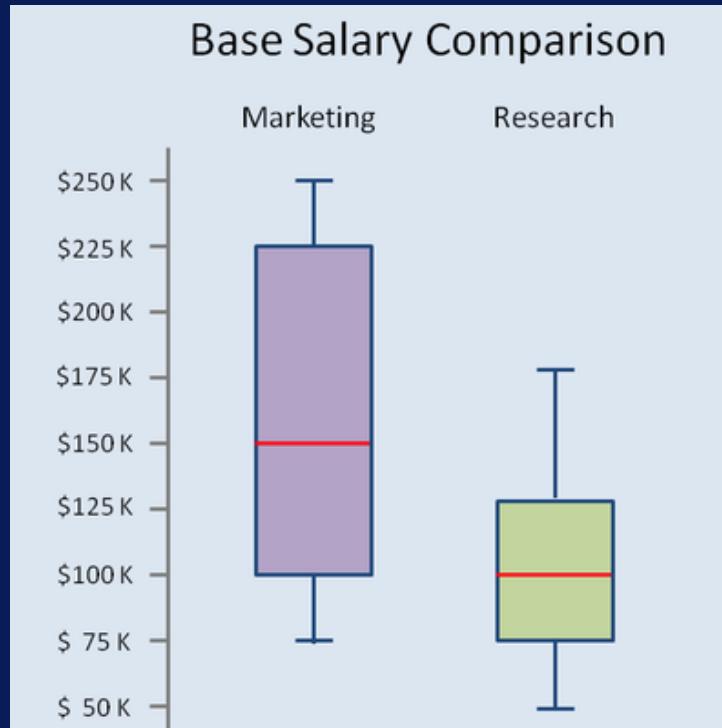


# Components of a Box Plot



The middle 50% of data are in this region

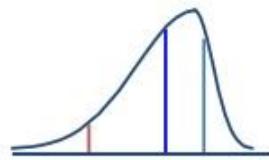
# What a Box Plot Shows



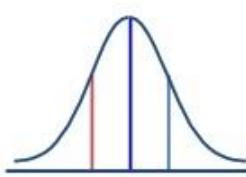
# What a Box Plot Shows

## Distribution Shape and The Boxplot

Negative Skew



Symmetric



Positive Skew

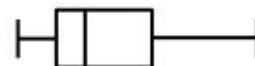
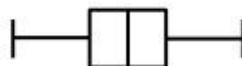
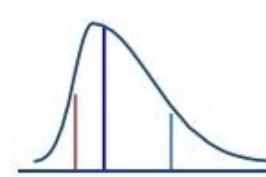


Image source: <http://www.slideshare.net/mido02/chap-3gbu>

# Data Visualization

- Provides intuitive way to look at data
- Should be used with summary statistics for data exploration
- Are also useful for communicating results



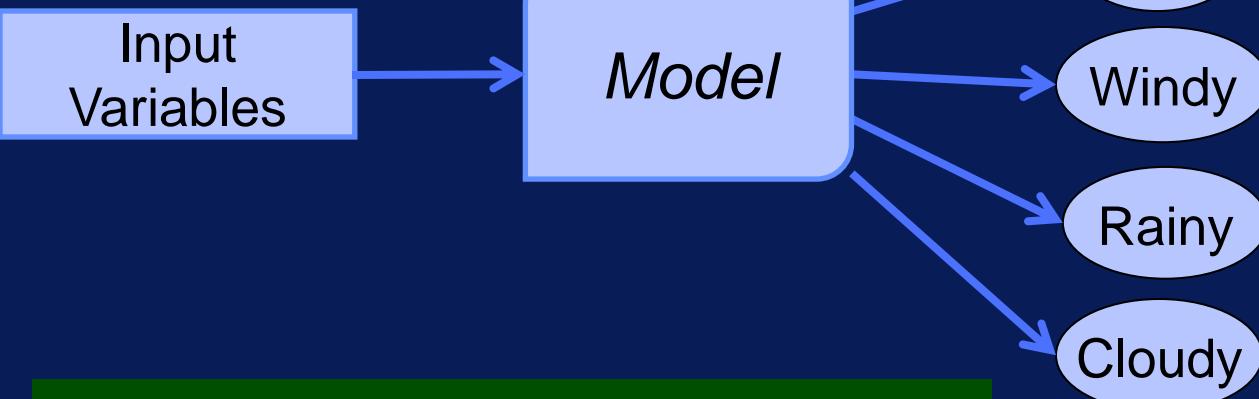
# Classification Overview

# After this video you will be able to..

- Define what classification is
- Discuss whether classification is supervised or unsupervised
- Describe how binomial classification differs from multinomial classification



# Classification



Target variable  
is categorical

Goal:  
Given input variables,  
predict category

# Data for Classification

The diagram illustrates the components of a classification dataset. At the top, two boxes are labeled: 'Input Variables' in blue text on a grey background, and 'Target Variable' in red text on a grey background. A blue bracket below these boxes spans across the first three columns of the table, indicating they represent input features. A red arrow points downwards from the 'Target Variable' box to the fourth column of the table, which is labeled 'Weather' in blue text.

Temperature	Humidity	Wind Speed	Weather
79	48	2.7	Sunny
60	80	3.8	Rainy
68	45	17.9	Windy
57	77	4.2	Cloudy

# Classification is Supervised

Target      Label      Output

Class Variable      Class      Category

The diagram illustrates the concept of classification in supervised learning. It shows four equivalent terms at the top: 'Target', 'Label', 'Output', and 'Class Variable'. Below them, three more terms are shown: 'Class', 'Category', and 'Category'. A red rounded rectangle encloses the first three terms ('Target', 'Label', 'Output') and the last three terms ('Class Variable', 'Class', 'Category'). An arrow points from this red box down to a table below. The table has four columns: 'Temperature', 'Humidity', 'Wind Speed', and 'Weather'. The 'Weather' column is highlighted with a red rounded rectangle. The data rows are: (79, 48, 2.7, Sunny), (60, 80, 3.8, Rainy), (68, 45, 17.9, Windy), and (57, 77, 4.2, Cloudy).

Temperature	Humidity	Wind Speed	Weather
79	48	2.7	Sunny
60	80	3.8	Rainy
68	45	17.9	Windy
57	77	4.2	Cloudy

# Types of Classification

**Binary  
Classification**

**Multi-class  
Classification**

value1

value2

value1

value2

...

valueN

**Target has  
two values**

**Target has  $> 2$   
values**

# Classification Examples

## Binary Classification

- Will it rain tomorrow or not?
- Is this transaction legitimate or fraudulent

## Multi-Class Classification

- What type of product will this customer buy?
- Is this tweet positive, negative, or neutral

# Classification Main Points

- Predict category from input variables
- Classification is a supervised task
- Target variable is categorical



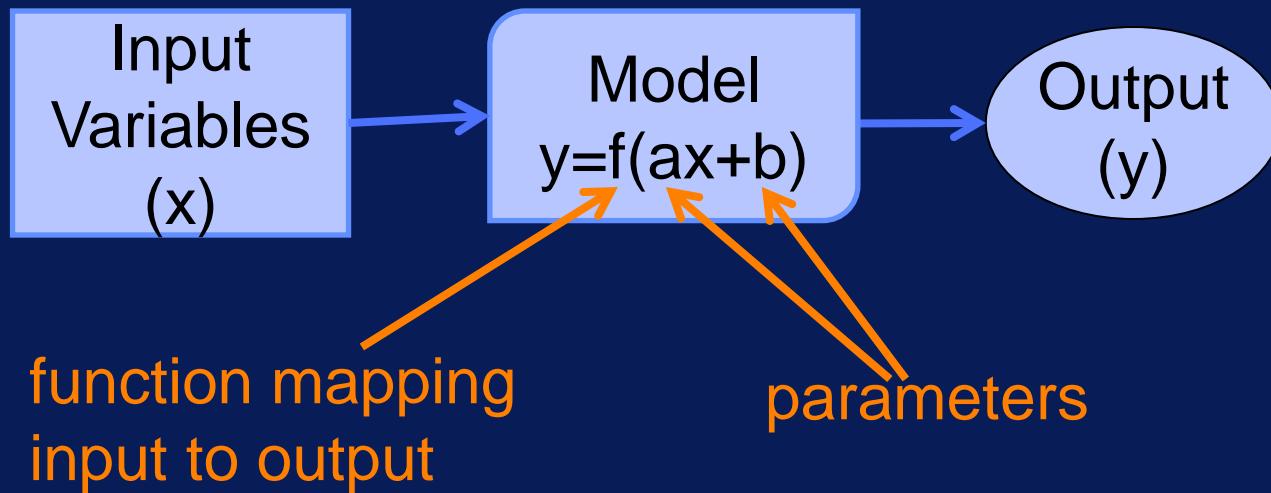
# Building and Applying a Classification Model

# After this video you will be able to..

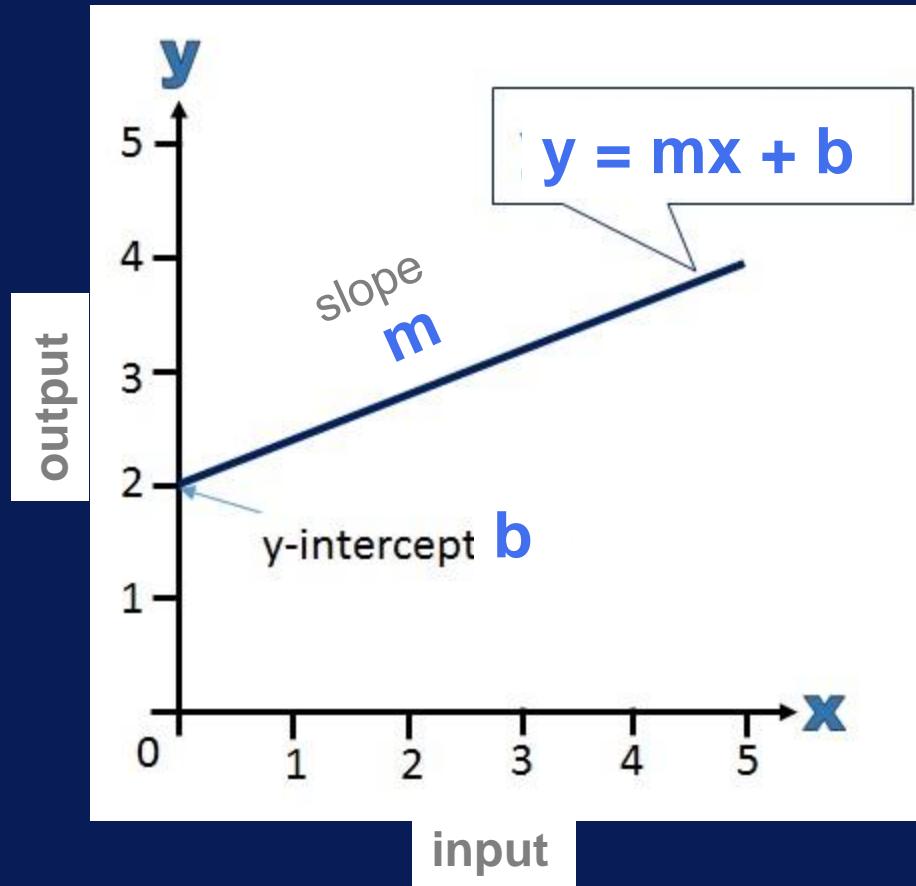
- Discuss what building a classification model means
- Explain the difference between building and applying a model
- Summarize why the parameters of a model need to be adjusted

# What is a Machine Learning Model?

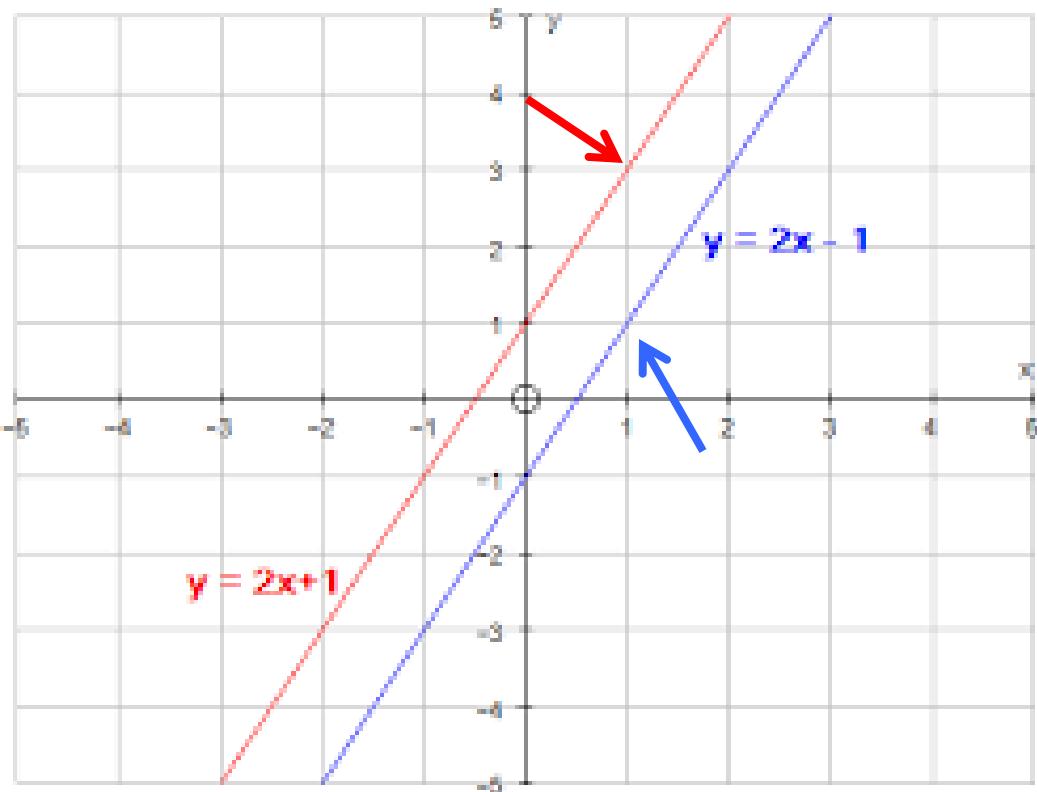
- A mathematical model with parameters that map input to output



# Example of Model



# Adjusting Model Parameters

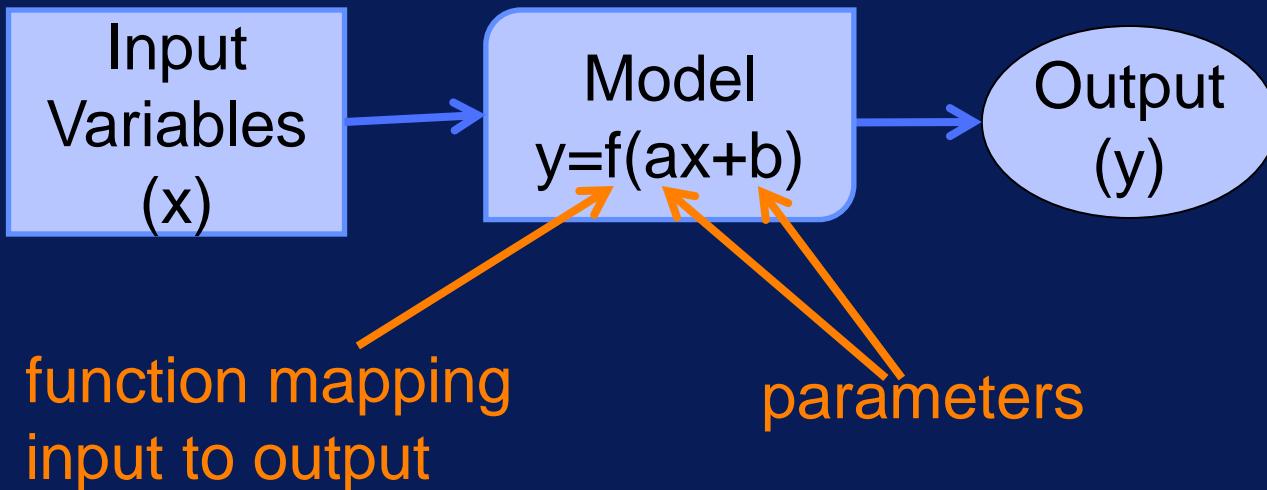


slope  $m = 2$   
y-intercept  $b = -1$   
 $x=1 \Rightarrow y=2*1-1=1$

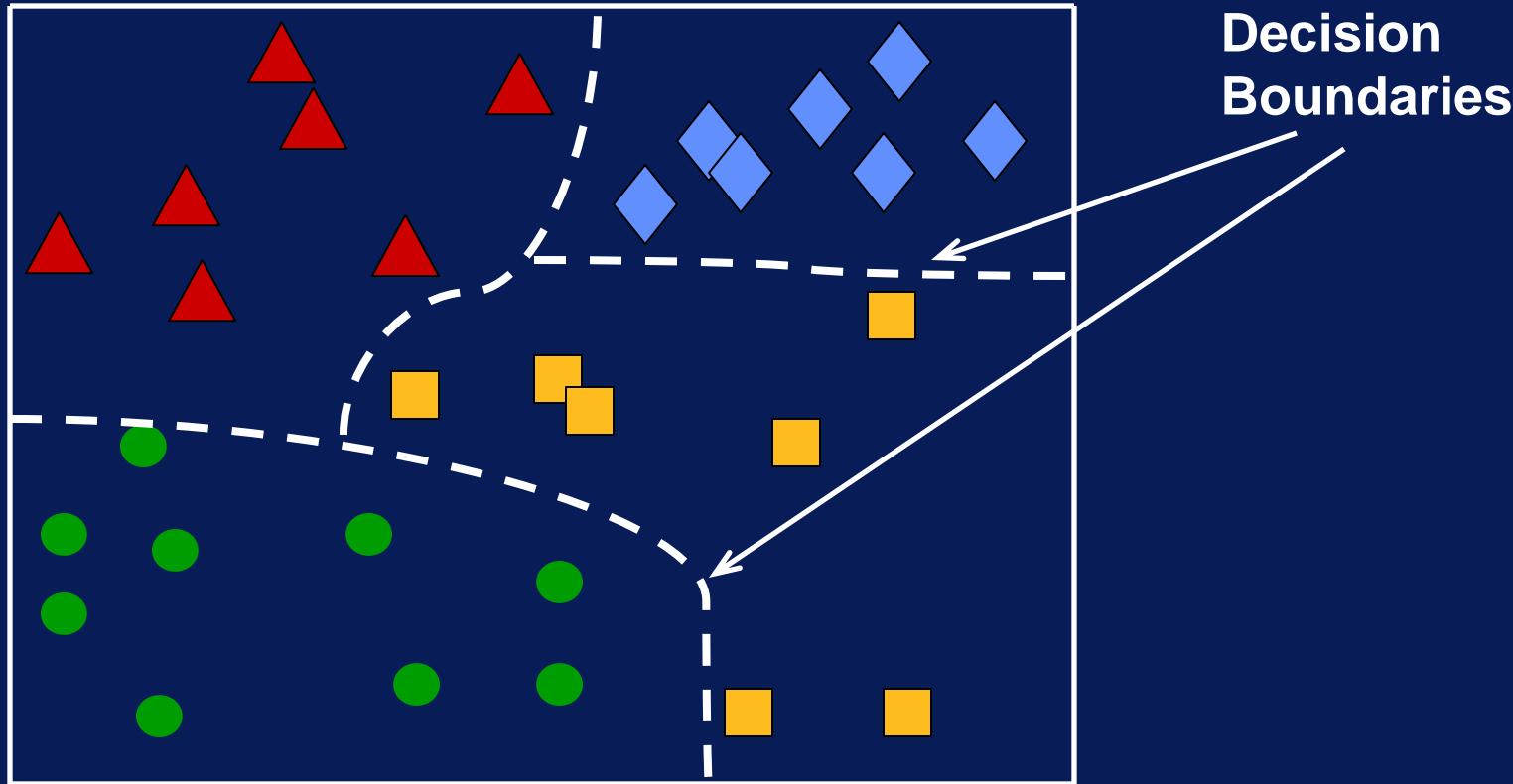
slope  $m = 2$   
y-intercept  $b = +1$   
 $x=1 \Rightarrow y=2*1+1=3$

# Building Machine Learning Model

Model parameters are adjusted during model training to change input-output mapping.



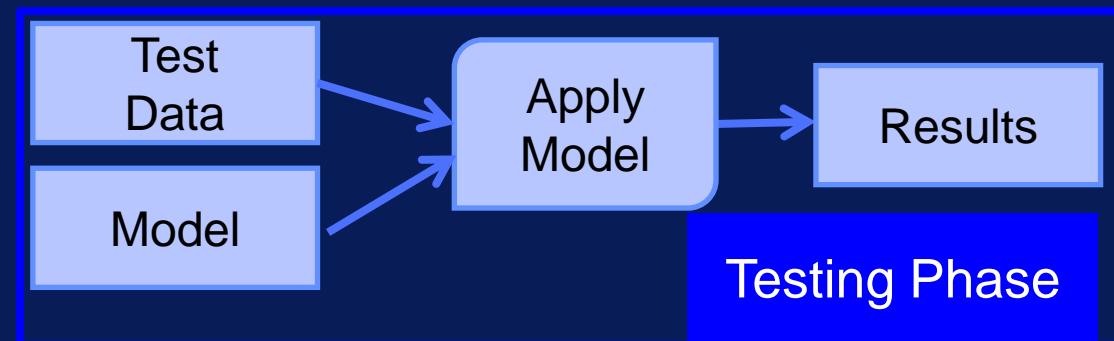
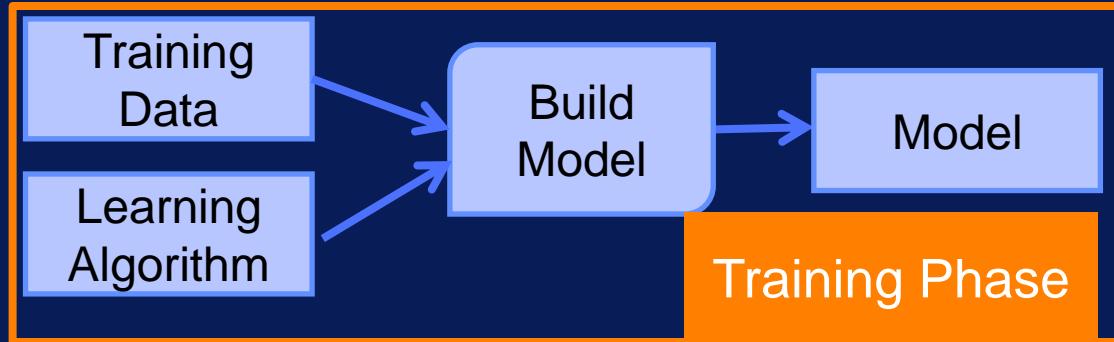
# Building Classification Model



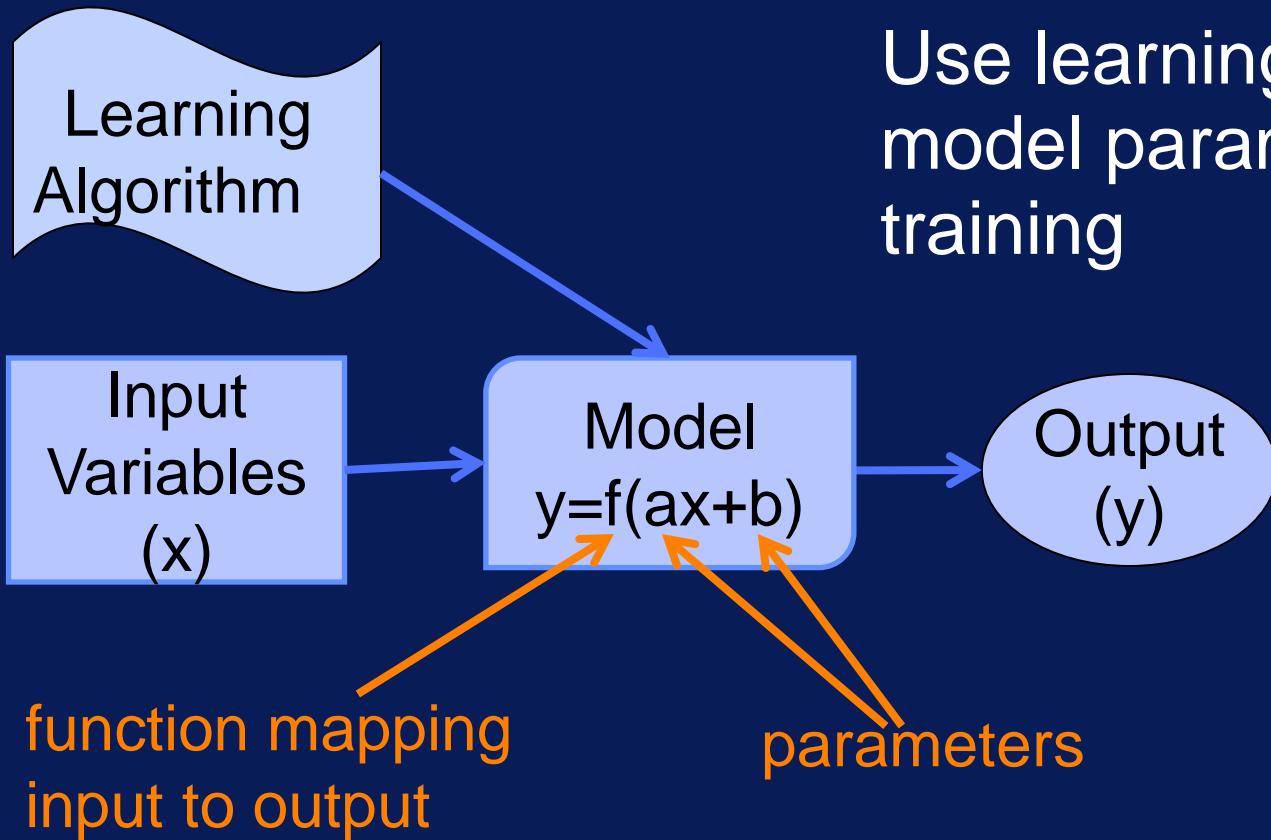
# Building vs. Applying Model

- Training Phase
  - Adjust model parameters
  - Use training data
- Testing Phase
  - Apply learned model
  - Use new data

# Building vs. Applying Model



# Building a Classification Model



Use learning algorithm to model parameters during training

Input Variables (x)

Model  
 $y=f(ax+b)$

Output (y)

function mapping  
input to output

parameters

# Classification Algorithms Overview

# After this video you will be able to..

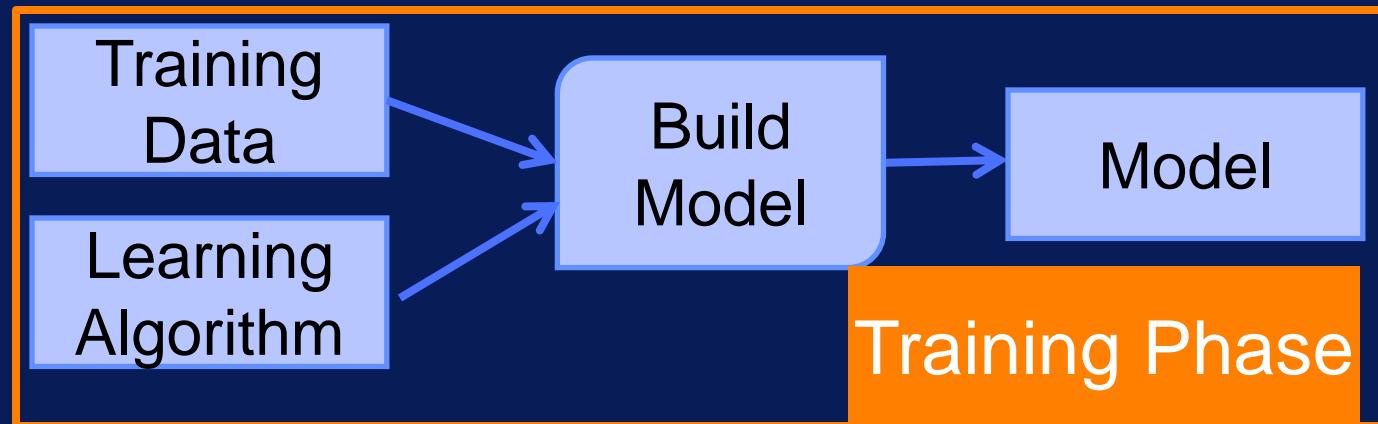
- Describe the goal of a classification algorithm
- Name some common algorithms for classification

# Classification

- **Task:** Predict category from input variables
- **Goal:** Match model outputs to targets (desired outputs)



# Learning Algorithm



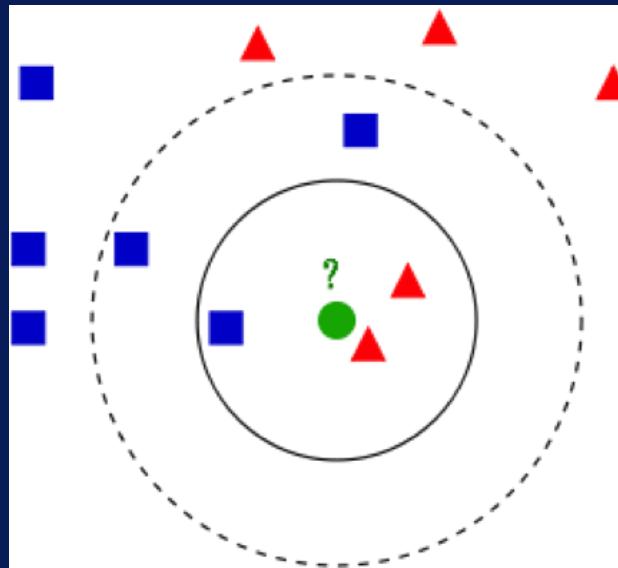
- Learning algorithm used to adjust model's parameters

# Classification Algorithms

- Common basic classification algorithms
  - kNN
  - Decision tree
  - Naïve bayes

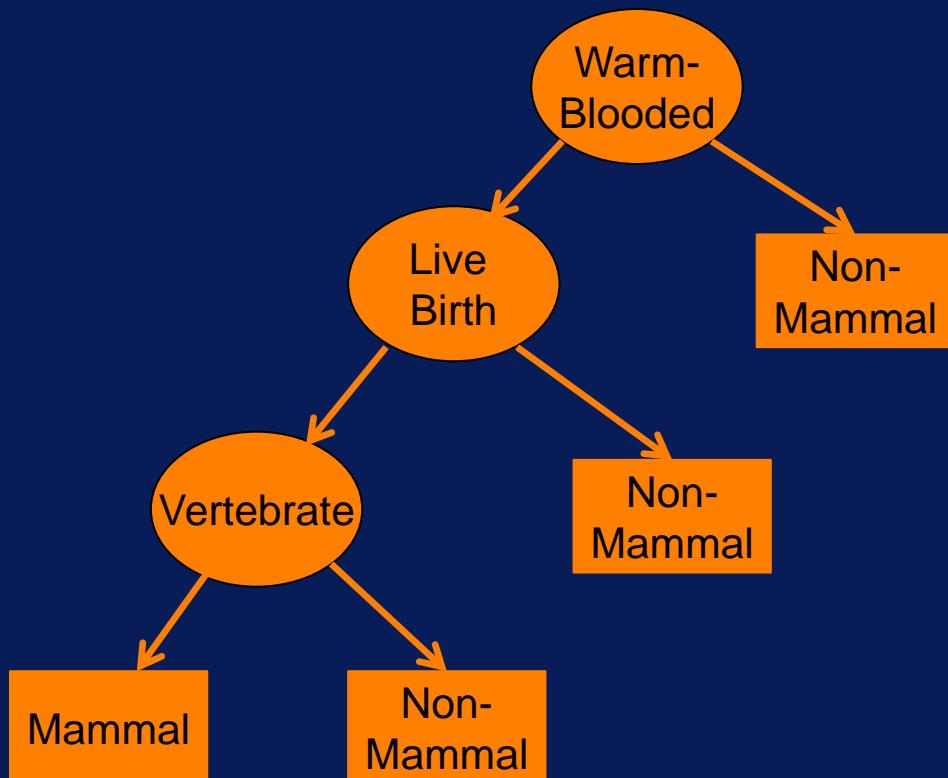
# kNN Overview

- Classify sample by looking at its closest neighbors



# Decision Tree Overview

- Tree captures multiple classification decision paths



# Naïve Bayes Overview

- Probabilistic approach to classification

*Bayes' Theorem:*

$$P(A|B) = \frac{P(B|A) P(A)}{P(B)}$$

# Classification Algorithms

k Nearest Neighbors

Decision Tree

Naïve Bayes

Many others ...

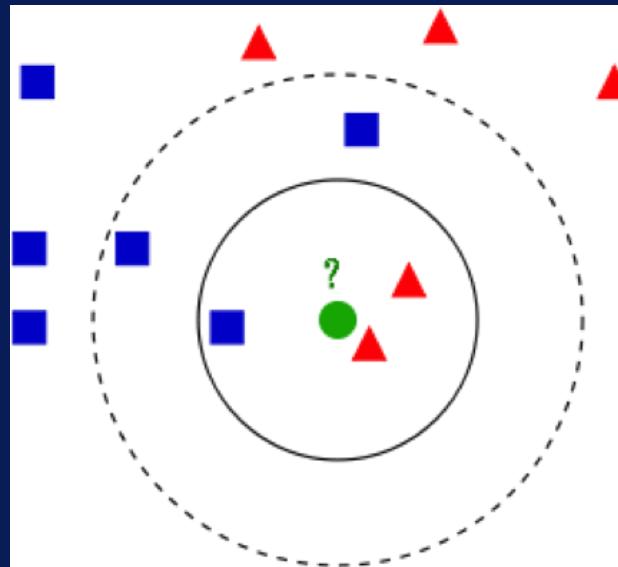
# k Nearest Neighbors

# After this video you will be able to..

- Describe how kNN is used for classification
- Discuss the assumption behind kNN
- Explain what the ‘k’ stands for in kNN

# kNN

- Simple classification technique
- Label sample based on its neighbors



# kNN Assumption

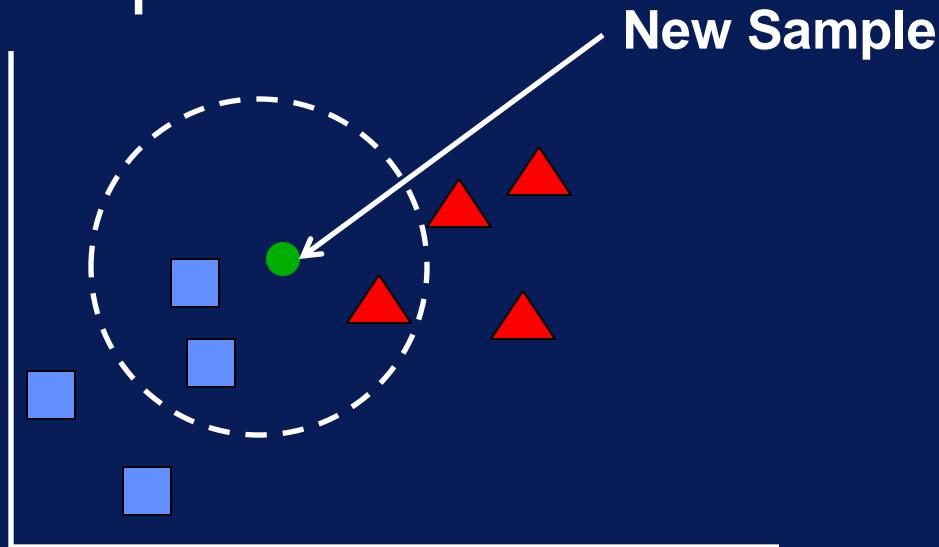
- Duck test

Quack



# How kNN Works

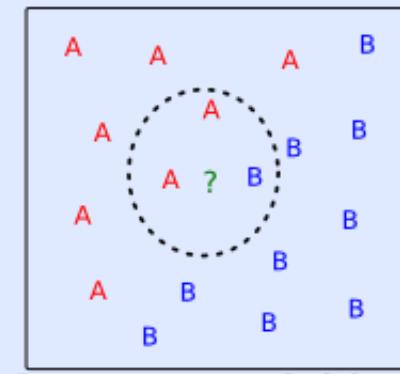
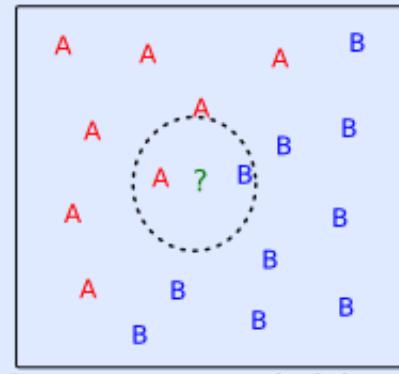
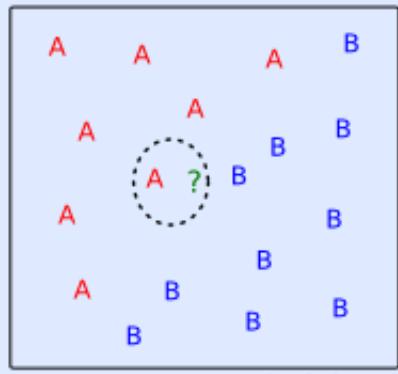
- Use labels of neighboring samples to determine label for new point



# What is k?

- Value of k determines number of closest neighbors to consider

1st, 2nd, and 3rd Nearest Neighbors  
of a Test Instance



1-nearest neighbor

2-nearest neighbor

3-nearest neighbor

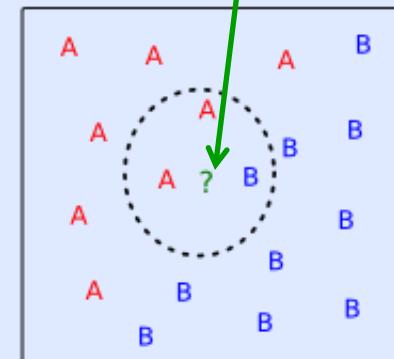
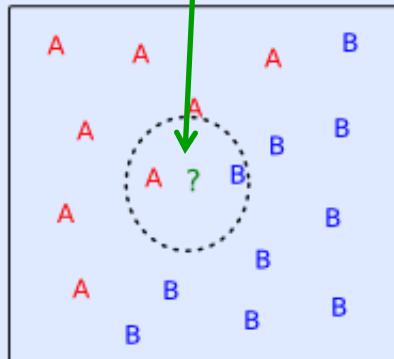
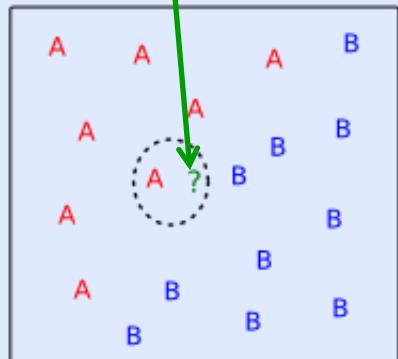
# Using k Nearest Neighbors

Label='A'  
(from  
neighbor)

Label='B'  
(random  
tiebreaker)

Label='A'  
(majority  
vote)

1st, 2nd, and 3rd Nearest Neighbors  
of a Test Instance



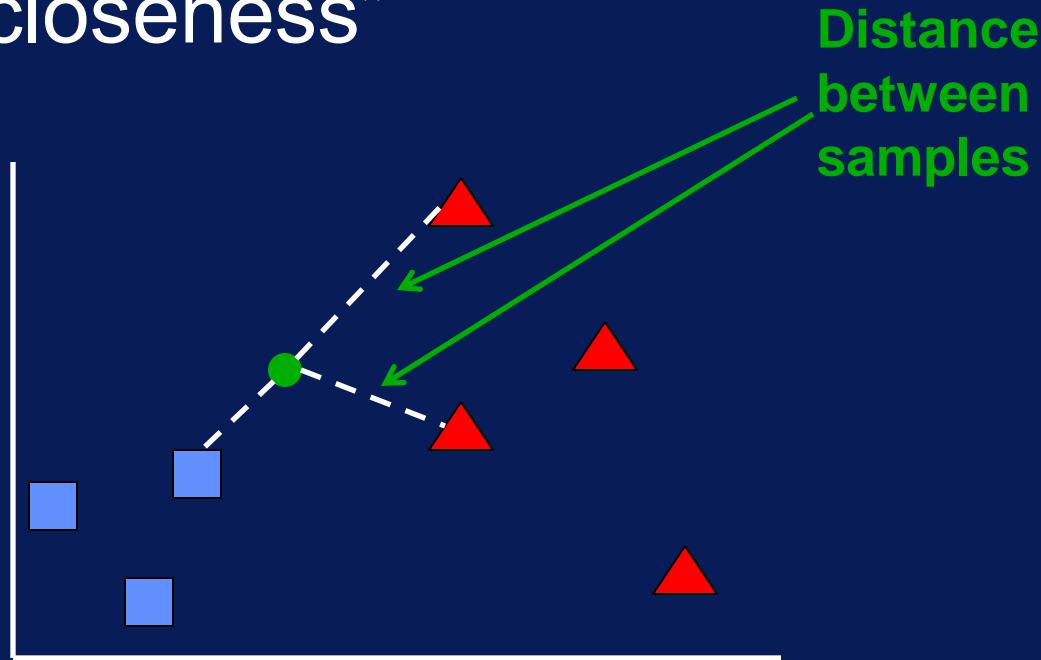
1-nearest neighbor

2-nearest neighbor

3-nearest neighbor

# Distance Measure

- Need measure to determine “closeness”



# kNN Classification

- No separate training phase
- Can generate complex decision boundaries
- Can be slow
  - Distance between new sample and all samples must be computed to classify new sample

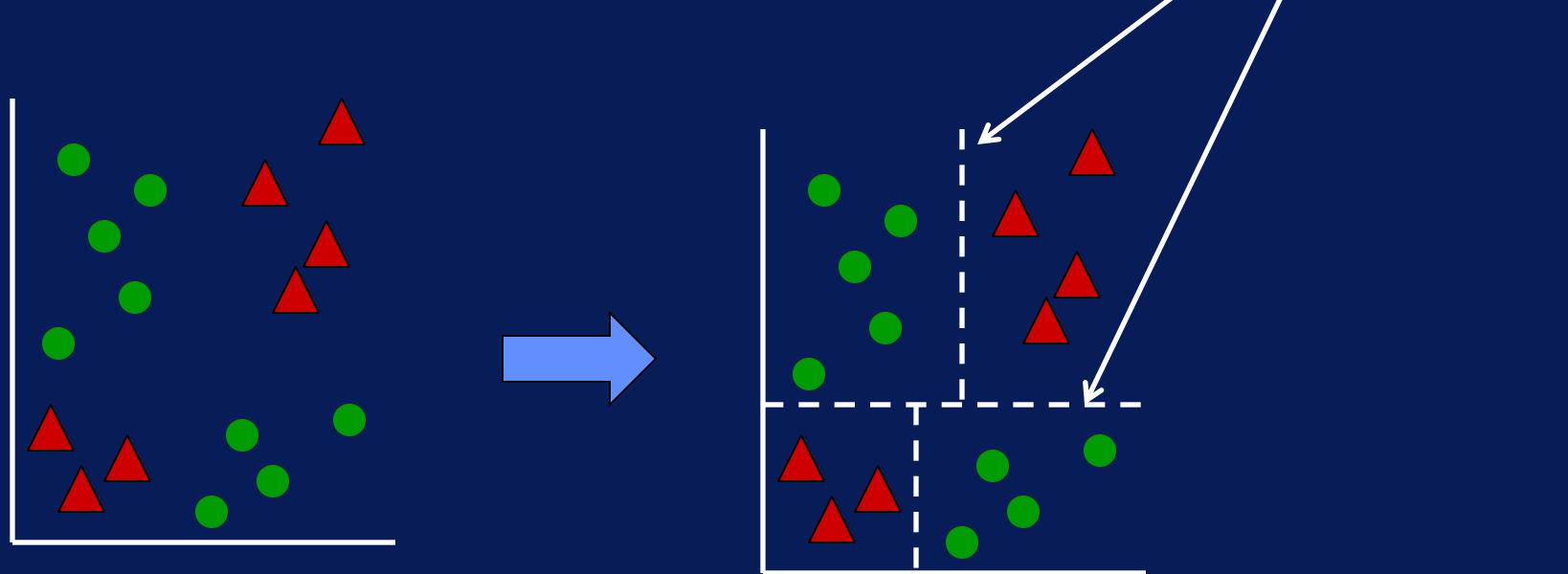
# Decision Tree

# After this video you will be able to..

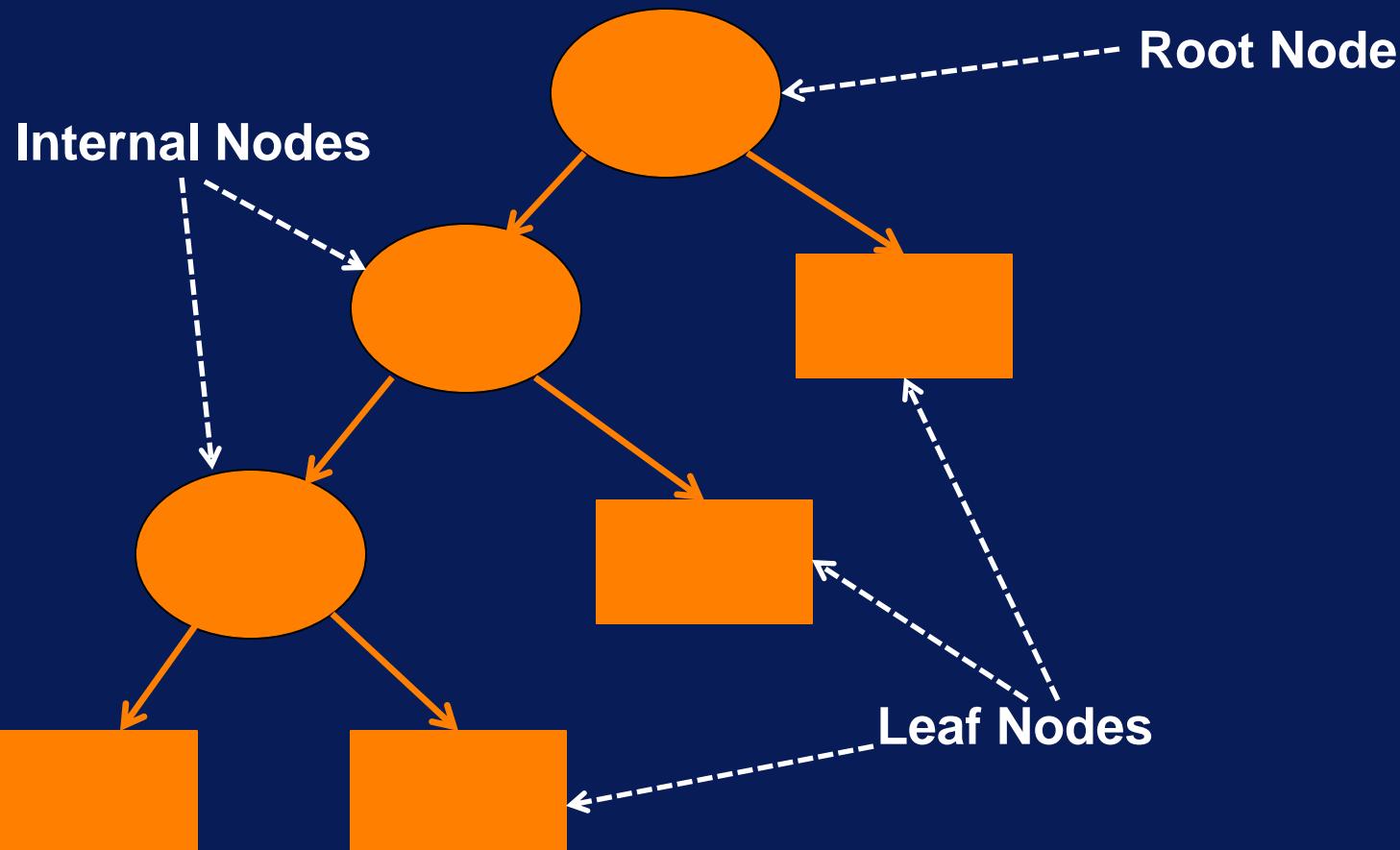
- Explain how a decision tree is used for classification
- Describe the process of constructing a decision tree for classification
- Interpret how a decision tree comes up with a classification decision

# Decision Tree Overview

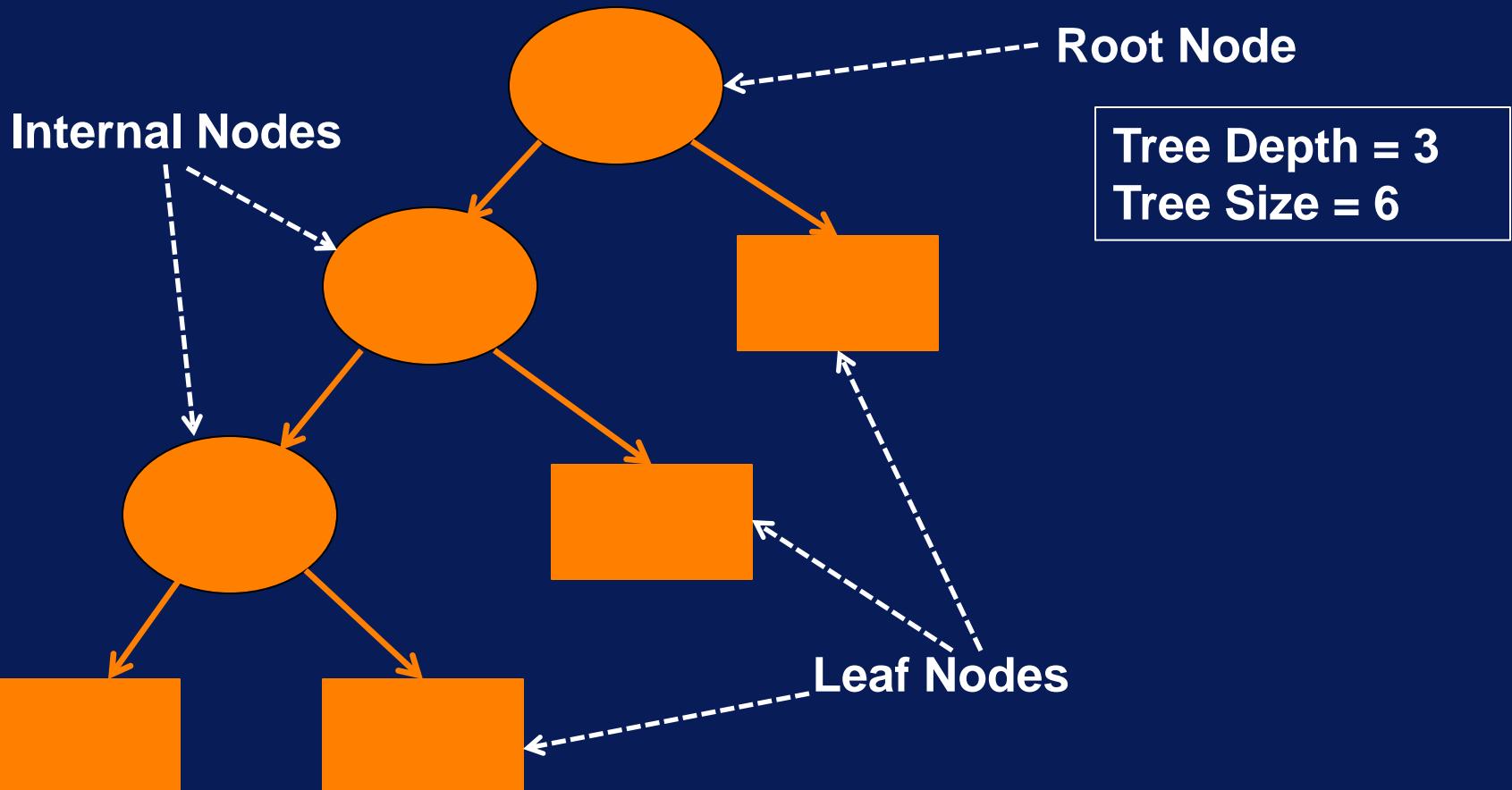
- Idea: Split data into “pure” regions



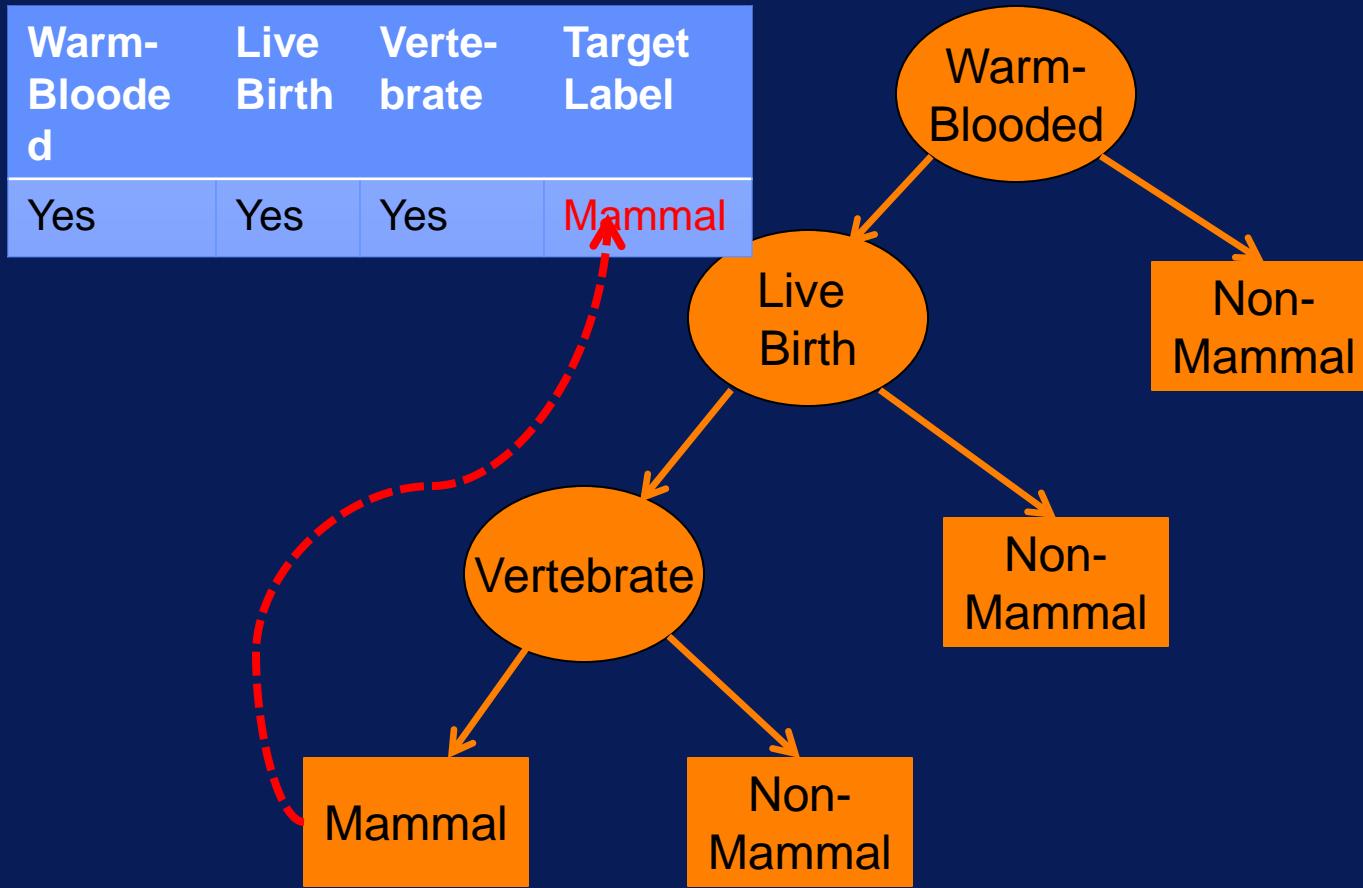
# Classification Using Decision Tree



# Classification Using Decision Tree



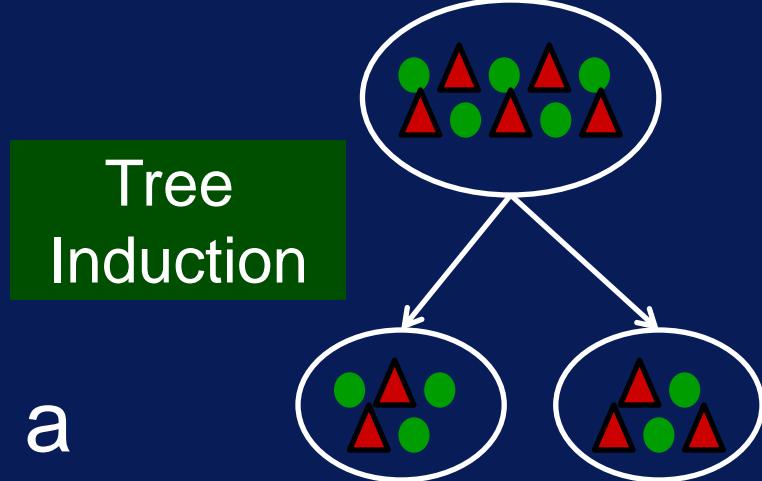
# Example Decision Tree



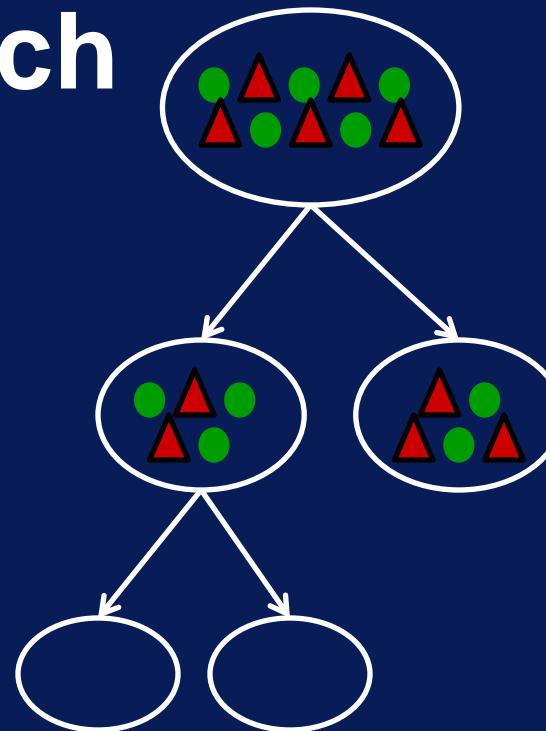
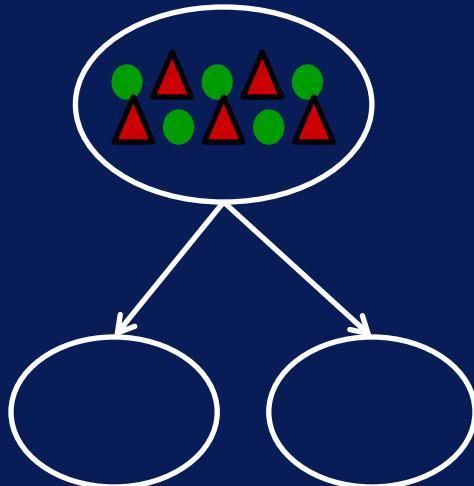
# Constructing Decision Tree

Tree  
Induction

- Start with all samples at a node.
- Partition samples based on input to create purest subsets.
- Repeat to partition data into successively purer subsets.



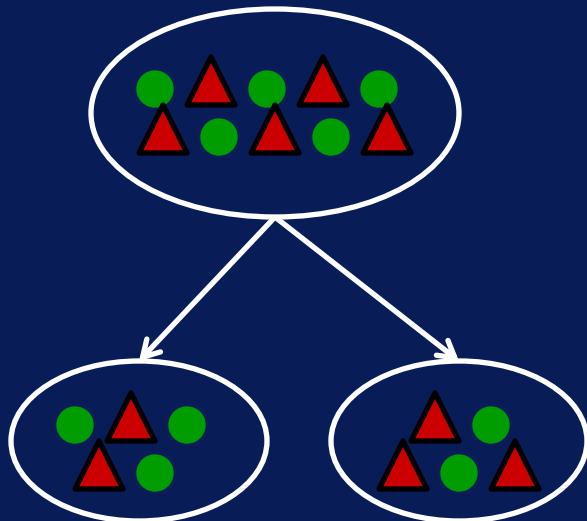
# Greedy Approach



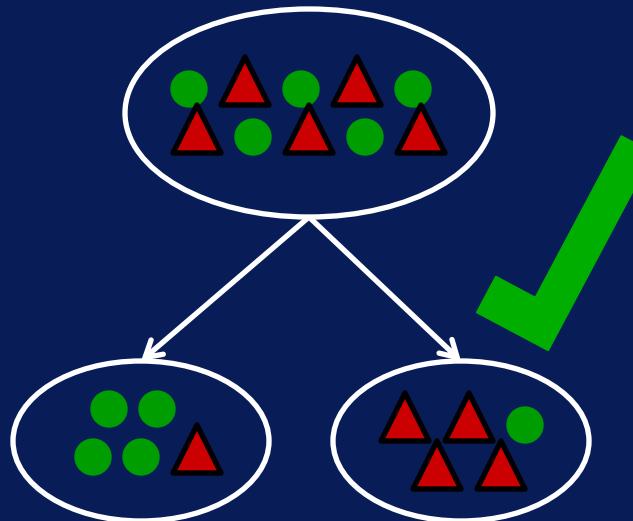
What's the best way to split the current node?

# How to Determine Best Split?

Want subsets to be as homogeneous as possible



Less homogeneous =  
More pure



More homogeneous =  
More pure

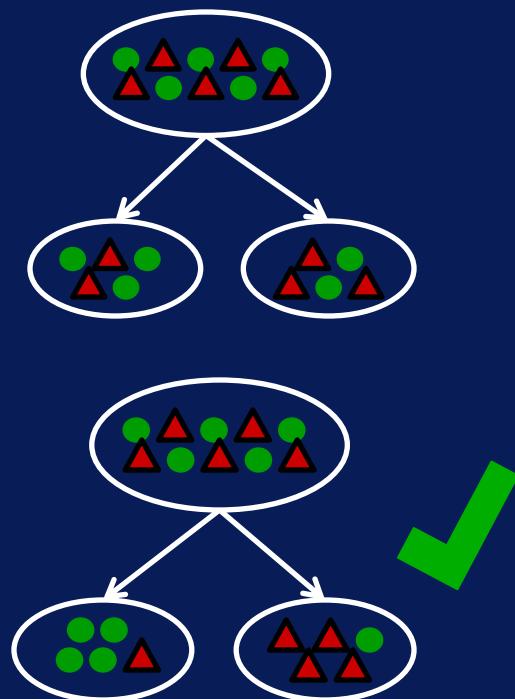
# Impurity Measure

- To compare different ways to split data in a node

Gini Index

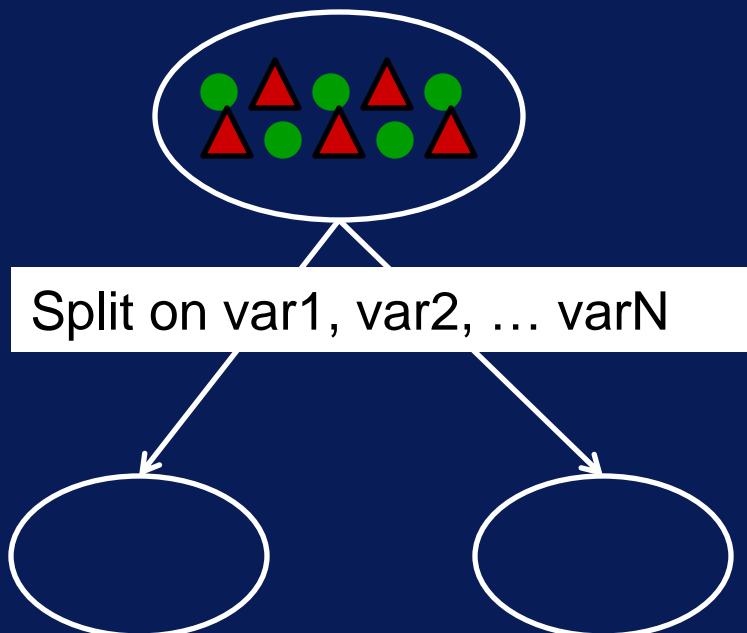
Higher = Less pure

Lower = More pure



# What Variable to Split On?

- Splits on all variables are tested



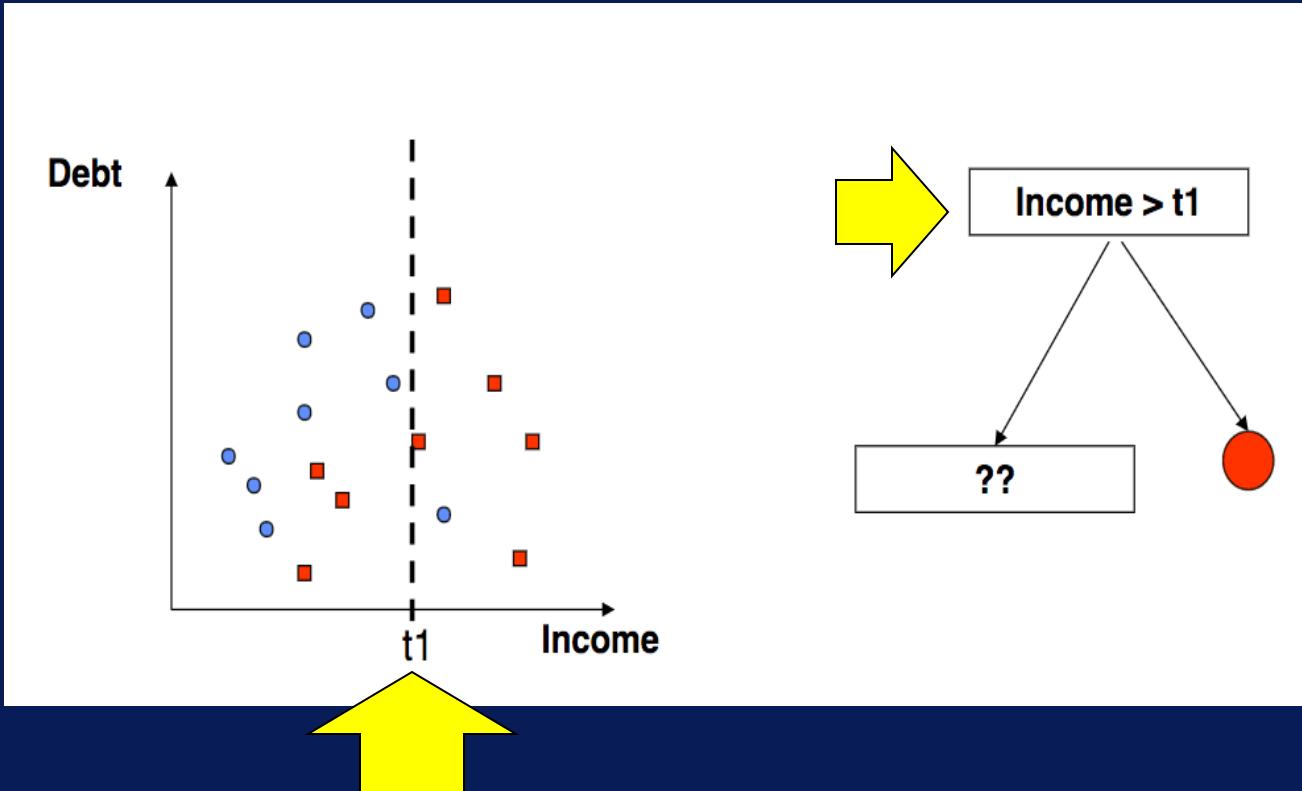
# When to Stop Splitting a Node?

- All (or X% of) samples have same class label
- Number of samples in node reaches minimum
- Change in impurity measure is smaller than threshold
- Max tree depth is reached
- Others...



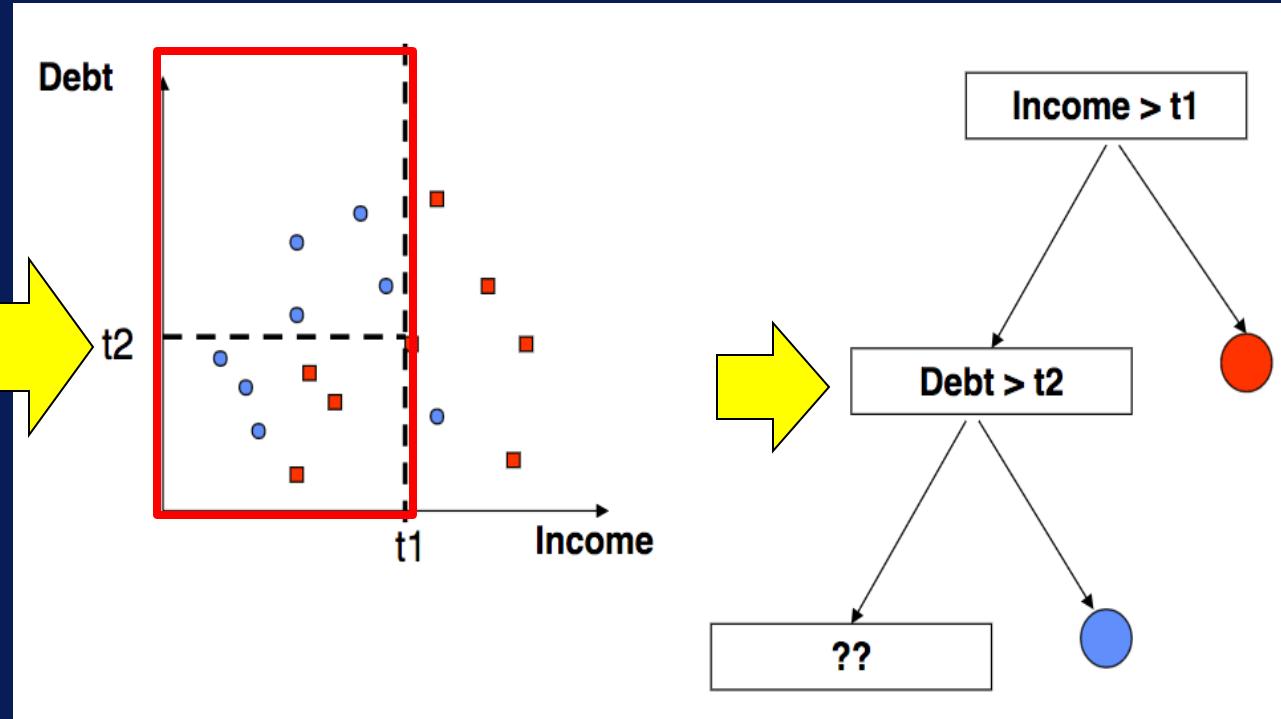
# Tree Induction Example

- Split 1



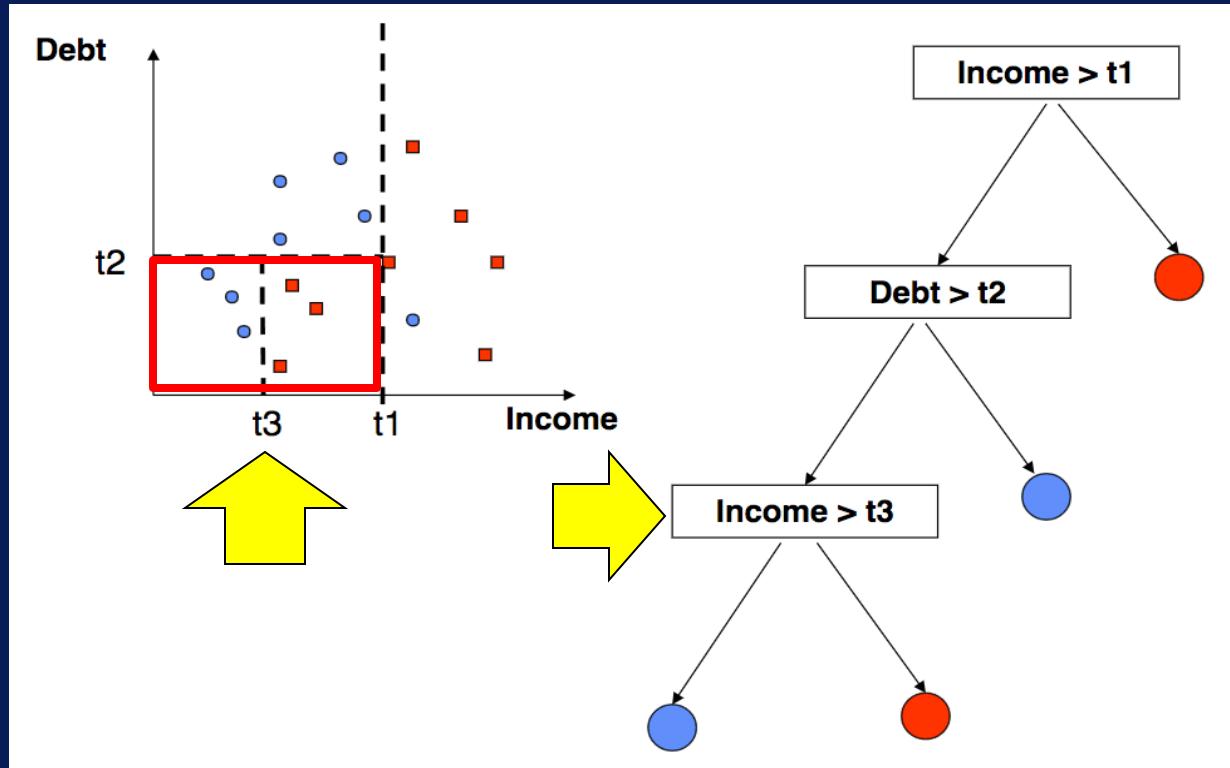
# Tree Induction Example

- Split 2



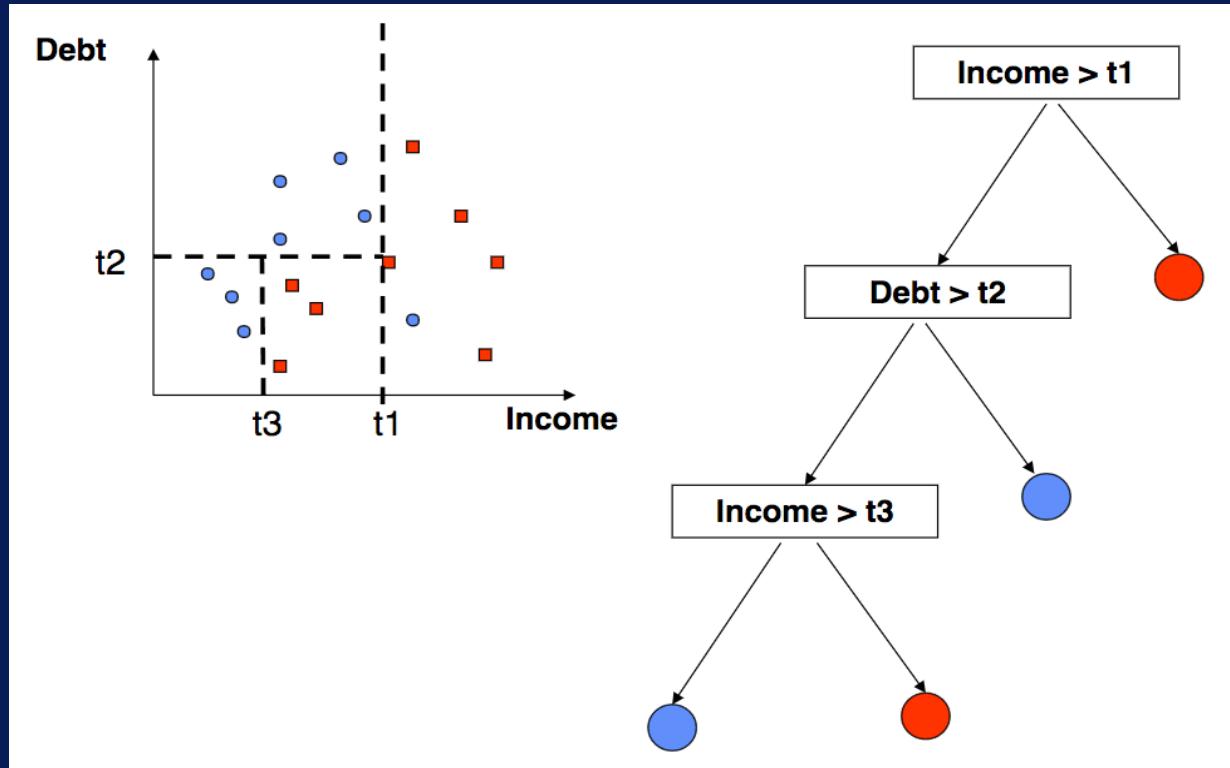
# Tree Induction Example

- # • Split 3



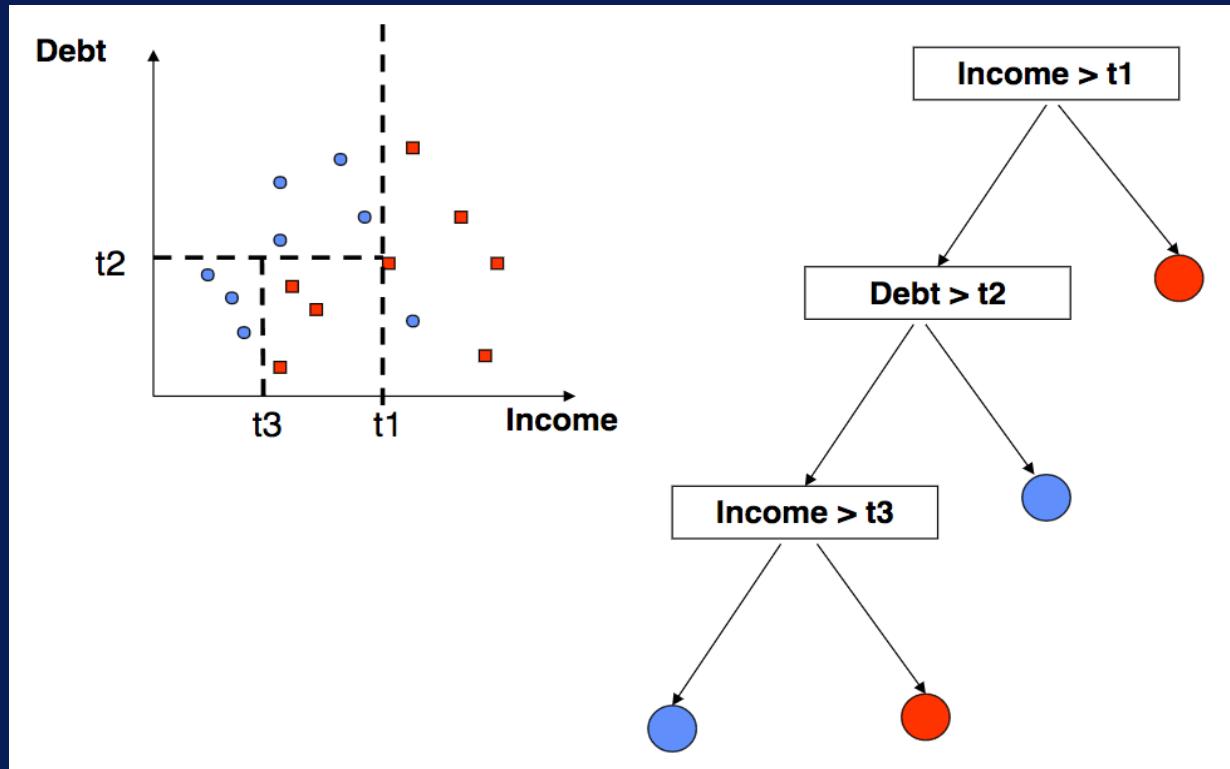
# Tree Induction Example

- Resulting model



# Decision Boundaries

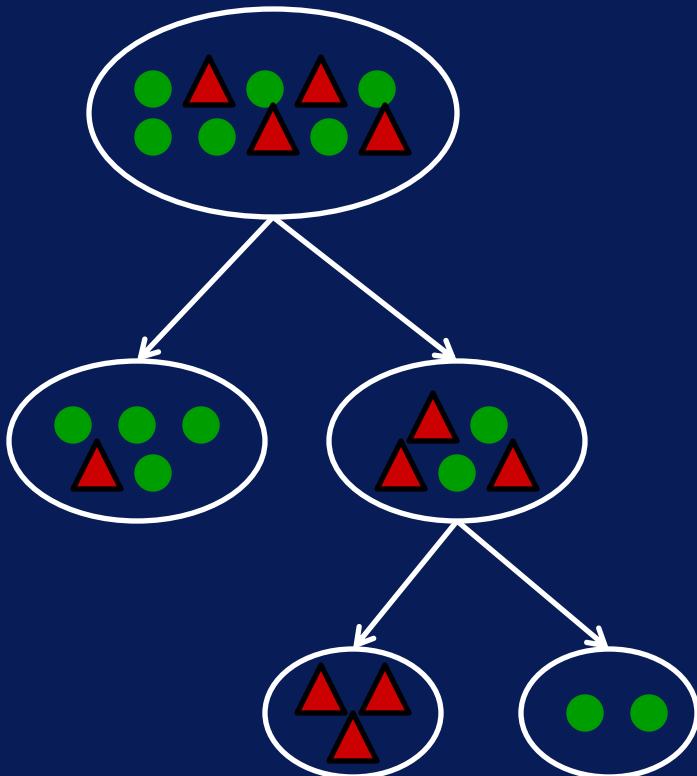
- Rectilinear = Parallel to axes



# Decision Tree for Classification

- Resulting tree is often simple and easy to interpret
- Induction is computationally inexpensive
- Greedy approach does not guarantee best solution
- Rectilinear decision boundaries

# Decision Tree



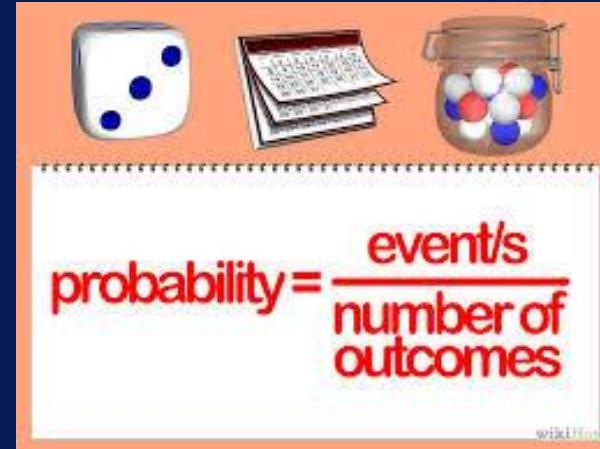
# Naïve Bayes

# After this video you will be able to..

- Discuss how a Naïve Bayes model works for classification
- Define the components of Bayes' Rule
- Explain what the ‘naïve’ means in Naïve Bayes

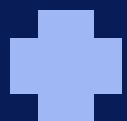
# Naïve Bayes Overview

- Probabilistic approach to classification
  - Relationships between input features and class expressed as probabilities
  - Label for sample is class with highest probability given input



# Naïve Bayes Classifier

Classification  
Using  
Probability



Bayes  
Theorem



Feature  
Independence  
Assumption

# Probability of Event

Probability is measure of how likely an event is

## Probability of Event ‘A’ Occurring

$$P(A) = \frac{\text{# ways for A}}{\text{# possible outcomes}}$$

# Probability of Event

What is the probability of rolling a die and getting 6?

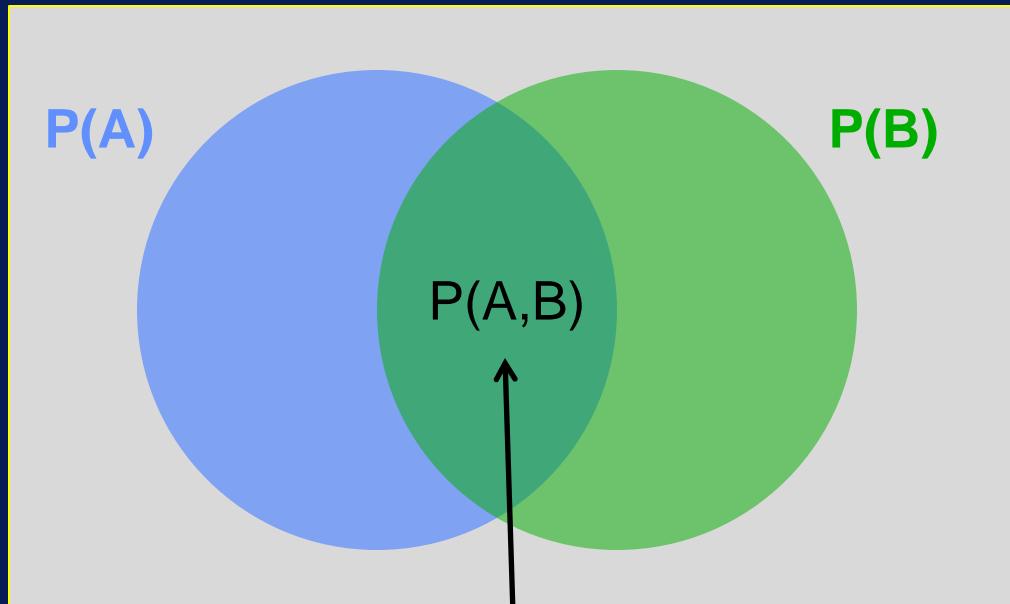


## Probability of Rolling 6 on a Die

$$P(6) = \frac{\text{\# ways for getting 6}}{\text{\# possible outcomes}} = \frac{1}{6}$$

# Joint Probability

Probability of events A and B occurring together



Joint Probability of A and B

# Joint Probability Example

What is the probability of two 6's when rolling two dice?

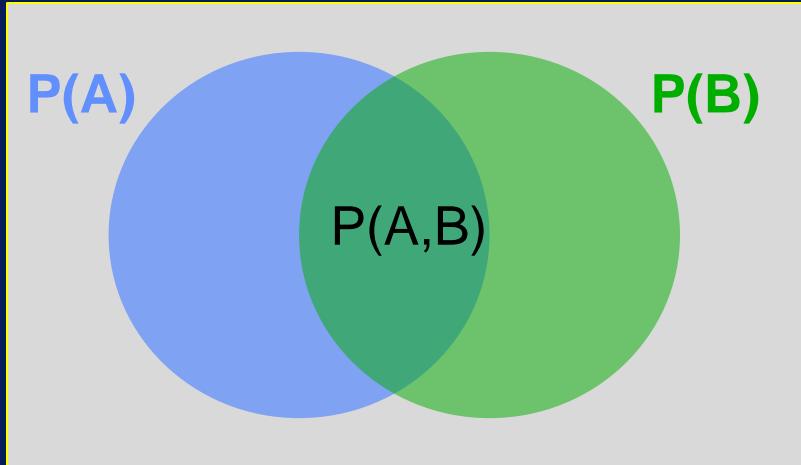


## Probability of Rolling Two 6's

$$P(A,B) = P(A) * P(B) = \frac{1}{6} * \frac{1}{6} = \frac{1}{36}$$

# Conditional Probability

Probability of event A occurring, given that event B occurred



$$P(A | B) = \frac{P(A,B)}{P(B)}$$

**Conditional  
Probability**

# Bayes' Theorem

- Relationship between  $P(B | A)$  and  $P(A | B)$  can be expressed through Bayes' Theorem

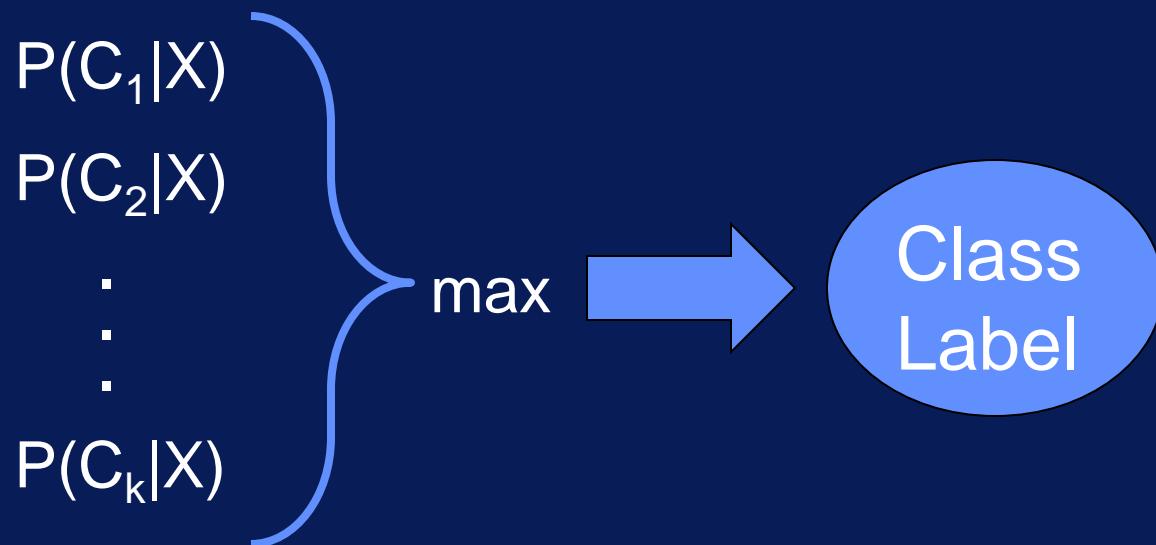
$$P(B | A) = \frac{P(A | B) * P(B)}{P(A)}$$

Bayes' Theorem

# Classification with Probabilities

Given features  $X = \{X_1, X_2, \dots, X_n\}$ , predict class C

Do this by finding value of C that maximizes  $P(C | X)$



# Bayes' Theorem for Classification

- But estimating  $P(C|X)$  is difficult
- Bayes' Theorem to the rescue!
  - Simplifies problem



# Bayes' Theorem for Classification

Posterior  
Probability

Class-  
Conditional  
Probability

Prior  
Probability

$$P(C | X) = \frac{P(X | C) * P(C)}{P(X)}$$

Probability of observing  
values for input features

# Bayes' Theorem for Classification

Need to calculate this

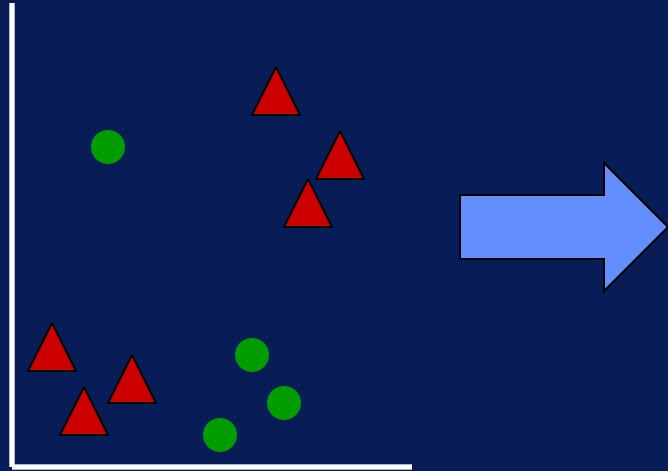
Can be estimated from data!

$$P(C | X) = \frac{P(X | C) * P(C)}{P(X)}$$

Constant (can be ignored)

To get  $P(C | X)$ , only need to find  $P(X | C)$  and  $P(C)$ , which can be estimated from the data!

# Estimating P(C)



$$P(\bullet) = 4/10 = 0.4$$

$$P(\blacktriangle) = 6/10 = 0.6$$

To estimate  $P(C)$ , calculate fraction of samples for class C in training data.

# Estimating $P(X | C)$

## Independence Assumption

- Features are independent of one another:

$$P(X_1, X_2, \dots, X_n | C) = P(X_1 | C) * P(X_2 | C) * \dots * P(X_n | C)$$

To estimate  $P(X | C)$ , only need to estimate  
 $P(X_i | C)$  individually → Much simpler!

# Estimating $P(X_i | C)$

Home Owner	Marital Status	Loan Default
Yes	Single	No
No	Married	No
No	Single	No
Yes	Married	No
No	Divorced	Yes
No	Married	No
Yes	Divorced	No
No	Single	Yes
No	Married	No
No	Single	Yes

$$P(\text{Home Owner} = \text{Yes} | \text{No}) = 3/7 = 0.43$$

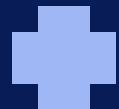
$$P(\text{Marital Status} = \text{Single} | \text{Yes}) = 2/3 = 0.67$$

# Naïve Bayes Classification

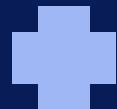
- **Fast and simple**
- **Scales well**
- **Independence assumption may not hold true**
  - In practice, still works quite well
- **Does not model interactions between features**

# Naïve Bayes Classifier

Classification  
Using  
Probability



Bayes  
Theorem



Feature  
Independence  
Assumption

# Classification Using Decision Tree in KNIME

## Learning Objectives

By the end of this activity, you will be able to perform the following operations in KNIME:

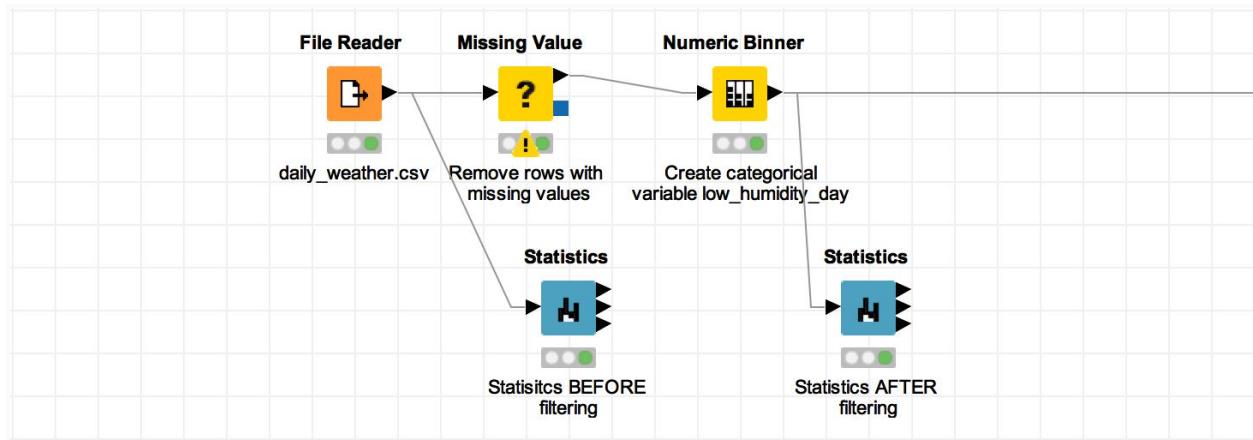
1. Create a categorical variable from a numeric variable
2. Examine the summary statistics of a dataset
3. Build a workflow for a classification task using a decision tree

## Problem Description

Now that we have explored the data and looked at how to handle missing values, the next step is to build a classification model to predict days with low humidity. Recall that low humidity is one of the weather conditions that increase the dangers of wildfires, so it would be helpful to be able to predict low-humidity days. We will build a decision model to classify low-humidity days vs. non-low-humidity days based on weather conditions observed at 9am.

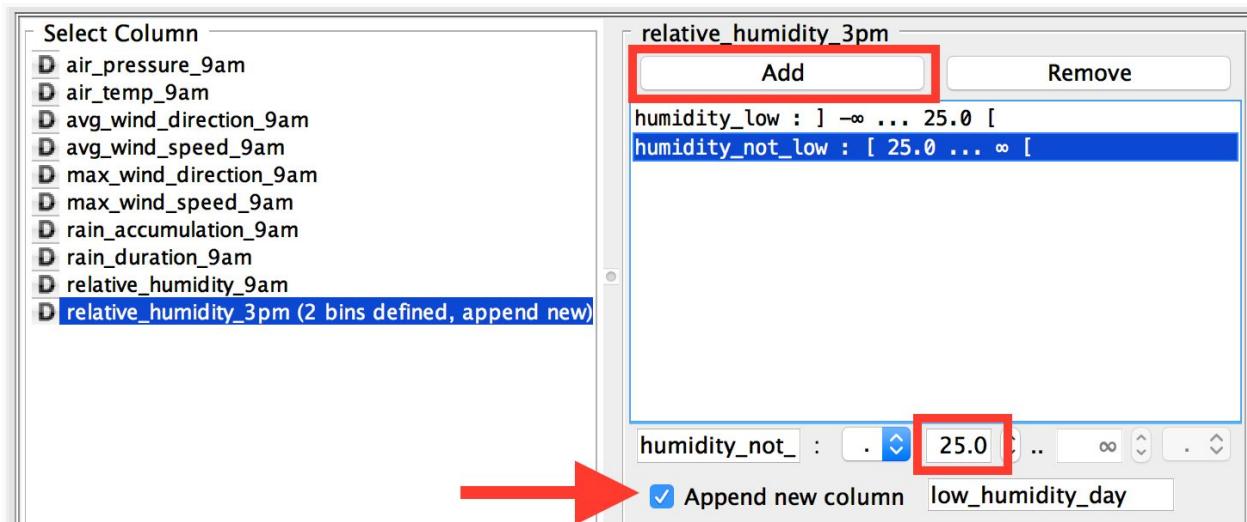
## Steps

### Prepare the data



Let's build a workflow to build a decision tree model to classify low-humidity days vs. days with normal or high humidity. The model will be used to predict low-humidity days

1. Start a new workflow in your local workspace.
2. Use a **File Reader** node to import the daily\_weather.csv dataset. Use the configuration dialog to specify the location of the daily\_weather.csv file.
3. Connect a **Missing Value** node to the File Reader node. This will handle the missing values that the dataset contains so the data can be analyzed properly. In the configure dialog, in the **Default** tab, choose **Remove Row\*** to remove all rows with missing values.
4. As with the Data Exploration Hands-On, use the **Numeric Binner** node to create a new categorical variable with the condition "*if relative\_humidity\_3pm < 25% then humidity\_low is true, else humidity\_not\_low is true*".
  - Locate the **Numeric Binner** node, which is in the Manipulation>Column>Binning category. Drag it to the Workflow Editor, and connect it to the **Missing Value** node.
  - Open the Configure Dialog for the Numeric Binner node. Select **relative\_humidity\_9am**, and **Add 2 bins**. Make one bin called "humidity\_low" with the range **-∞ to 25** excluding 25, and another called "humidity\_not\_low" with the range **25 to ∞**. The endpoint brackets specify that humidity\_low excludes 25.0, while humidity\_not\_low includes 25.0. This is necessary to capture the condition "*if relative\_humidity\_3pm < 25% then low\_humidity\_day=1, else low\_humidity\_day=0*". Click the checkbox to "**Append new column**" and name it **low\_humidity\_day**.



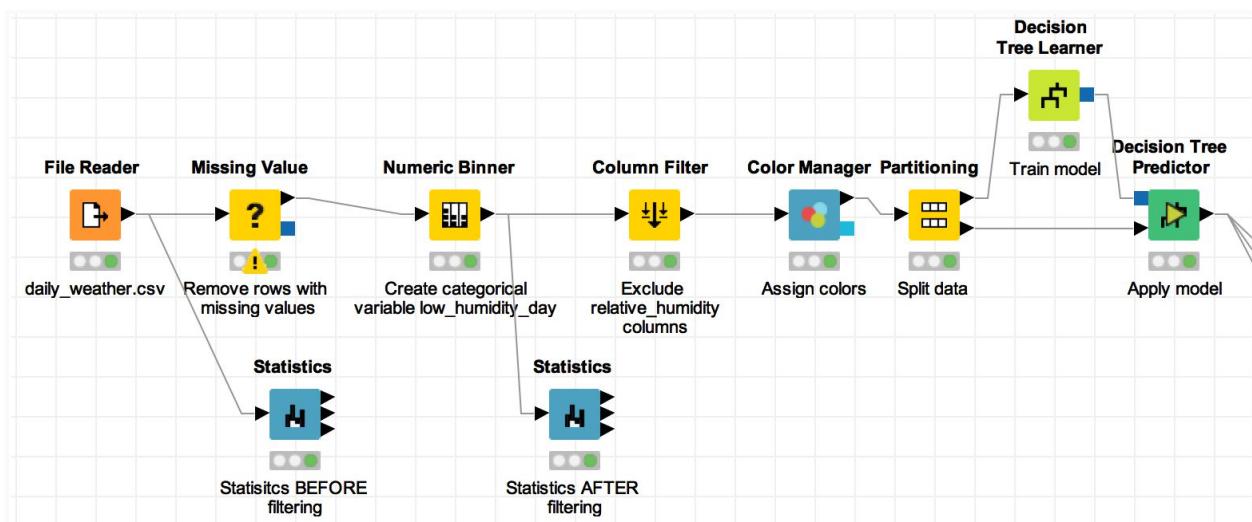
## Examine Summary Statistics

Before we build the workflow any further, let's use some **Statistics** nodes to check a few things about our processed data.

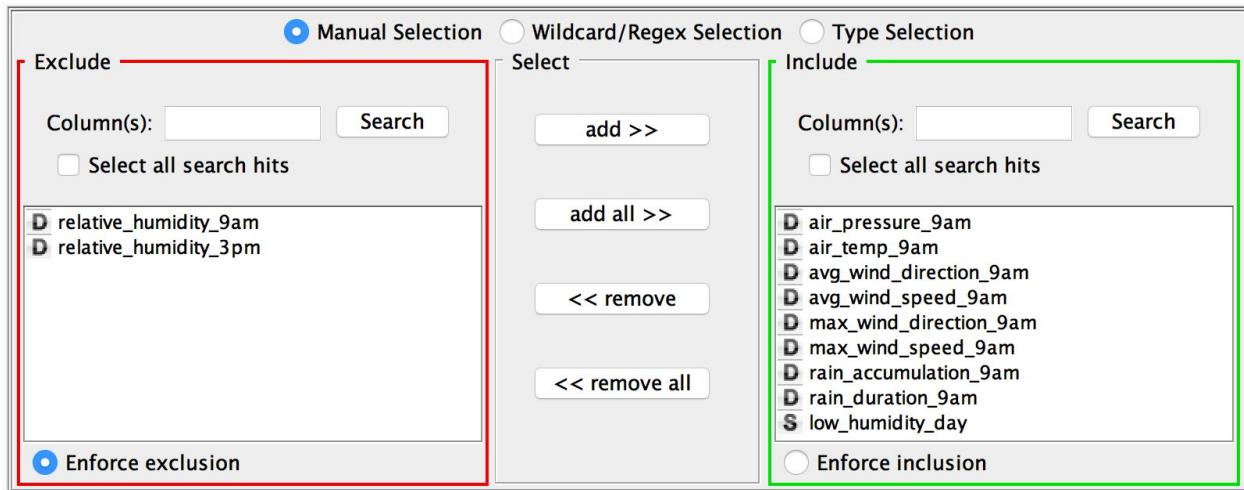
1. Connect a **Statistics** node to the output of the **File Reader** node. In the Configure Dialog of the Statistics node, change **Max no. of possible values per column (in output table)** to **1,500**, and **add all >>** columns to the **Includeside**. Rename this node to "Statistics BEFORE filtering".
2. Connect a **Statistics** node to the output of the **Numeric Binner** node. In the Configure Dialog of this Statistics node, change **Max no. of possible values per column (in output table)** to **1,500**, and **add all >>** columns to the **Includeside**. Rename this node to "Statistics AFTER filtering".
3. Execute and view both Statistics nodes, and you should see the resulting histograms have the same features in both. This is to ensure that the way we handled the missing values did not skew our data. Now we can check the following:
4. There are missing values for many of the variables in the "Statistics BEFORE filtering" node, but zero missing values in the "Statistics AFTER filtering" node.
5. The distributions of each variable in both "Statistics BEFORE filtering" and "Statistics AFTER filtering" should be about the same. You can spot check a couple of variables by looking at histograms, min, max, mean, and standard deviation.
6. In the "Statistics AFTER filtering" node, look at the **Nominal** tab to see the distribution of **low\_humidity\_day**. This shows that the samples are equally distributed between low-humidity days and days with normal or high humidity.



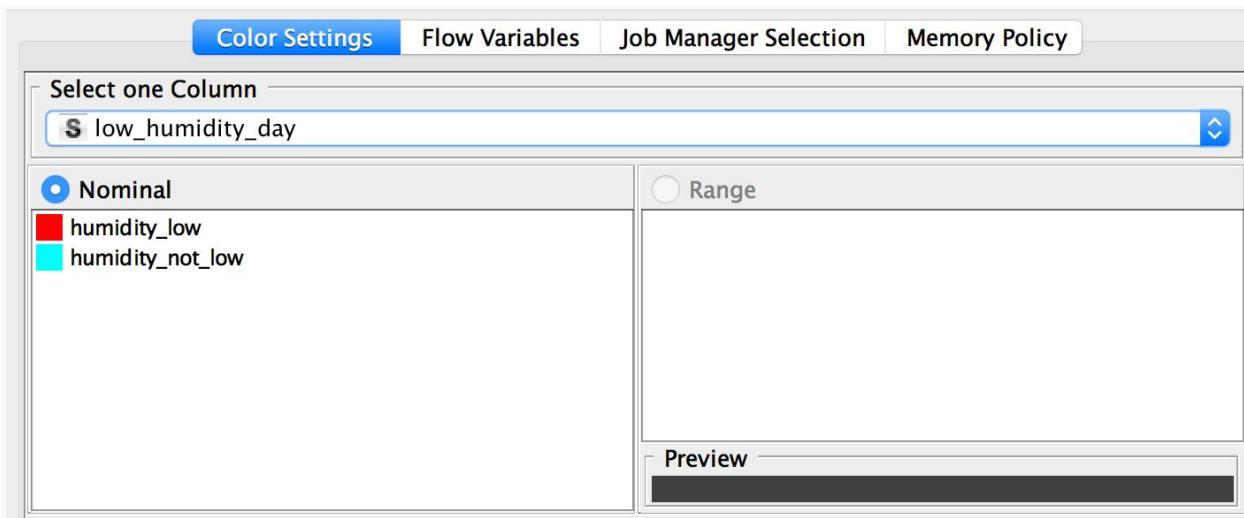
## Build a Decision Tree Workflow



1. Connect a **Column Filter** node to the **Numeric Binner** node. In the Configure Dialog of the Column Filter node, exclude only the **relative\_humidity\_9am** and **relative\_humidity\_3pm** columns.

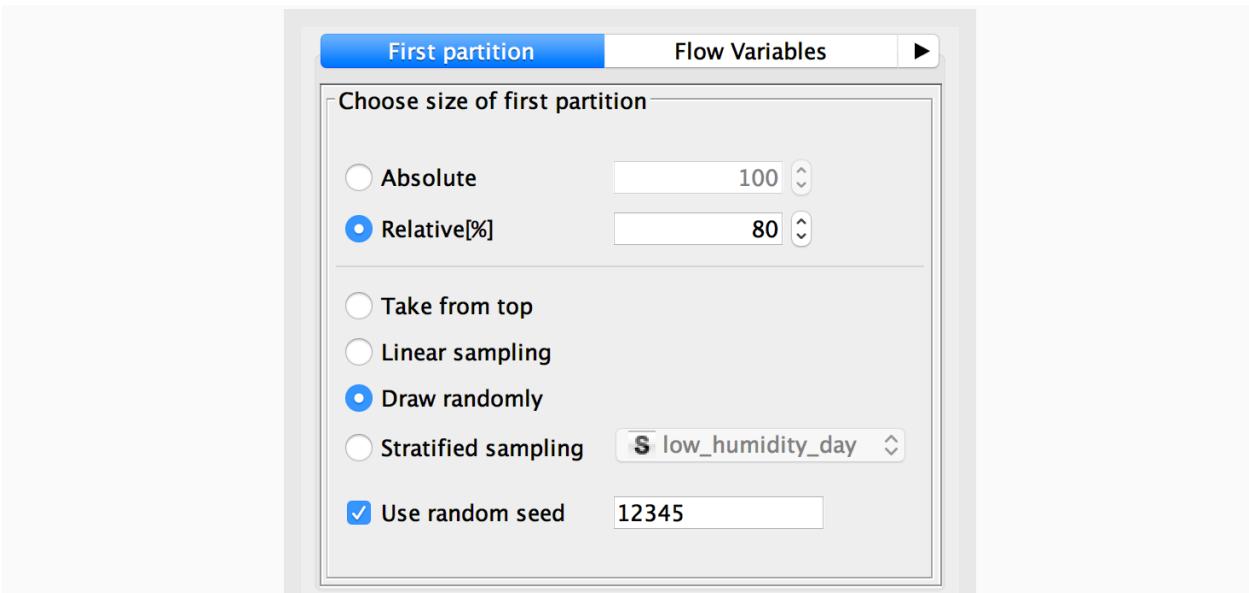


2. Connect a **Color Manager** node to the **Column Filter** node. This will color-code our categorical **low\_humidity\_day** variable so it is easier to visualize later on in the workflow. In the Configure Dialog of the Color Manager node, check that for **low\_humidity\_day**, the **humidity\_low** is colored red and the **humidity\_not\_low** is colored blue.

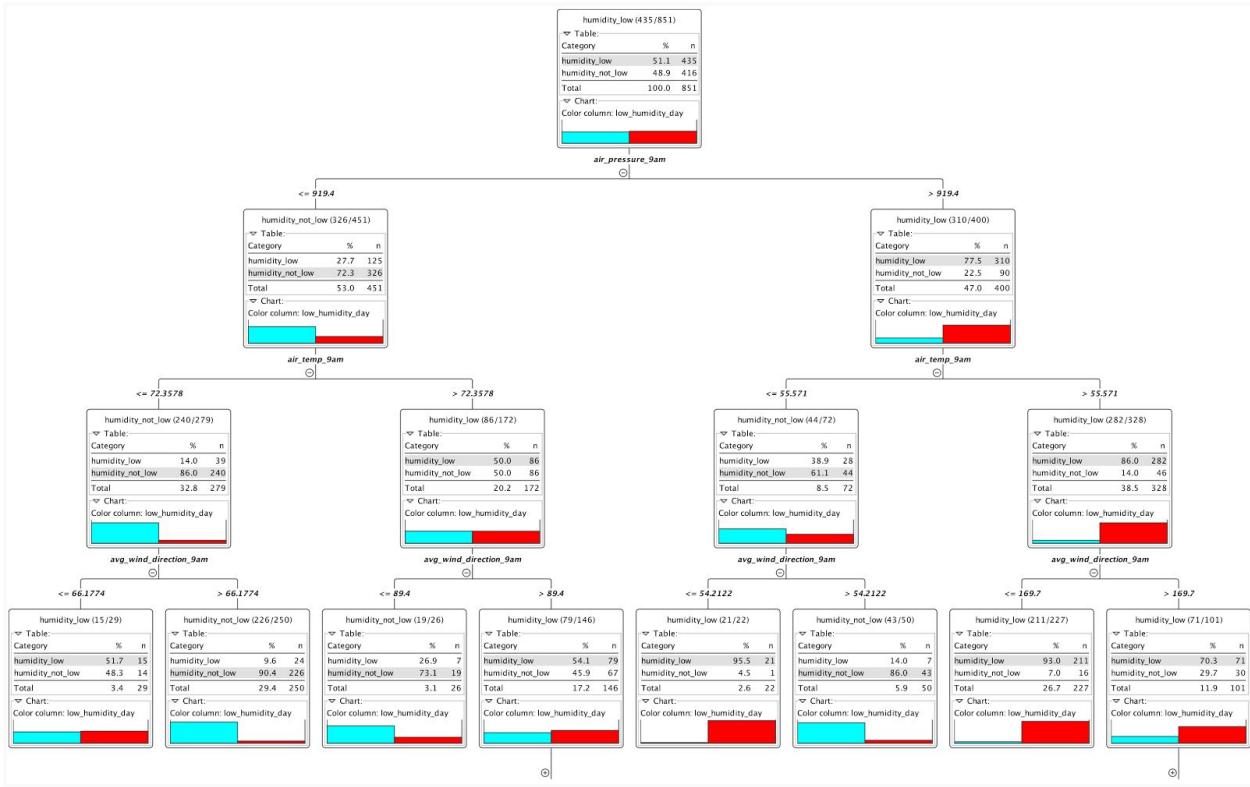


3. Connect a **Partitioning** node to the **Color Manager** node. The Partitioning node is needed to split the data into training and testing portions. Training data is used to build the decision tree, and test data is used to evaluate the classifier on new data. In the Configure Dialog of the Partitioning node, choose **Relative[%] 80**, **Draw Randomly**, and **Use random seed 12345**. This will randomly select 80% of the data to the 1st output (the training data), and the rest to the 2nd output (the test data).

The random seed is set so that everyone can get the same training and test data sets to train and test the decision tree model for this exercise.



4. Connect a **Decision Tree Learner** node to the 1st output of the **Partitioning** node. This node will generate the decision tree classifier using the training data. In the Configure Dialog, change the **Min number of records per node** to **20**. This is a stopping criterion for the tree induction algorithm. It specifies that a node with this number of samples can no longer be split. The default value for this is 2, which is very small, and may result in overfitting.
5. Connect a **Decision Tree Predictor** to the 2nd output of the **Partitioning** node and to the output from the **Decision Tree Learner** node. This node will apply the model to the test data.
6. Execute the workflow. Right-click on the **Decision Tree Predictor** node and select 'View: Decision Tree View' to see the generated decision tree. You can zoom out and click the little '+' buttons to expand the nodes. The next Reading, **Interpreting a Decision Tree in KNIME**, describes how to interpret the resulting decision tree model.



## Save Your Workflow

Save your workflow using **<control>-s** on Windows or **<command>-s** on Mac, or selecting **File>Save** or **File>Save As**.

# Classification in Spark

By the end of this activity, you will be able to perform the following in Spark:

1. Generate a categorical variable from a numeric variable
2. Aggregate the features into one single column
3. Randomly split the data into training and test sets
4. Create a decision tree classifier to predict days with low humidity.

In this activity, you will be programming in a Jupyter Python Notebook. If you have not already started the Jupyter Notebook server, see the instructions in the Reading *Instructions for Starting Jupyter*.

Step 1. **Open Jupyter Python Notebook.** Open a web browser by clicking on the web browser icon at the top of the toolbar:



Navigate to `localhost:8889/tree/Downloads/big-data-4`:



Open the handling missing values notebook by clicking on `classification.ipynb`:



Step 2. **Load classes and data.** Execute the first cell in the notebook to load the classes used for this exercise.

```
In [1]: from pyspark.sql import SQLContext
from pyspark.sql import DataFrameNaFunctions
from pyspark.ml import Pipeline
from pyspark.ml.classification import DecisionTreeClassifier
from pyspark.ml.feature import Binarizer
from pyspark.ml.feature import VectorAssembler, StringIndexer, VectorIndexer
```

Next, execute the second cell which loads the weather data into a DataFrame and prints the columns.

```
In [2]: sqlContext = SQLContext(sc)
df = sqlContext.read.load('file:///home/cloudera/Downloads/big-data-4/daily_weather.csv',
                           format='com.databricks.spark.csv',
                           header='true',inferSchema='true')
df.columns

Out[2]: ['number',
          'air_pressure_9am',
          'air_temp_9am',
          'avg_wind_direction_9am',
          'avg_wind_speed_9am',
          'max_wind_direction_9am',
          'max_wind_speed_9am',
          'rain_accumulation_9am',
          'rain_duration_9am',
          'relative_humidity_9am',
          'relative_humidity_3pm']
```

Execute the third cell, which defines the columns in the weather data we will use for the decision tree classifier.

```
In [3]: featureColumns = ['air_pressure_9am','air_temp_9am','avg_wind_direction_9am','avg_wind_speed_9am'
                           'max_wind_direction_9am','max_wind_speed_9am','rain_accumulation_9am',
                           'rain_duration_9am']
```

Step 3. **Drop unused and missing data.** We do not need the *number* column in our data, so let's remove it from the DataFrame:

```
In [4]: df = df.drop('number')
```

Next, let's remove all rows with missing data:

```
In [5]: df = df.na.drop()
```

We can print the number of rows and columns in our DataFrame:

```
In [6]: df.count(), len(df.columns)

Out[6]: (1064, 10)
```

Step 4. **Create categorical variable.** Let's create a categorical variable to denote if the humidity is not low. If the value is less than 25%, then we want the categorical value to be 0, otherwise the

categorical value should be 1. We can create this categorical variable as a column in a DataFrame using *Binarizer*:

```
In [7]: binarizer = Binarizer(threshold=24.99999, inputCol="relative_humidity_3pm", outputCol="label")
binarizedDF = binarizer.transform(df)
```

The *threshold* argument specifies the threshold value for the variable, *inputCol* is the input column to read, and *outputCol* is the name of the new categorical column. The second line applies the *Binarizer* and creates a new DataFrame with the categorical column. We can look at the first four values in the new DataFrame:

```
In [8]: binarizedDF.select("relative_humidity_3pm", "label").show(4)
```

relative_humidity_3pm	label
36.160000000000494	1.0
19.4265967985621	0.0
14.460000000000045	0.0
12.742547353761848	0.0

only showing top 4 rows

The first row's humidity value is greater than 25% and the label is 1. The other humidity values are less than 25% and have labels equal to 0.

Step 5. **Aggregate features.** Let's aggregate the features we will use to make predictions into a single column:

```
In [9]: assembler = VectorAssembler(inputCols=featureColumns, outputCol="features")
assembled = assembler.transform(binarizedDF)
```

The *inputCols* argument specifies our list of column names we defined earlier, and *outputCol* is the name of the new column. The second line creates a new DataFrame with the aggregated features in a column.

Step 6. **Split training and test data.** We can split the data by calling *randomSplit()*:

```
In [10]: (trainingData, testData) = assembled.randomSplit([0.8,0.2], seed = 13234 )
```

The first argument is how many parts to split the data into and the *approximate* size of each. This specifies two sets of 80% and 20%. Normally, the seed should not be specified, but we use a specific value here so that everyone will get the same decision tree.

We can print the number of rows in each DataFrame to check the sizes ( $1095 * 80\% = 851.2$ ):

```
In [11]: trainingData.count(), testData.count()  
Out[11]: (854, 210)
```

Step 7. **Create and train decision tree.** Let's create the decision tree:

```
In [12]: dt = DecisionTreeClassifier(labelCol="label", featuresCol="features", maxDepth=5,  
minInstancesPerNode=20, impurity="gini")
```

The *labelCol* argument is the column we are trying to predict, *featuresCol* specifies the aggregated features column, *maxDepth* is stopping criterion for tree induction based on maximum depth of tree, *minInstancesPerNode* is stopping criterion for tree induction based on minimum number of samples in a node, and *impurity* is the impurity measure used to split nodes.

We can create a model by training the decision tree. This is done by executing it in a *Pipeline*:

```
In [13]: pipeline = Pipeline(stages=[dt])  
model = pipeline.fit(trainingData)
```

Let's make predictions using our test data set:

```
In [14]: predictions = model.transform(testData)
```

Looking at the first ten rows in the prediction, we can see the prediction matches the input:

```
In [15]: predictions.select("prediction", "label").show(10)
```

prediction	label
1.0	1.0
1.0	1.0
1.0	1.0
1.0	1.0
1.0	1.0
1.0	1.0
1.0	1.0
0.0	0.0
1.0	1.0
1.0	1.0
1.0	1.0

only showing top 10 rows

Step 8. **Save predictions to CSV.** Finally, let's save the predictions to a CSV file. In the next Spark hands-on activity, we will evaluate the accuracy.

Let's save only the *prediction* and *label* columns to a CSV file:

```
In [16]: predictions.select("prediction", "label").write.save(path="file:///home/cloudera/Downloads/big-data-4/predictions.csv",
format="com.databricks.spark.csv",
header='true')
```

# Interpreting a Decision Tree in KNIME

This document describes how to interpret a decision tree classifier. We will use the tree created in the Classification Using Decision Tree in KNIME Hands-On.

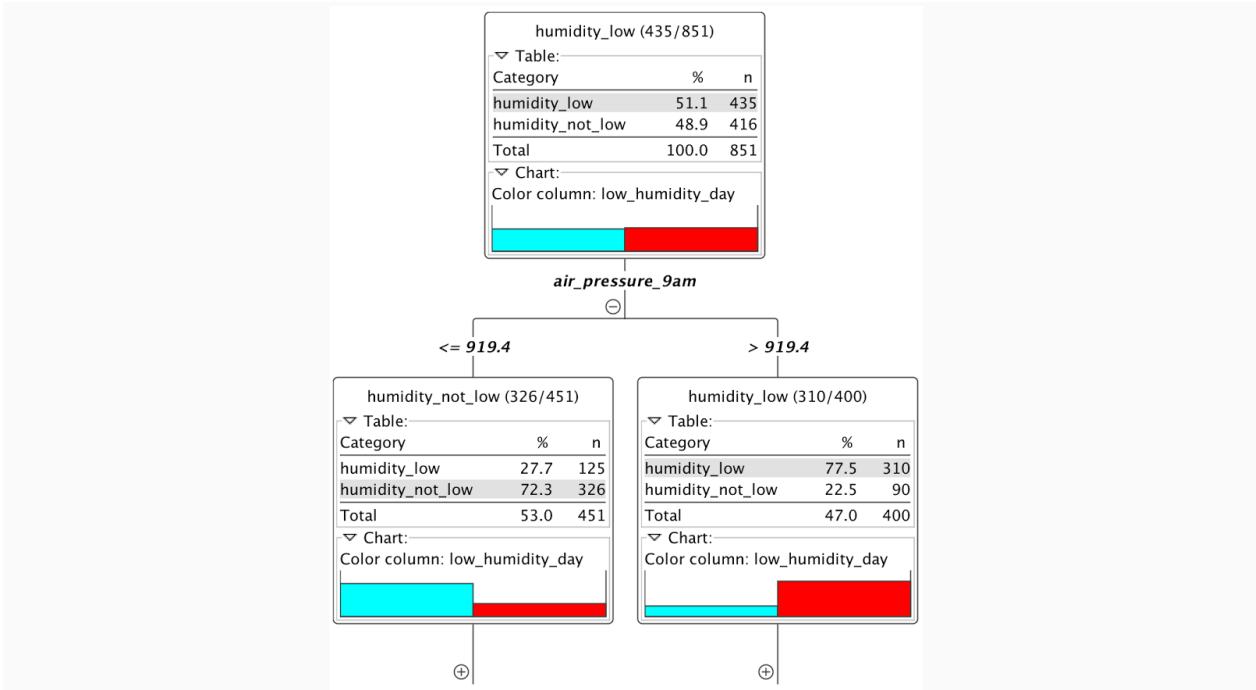
## Classification Task

Recall that the task is to classify low-humidity days vs. days with normal or high relative humidity. The class label is based on the categorical variable **low\_humidity\_day**. This variable was created from the numeric variable `relative_humidity_3pm`. The class target `low_humidity_day` was created with the following categories:

- **humidity\_low**: if `relative_humidity_3pm < 25`
- **humidity\_not\_low**: if `relative_humidity_3pm >= 25`

## Decision Tree Model

First, let's take a look at just the first two levels of the tree. You can see the following image by right-clicking on the **Decision Tree Learner** node in the workflow and selecting “View: Decision Tree View”:



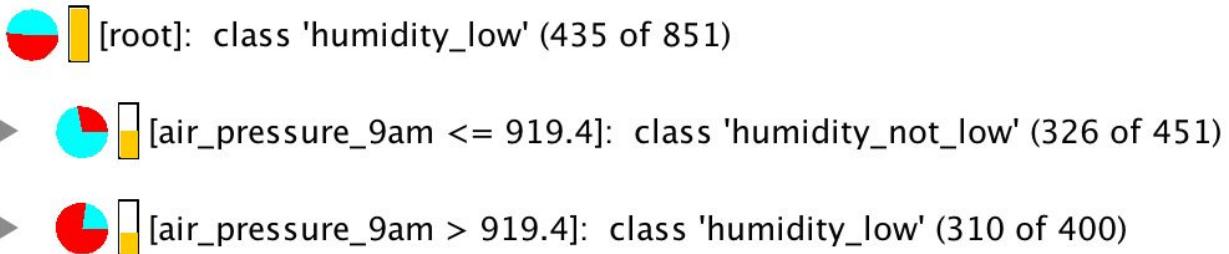
Looking at the root node (the top node), we see that there 851 samples in total. Of these, 435 or 51.1% of the samples are labeled as humidity\_low; that is, the true label of these samples is humidity\_low. Of the total number of samples, 416 or 48.9% are labeled as humidity\_not\_low. So at the root node, approximately half of the samples are humidity\_low and half are humidity\_not\_low. This is indicated by the color bars at the bottom of the root node: blue is for humidity\_not\_low, and red is for humidity\_low, and the height of each bar specifies the percentage of samples labeled with the respective category.

## Split #1 on air\_pressure\_9am

The first split is on the variable **air\_pressure\_9am**. Samples with  $\text{air\_pressure\_9am} \leq 919.4$  are placed in the left child node where most of the samples are labeled as humidity\_not\_low. Samples with  $\text{air\_pressure} > 919.4$  are placed in the right child node where most of the samples are labeled as humidity\_low. Note that the color red is associated with the humidity\_low class. What this first split specifies is that high values of air\_pressure are associated with humidity\_low. This makes sense since high air pressure usually corresponds to sunny days, which have normal or high relative humidity.

To look at more levels in the decision tree, we need a more compact view. So we will now switch to the 'simple' view. The following image shows the same tree structure as the image of the decision

tree above, and is generated by clicking on the Decision Tree Learner node and selecting “View: Decision Tree View (simple)”:

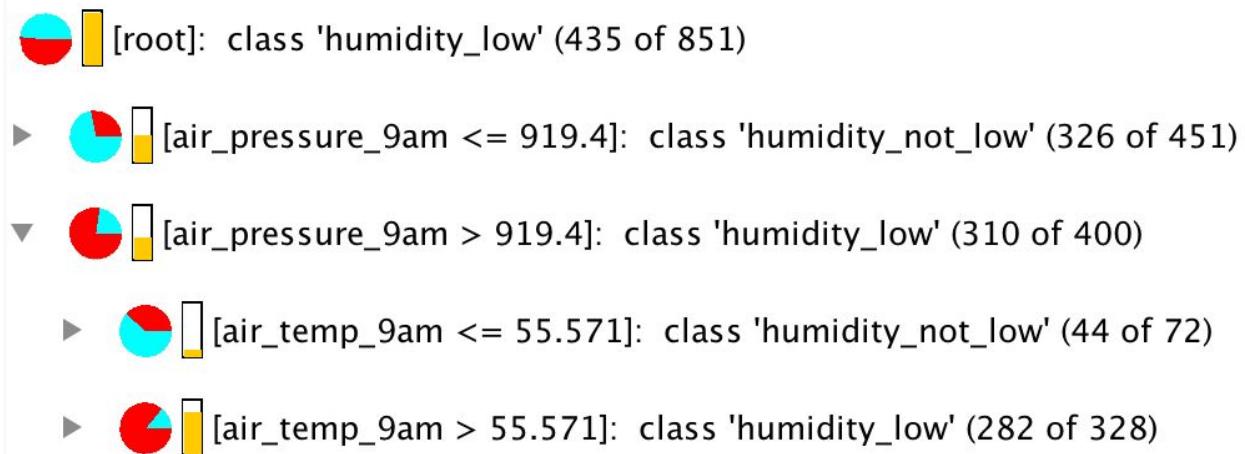


The root node is shown as the top line, followed by the children nodes resulting from the split on `air_pressure_9am`. Again, samples with  $\text{air\_pressure\_9am} \leq 919.4$  are placed in the left child node (shown right under the root node) where most of the samples are labeled as `humidity_not_low`. In other words, the true label for most samples in the left child node is `humidity_not_low`, which is indicated by the pie chart symbol being mostly blue, and the numbers in parentheses specifying that 326 out of 451 samples in that node are labeled `humidity_not_low`. Samples with  $\text{air\_pressure\_9am} > 919.4$  are placed in the right child node where most of the samples are labeled as `humidity_low`.

Note that no prediction has been made yet since classification decisions are not made until a leaf node is reached.

## Split #2 on `air_temp_9am`

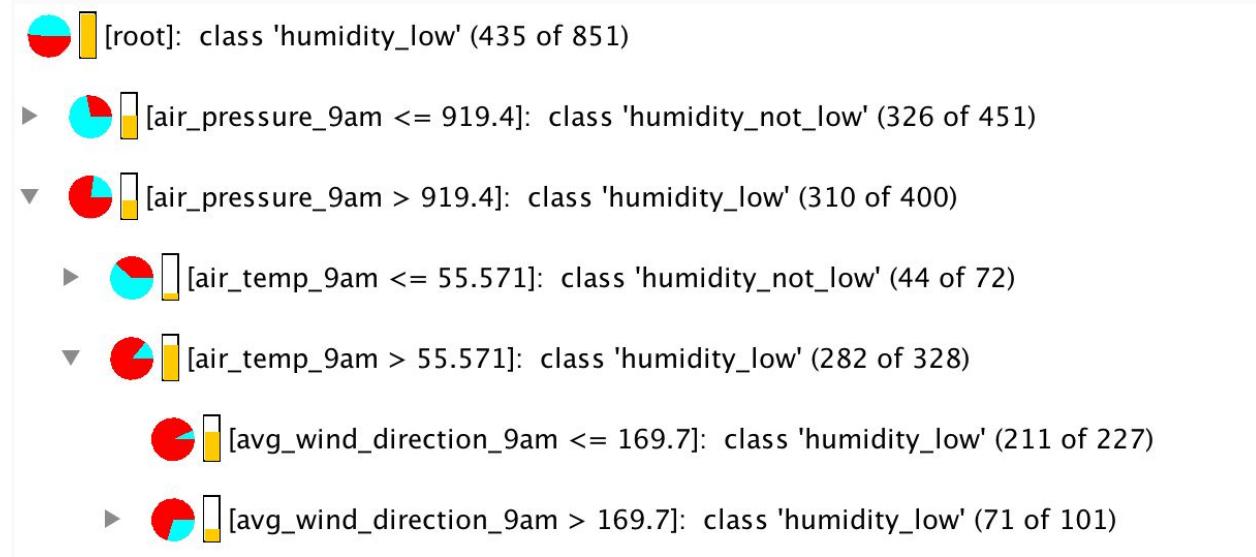
Let's continue down the branch of the right child, where most of the samples have true label as `humidity_low`. If we expand that node, we get the following tree:



We see that this split is based on the variable **air\_temp\_9am**. If a sample has a value for  $\text{air\_temp\_9am} \leq 55.571$ , then it placed in the left child node, where most of the samples are labeled as **humidity\_not\_low**. And if a sample has a value for  $\text{air\_temp\_9am} > 55.571$ , then it is placed in the right child node, where most of the samples are labeled as **humidity\_low**. What this means is that low-humidity days are associated with warmer days. This makes sense since days with high humidity tend to be rainy days with cooler temperatures, while days with low humidity are sunny days with warmer temperatures.

## Split #3 on avg\_wind\_direction\_9am

Continuing with the **humidity\_low** branch, we expand the right child node to get the following:



The third split is based on the variable **avg\_wind\_direction\_9am**. Samples with  $\text{avg\_wind\_direction\_9am} \leq 169.7$  are placed in the left child node where most of the samples are labeled as **humidity\_low**. Samples with  $\text{avg\_wind\_diection\_9am} > 169.7$  are placed in the right child node. Notice that most of the samples in the right child node are also labeled **humidity\_low**, but there is still additional processing needed with those samples since the right child node is not a leaf node.

## Classification Rules

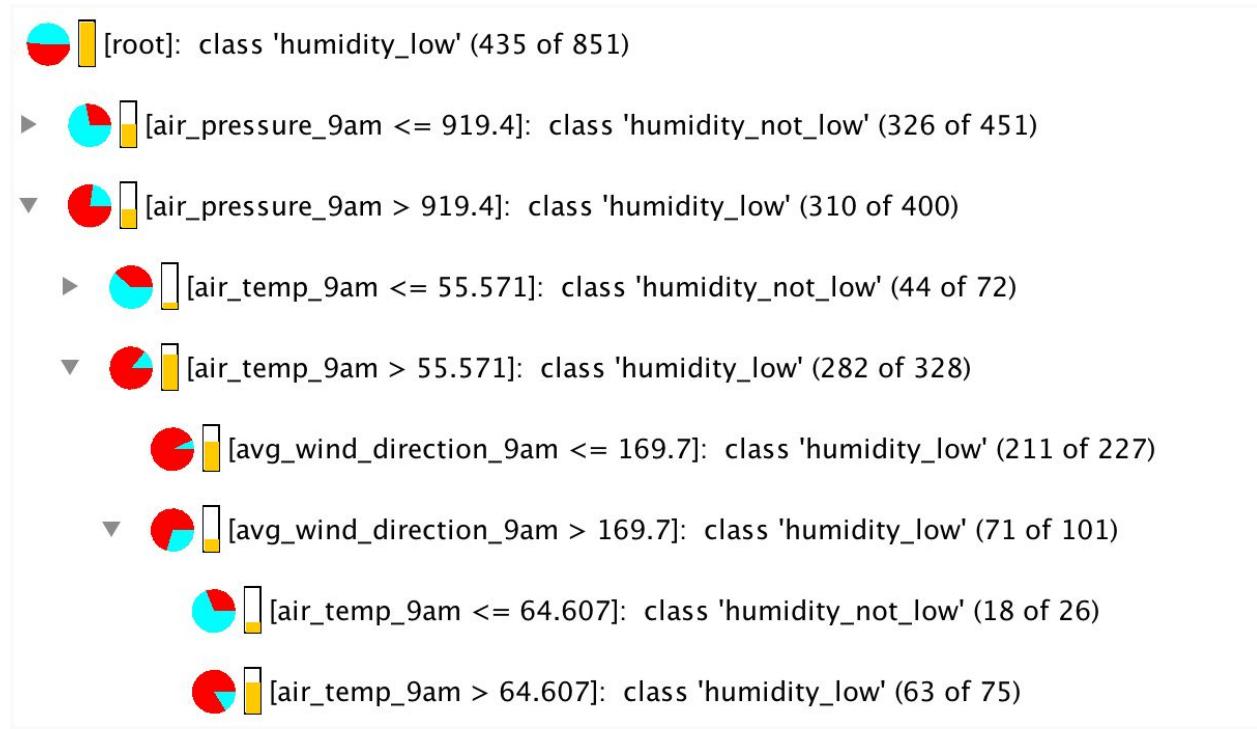
With the left child node, we have now reached a leaf node! Traversing from the root node to this leaf node, we can now see how a sample is classified as **humidity\_low**:

1. If air\_pressure\_9am > 919.4 and
2. If air\_temp\_9am > 55.571 and
3. If avg\_wind\_direction\_9am <= 169.7
4. Then sample is classified as humidity\_low

This translates to days with high air pressure and warmer temperatures, with wind direction from the east are likely to be days with low relative humidity.

We have discussed above that low humidity is more likely to occur on sunny days with high air pressure and warmer temperatures. Now let's consider wind direction. Values for wind direction start at 0 degree for due North, and increases clockwise. So wind direction  $\leq 169.7$  means that the wind is from an eastern direction. For San Diego, this means warmer, drier air from the inland areas as opposed to cooler air with more moisture from the ocean. So this relationship between winds from the east and days with low humidity makes sense.

Expanding the right child node with  $\text{avg\_wind\_direction\_9am} > 169.7$ , we get:



For the left leaf node, we see the following rules:

1. If air\_pressure\_9am > 919.4 and
2. If air\_temp\_9am > 55.571 and
3. If avg\_wind\_direction\_9am > 169.7 and

4. If air\_temp\_9am <= 64.607
5. Then sample is classified as humidity\_not\_low

This translates to the following: Days with high air pressure, winds from the west, and temperatures between 56 and 65 degrees Fahrenheit are likely to be days with normal or high relative humidity.

For the right leaf node, we get:

1. If air\_pressure\_9am > 919.4 and
2. If air\_temp\_9am > 55.571 and
3. If avg\_wind\_direction\_9am > 169.7 and
4. If air\_temp\_9am > 64.607
5. Then sample is classified as humidity\_low

This translates to: Days with high air pressure, winds from the west, and temperatures greater than 65 degrees Fahrenheit are likely to be days with low humidity.

This branch is now complete. There are three leaf nodes, so there are three ways to assign a prediction of either humidity\_low or humidity\_not\_low to each sample that is sent down this branch of the tree.

Other branches of the tree can be interpreted in a similar way. As with any other real dataset, some cases may require complex rules to form a classification decision.

## Feature Importance

Aside from interpretability, another advantage of decision tree is that the resulting model tells you which features are important in the classification task. If you expand the tree to show all leaf nodes, you will see all the variables that the tree uses to perform the classification task.

For our daily weather dataset, note that out of the seven original input variables, only four variables (air\_pressure\_9am, air\_temp\_9am, avg\_wind\_direction\_9am, max\_wind\_direction\_9am) are used in the construction of the tree. These four variables are deemed important variables for this classification task, while the other variables do not contribute to the classification decisions made by the model.

# Generalization & Overfitting

# After this video you will be able to..

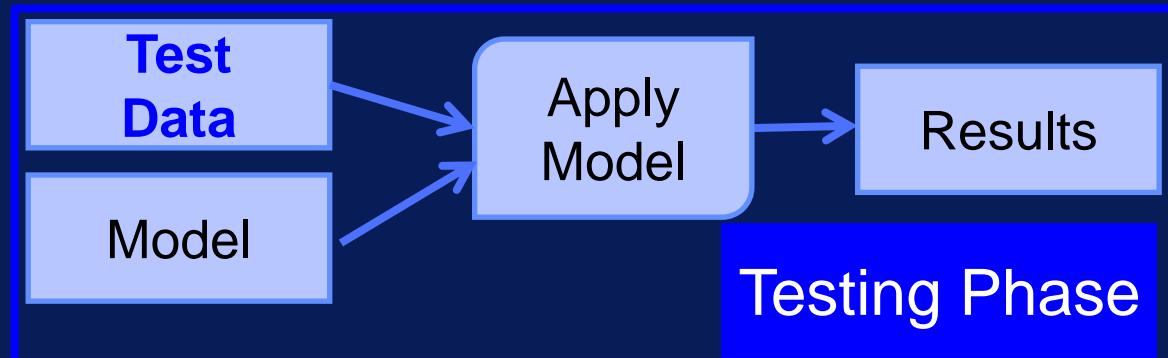
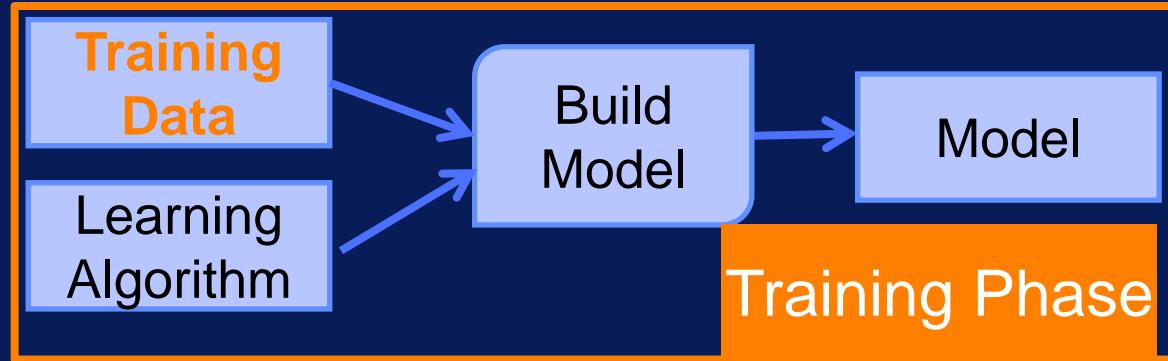
- Define what generalization is
- Describe how overfitting is related to generalization
- Explain why overfitting should be avoided

# Errors in Classification

- Success:  $\text{Output} = \text{Target}$  ← true label
- Error:  $\text{Output} \neq \text{Target}$
- Error rate = Error = Misclassification Error
  - $\# \text{ errors} / \# \text{ samples} = \% \text{ error}$



# Training vs. Testing Phases



# Errors in Classification

Error on  
Training  
Data

Error on  
Test  
Data

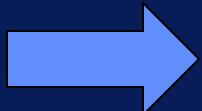
Training Error

Test Error

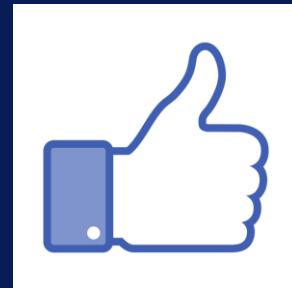
Test error indicates how well  
model will perform on new data!

# Generalization

*Performs well  
on new data*



*Good  
Generalization*



Test Error = Generalization Error

# *Overfitting*

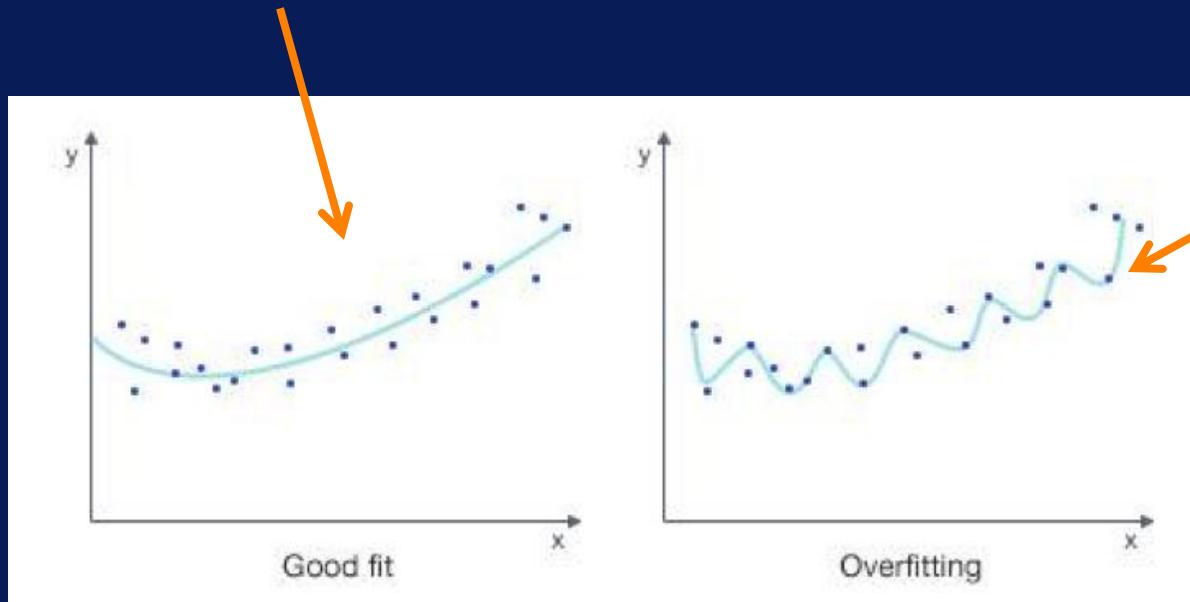
Test Error



Training Error

# Overfitting

Model is fitting to  
structure of data



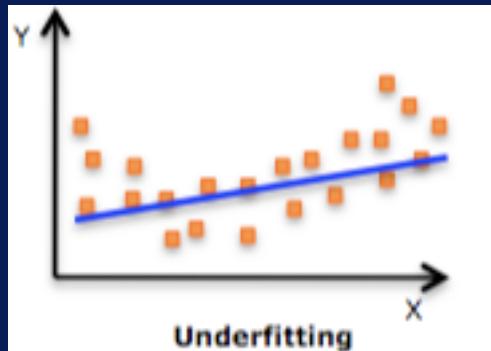
Model is fitting  
to noise in data

Source: <http://blog.fliptop.com/blog/2015/03/02/bias-variance-and-overfitting-machine-learning-overview/>

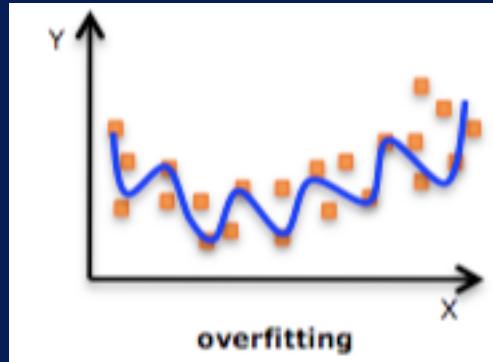
# Overfitting & Generalization

Overfitting → Poor  
Generalization

# Overfitting & Underfitting



Underfitting

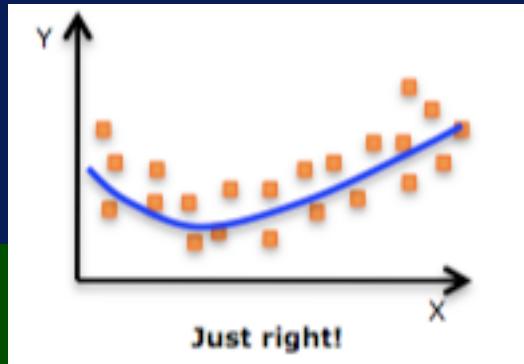


overfitting

Underfitting

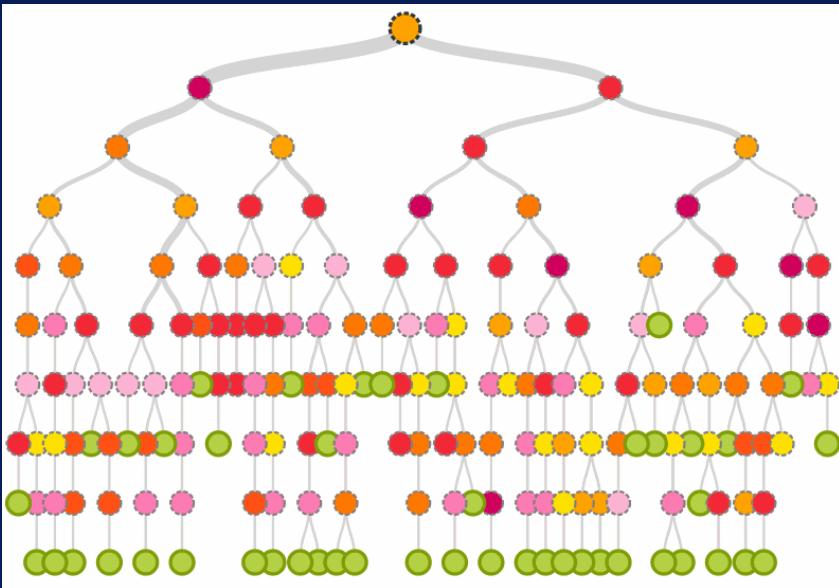
Overfitting

Just Right



Just right!

# What Causes Overfitting?



Overfitting

Overly complex model

# Generalization & Overfitting

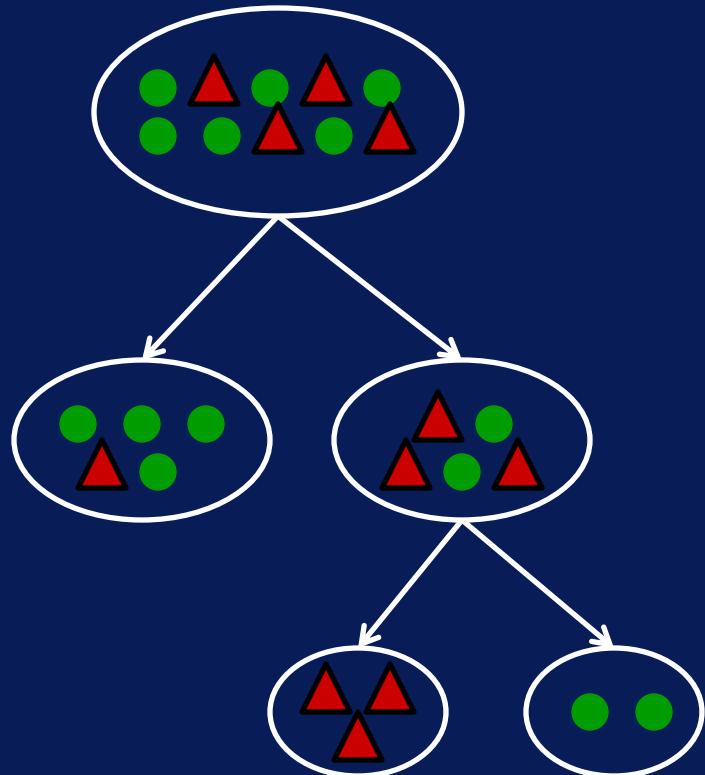
Over~~f~~itting → Good  
Generalization

# Overfitting in Decision Trees

# After this video you will be able to..

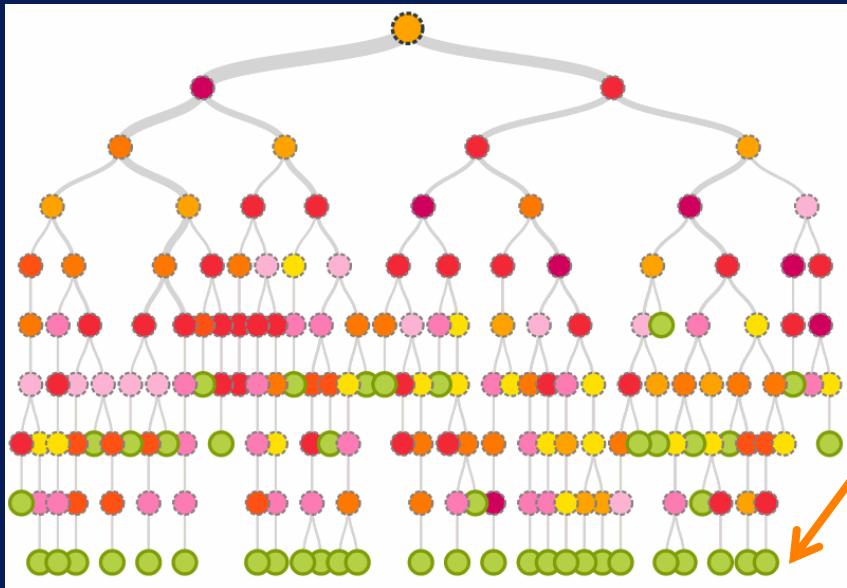
- Discuss overfitting in the context of decision tree models
- Explain how overfitting is addressed in decision tree induction
- Define pre-pruning and post-pruning

# Decision Tree Induction



# Overfitting in Decision Tree

If nodes are fitting to noise  
in training data, model will  
not generalize well



Source: <http://piepdx.org/blog/2013/12/10/which-one-is-is>

# Avoiding Overfitting in Decision Tree

## Pre-Pruning

Stop growing tree before fully grown

## Post-Pruning

Grow tree to max size, then prune

Control number of nodes to limit complexity of tree

# Pre-Pruning

Pre-Pruning

- Restrictive stopping conditions for growing tree:
  - Stop if number of records < some threshold
  - Stop if improvement in impurity measure < some threshold

Stop growing tree before fully grown

# Post-Pruning

- Pruning
  - Remove nodes from bottom up
  - Replace subtree with leaf node if generalization error improves or does not change

Post-Pruning

Grow tree to  
max size,  
then prune

# Overfitting in Decision Tree

## Pre-Pruning

Stop growing tree before fully grown

## Post-Pruning

Grow tree to max size, then prune

- Post-pruning used more often
- But is more computational expensive

# Tree Pruning to Avoid Overfitting

## Pre-Pruning

Stop growing tree before fully grown

## Post-Pruning

Grow tree to max size, then prune

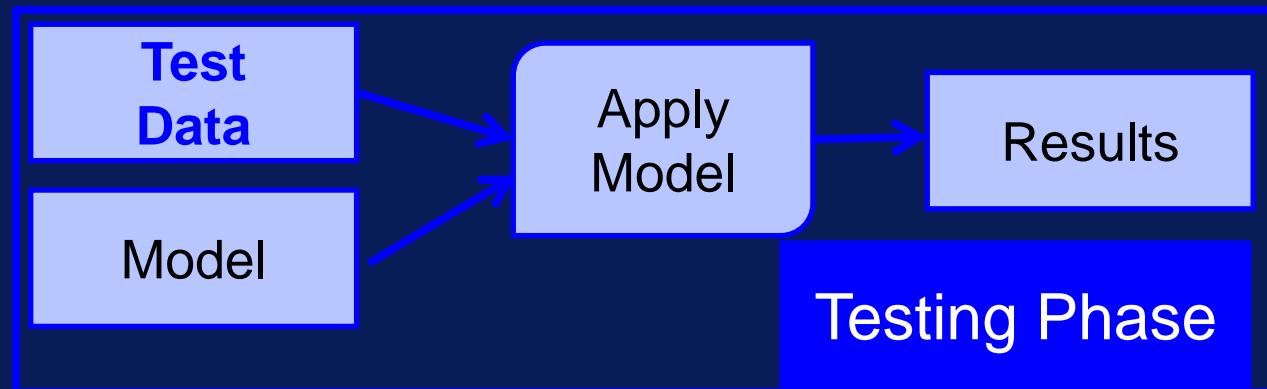
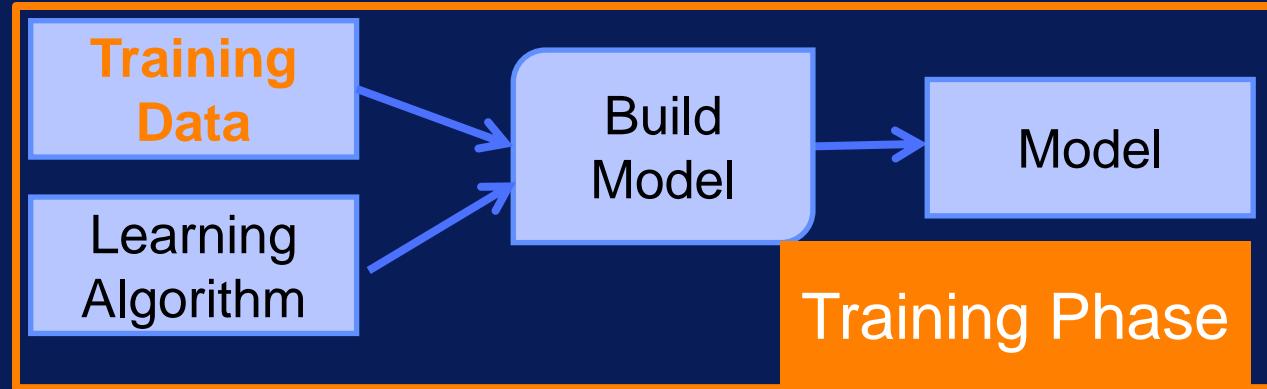
Control number of nodes to limit complexity of tree

# Using a Validation Set

# After this video you will be able to..

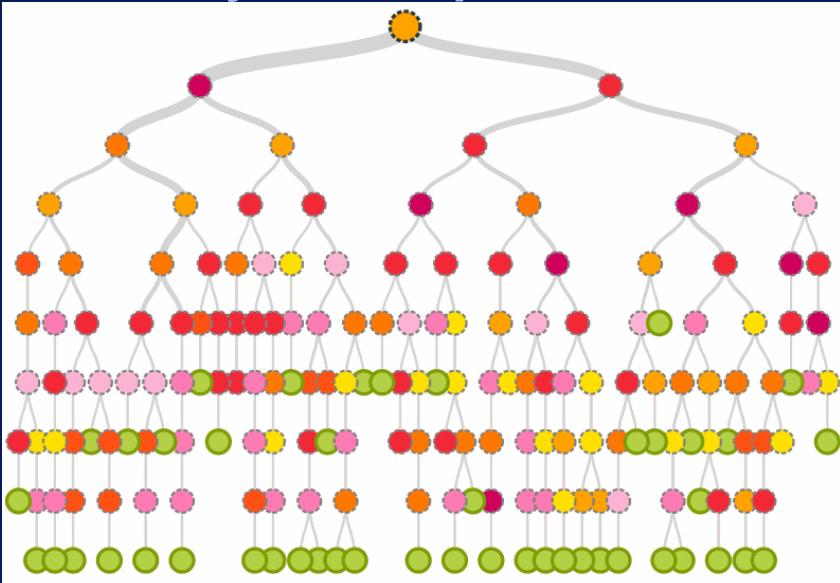
- Describe how a validation set can be used to avoid overfitting
- Articulate how training, validation, and test sets are used
- List three ways that validation can be performed

# Training vs. Testing Phases



# Avoiding Overfitting

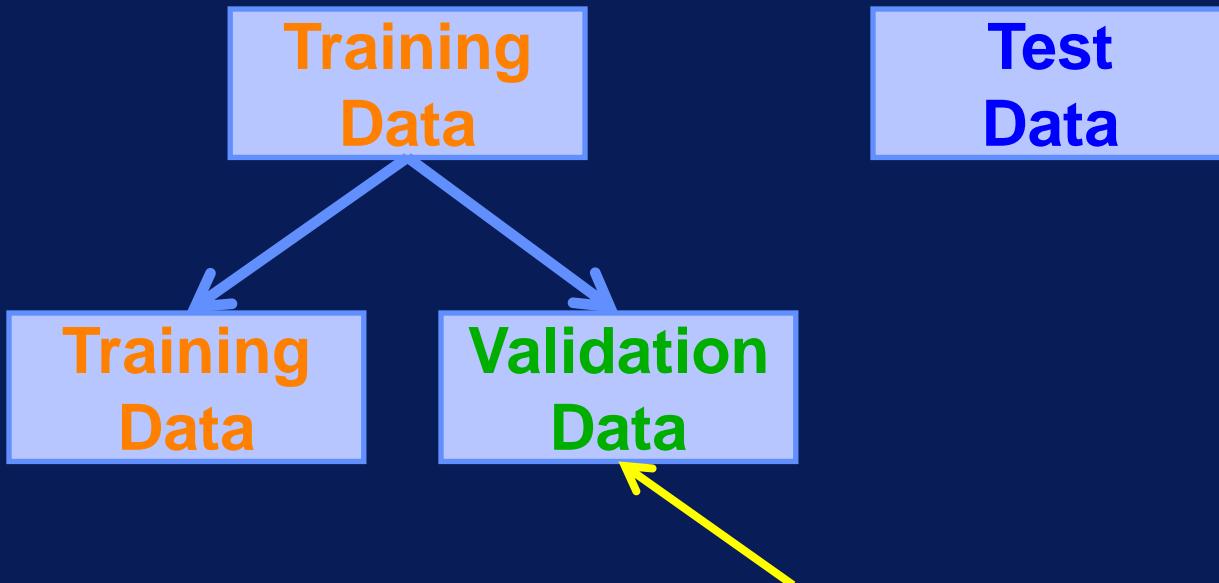
Overly complex model



Overfitting

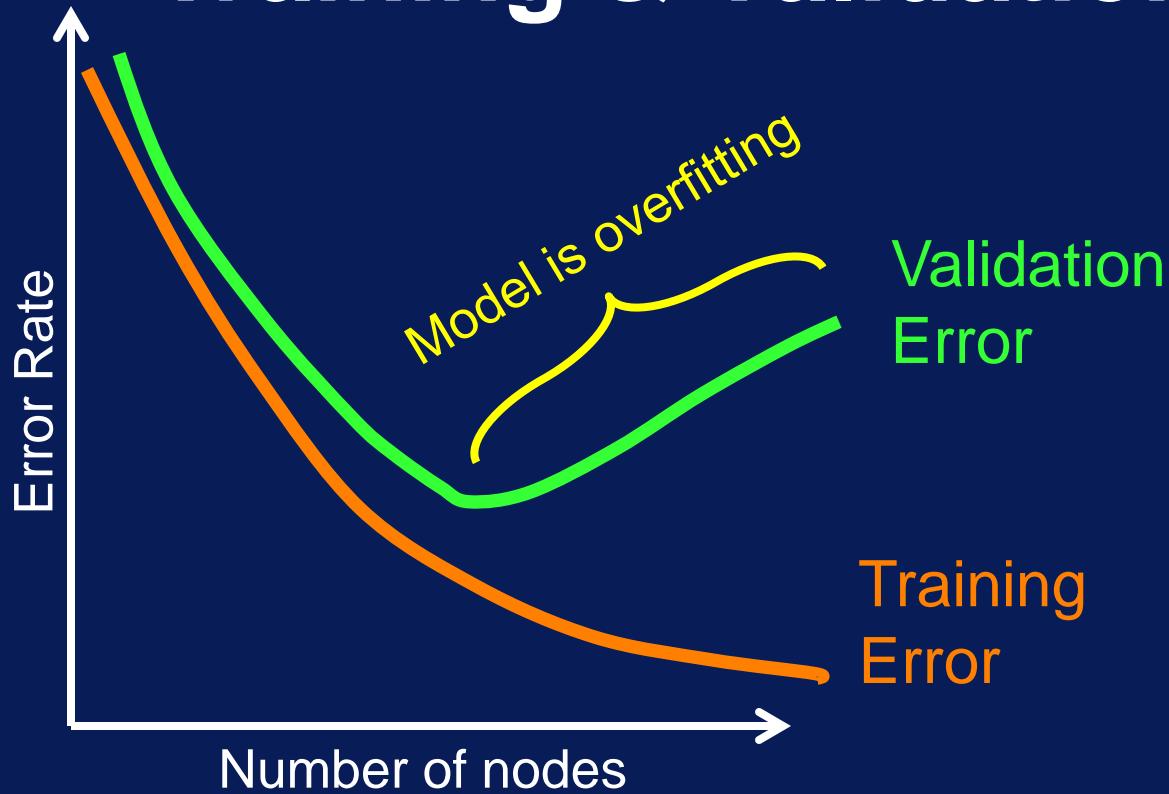
When to stop  
training before  
model gets  
too complex?

# Validation Set

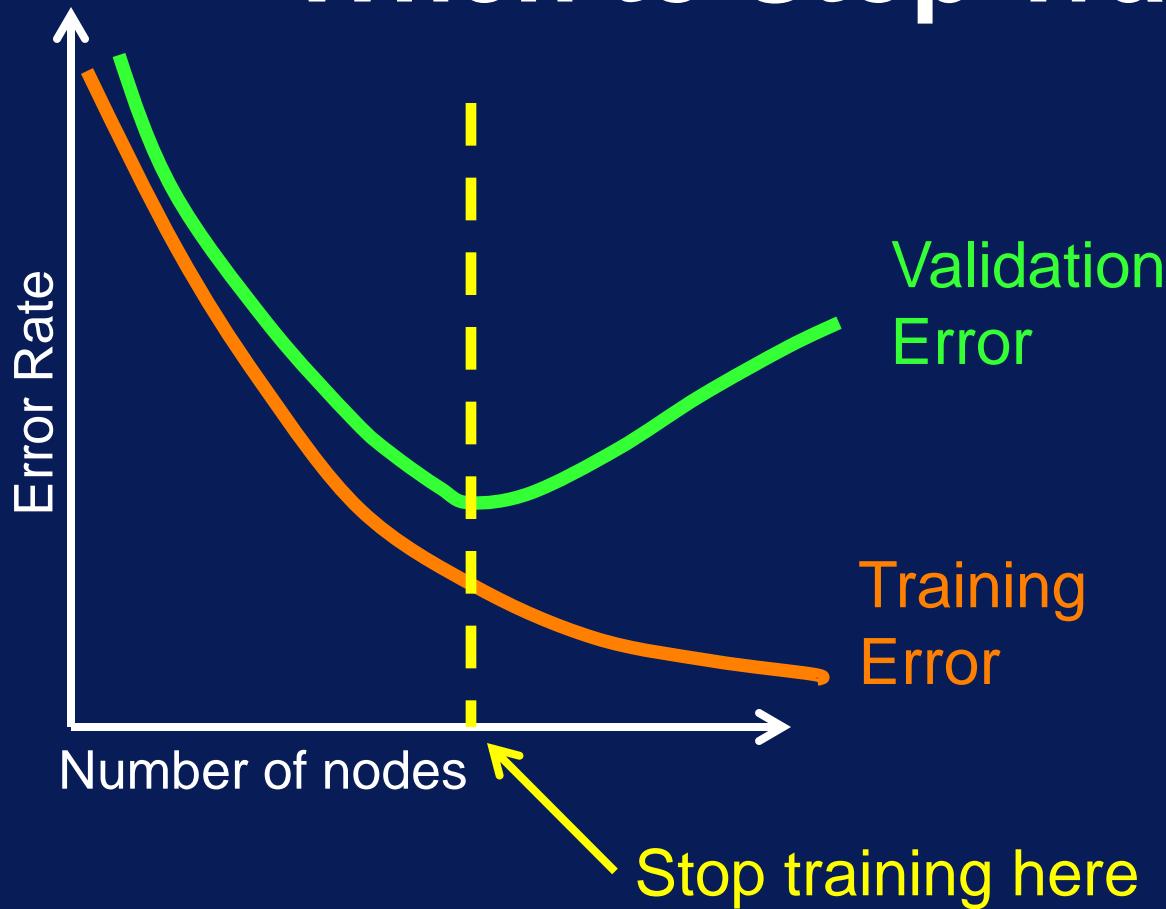


Used to determine when to stop  
training to avoid overfitting

# Training & Validation Errors



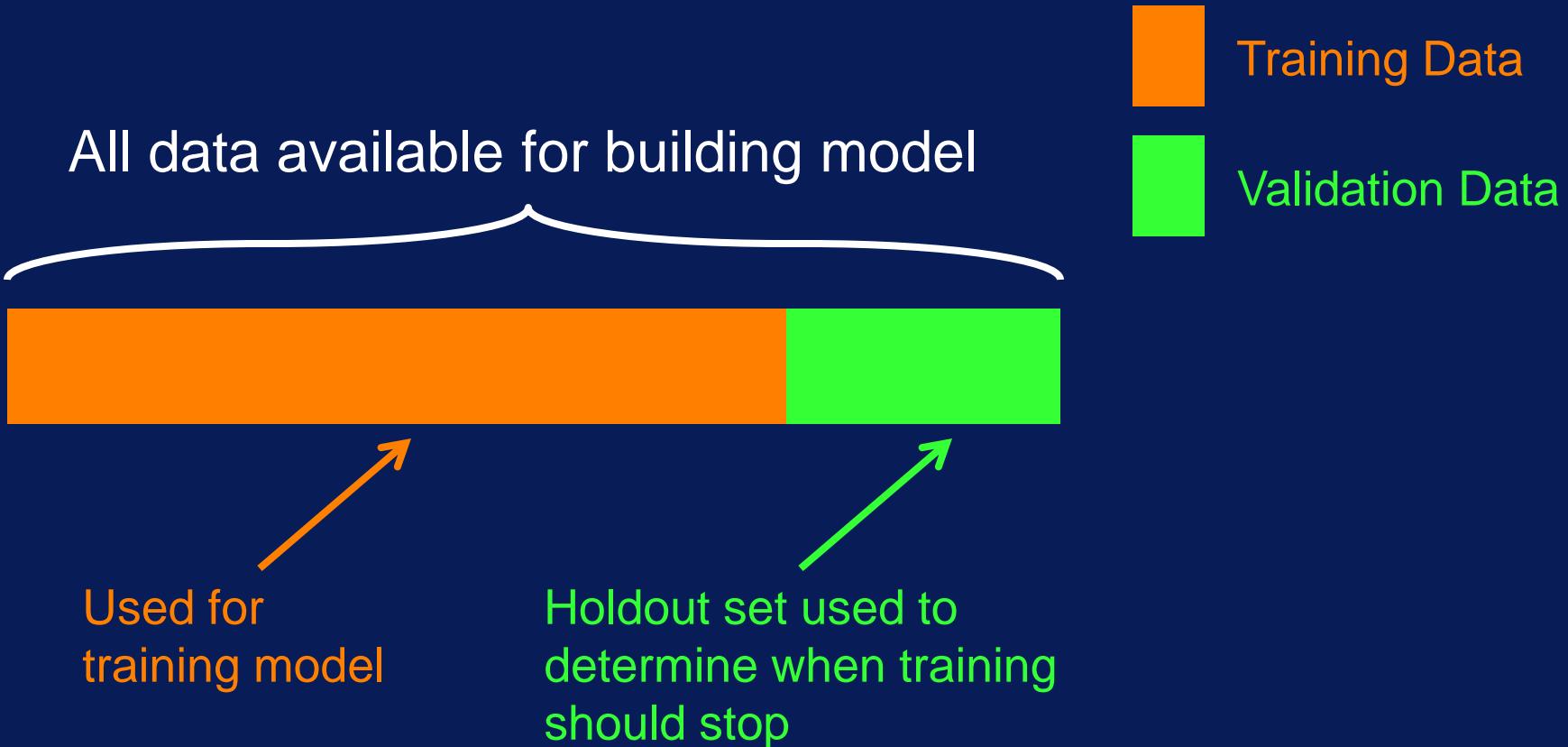
# When to Stop Training



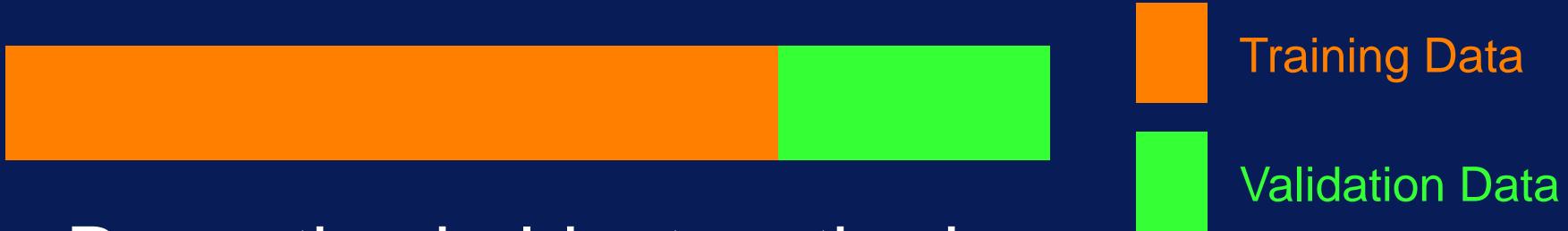
# Ways to Create & Use Validation Set

- Holdout method
- Random subsampling
- K-fold cross-validation
- Leave-one-out cross-validation

# Holdout Method



# Repeated Holdout



- Repeating holdout method several times
- Randomly select different hold out set each iteration
- Average validation errors over all repetitions

# K-Fold Cross-Validation



# Leave-One-Out Cross-Validation



# Uses of Validation Set

Validation  
Data

- Uses:
  - Address overfitting
  - Estimate generalization performance

# Datasets

## Training Data

Adjust model parameters

## Validation Data

Determine when to stop training (avoid overfitting)

Estimate generalization performance

## Test Data

Evaluate performance on new data

Cannot be used in any way in model creation!



# Validation Set Summary

Training  
Data

Validation  
Data

Test  
Data

- Datasets: training, validation, test
- Validation set: avoid overfitting, estimate generalization
- Using validation: holdout, repeated holdout, cross-validation (k-fold, leave-one-out)

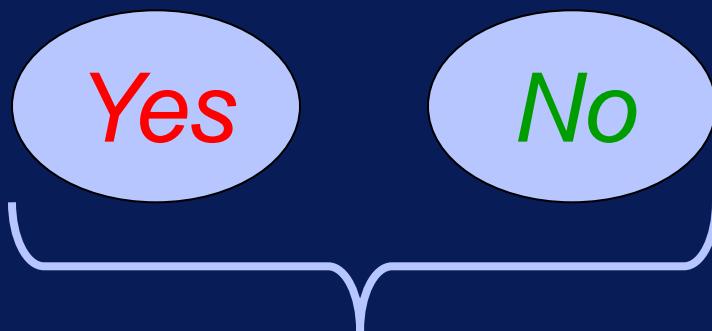
# Metrics to Evaluate Model Performance

# After this video you will be able to..

- Discuss how performance metrics can be used to evaluate models
- Name three model evaluation metrics
- Explain why accuracy may be misleading

# Classification

Is this animal a mammal?

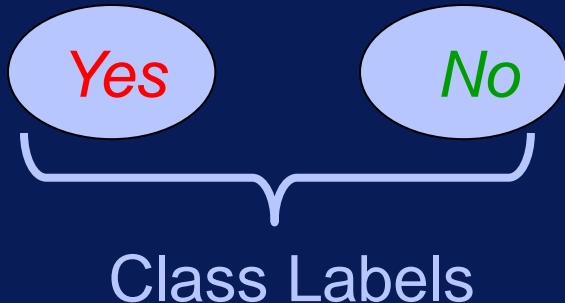


Class Labels

# Types of Classification Errors

Is this animal a mammal?

True Label	Predicted Label	Error Type
Yes	Yes	True Positive (TP)
No	No	True Negative (TN)
No	Yes	False Positive (FP)
Yes	No	False Negative (FN)



# Accuracy Rate

True	Predicted	Error
Yes	Yes	True Positive (TP)
No	No	True Negative (TN)
No	Yes	False Positive (FP)
Yes	No	False Negative (FN)

$$\begin{aligned} \text{Accuracy Rate} &= \frac{\# \text{ correct predictions}}{\# \text{ total predictions}} \\ &= \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \end{aligned}$$

# Error Rate

True	Predicted	Error
Yes	Yes	True Positive (TP)
No	No	True Negative (TN)
No	Yes	False Positive (FP)
Yes	No	False Negative (FN)

$$\begin{aligned}\text{Error Rate} &= \frac{\# \text{ incorrect predictions}}{\# \text{ total predictions}} \\ &= \frac{FN + TP}{TP + TN + FP + FN} \\ &= 1 - \text{Accurate Rate}\end{aligned}$$

True Label	Predicted Label
Yes	No
No	No
No	No
Yes	Yes
Yes	Yes
No	No
Yes	No
Yes	Yes
No	No
No	Yes

True	Predicted	Error
Yes	Yes	True Positive (TP)
No	No	True Negative (TN)
No	Yes	False Positive (FP)
Yes	No	False Negative (FN)

# Classification Example

True Label	Predicted Label	True	Predicted	Error
Yes	No	Yes	Yes	True Positive (TP)
No	No	No	No	True Negative (TN)
No	No	No	Yes	False Positive (FP)
Yes	Yes	Yes	No	False Negative (FN)
Yes	Yes	Yes	Yes	TP = 3
No	No	No	No	
Yes	No	No	No	
Yes	Yes	Yes	Yes	
No	No	No	No	
No	Yes	No	Yes	

Classification  
Example

True Label	Predicted Label	True	Predicted	Error
Yes	No	Yes	Yes	True Positive (TP)
No	No	No	No	True Negative (TN)
No	No	No	Yes	False Positive (FP)
Yes	Yes	Yes	No	False Negative (FN)
Yes	Yes			
No	No			
Yes	No			
Yes	Yes			
No	No			
No	Yes			

TN = 4

## Classification Example

# Accuracy Rate

True	Predicted	Error
Yes	Yes	True Positive (TP)
No	No	True Negative (TN)
No	Yes	False Positive (FP)
Yes	No	False Negative (FN)

$$\text{Accuracy Rate} = \frac{\# \text{ correct predictions}}{\# \text{ total predictions}}$$

$$= \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

$$= (3 + 4) / 10 = 7 / 10 = 0.7$$

# Error Rate

True	Predicted	Error
Yes	Yes	True Positive (TP)
No	No	True Negative (TN)
No	Yes	False Positive (FP)
Yes	No	False Negative (FN)

$$\text{Error Rate} = \frac{\# \text{ incorrect predictions}}{\# \text{ total predictions}}$$

$$= 1 - \text{Accuracy Rate}$$

$$= 1 - 0.7 = 0.3$$

# Limitation with Accuracy

Is this tumor cancerous?



most are  
negative  
examples

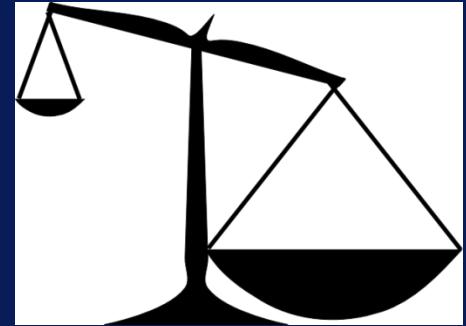
very few  
positive  
examples

Class Imbalance  
Problem

# Limitation with Accuracy

Is this tumor cancerous?

- Say 3% of samples are cancer
- If model always predicts non-cancer
  - Accuracy = 97%
  - But no cancer cases detected!

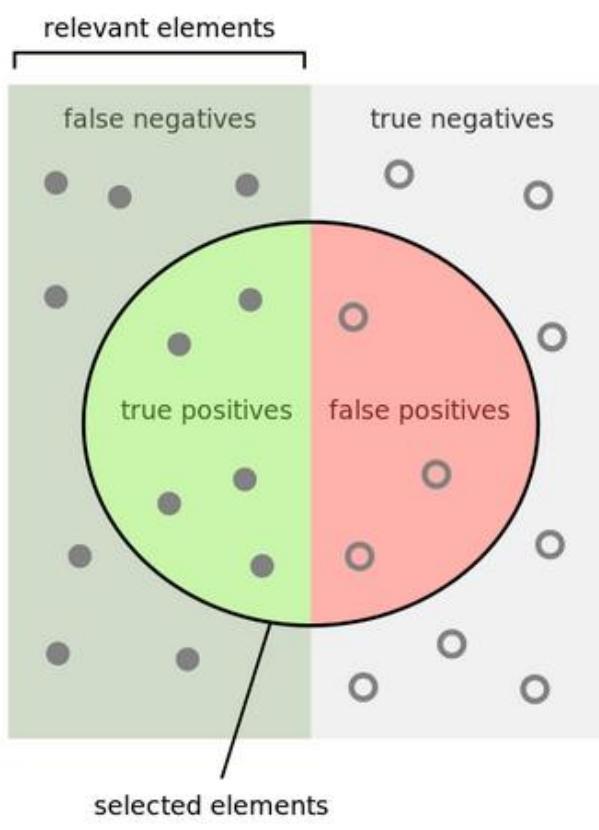


# Precision & Recall

True	Predicted	Error
Yes	Yes	True Positive (TP)
No	No	True Negative (TN)
No	Yes	False Positive (FP)
Yes	No	False Negative (FN)

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \leftarrow \begin{array}{l} \text{All samples with} \\ \text{Predicted} = \text{Yes} \end{array}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \leftarrow \begin{array}{l} \text{All samples with} \\ \text{True} = \text{Yes} \end{array}$$



How many selected items are relevant?

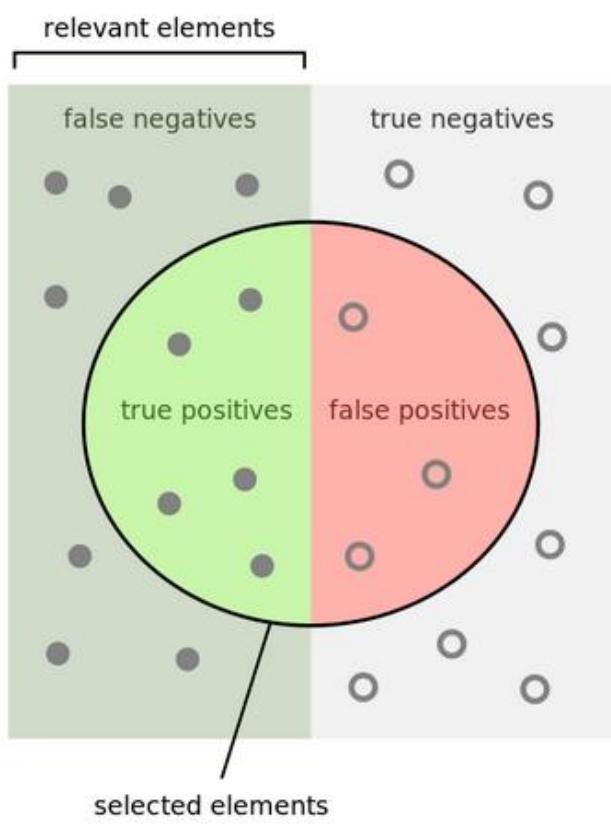
$$\text{Precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

How many relevant items are selected?

$$\text{Recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

Samples correctly predicted as Positive

# Recall



How many selected items are relevant?

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$



How many relevant items are selected?

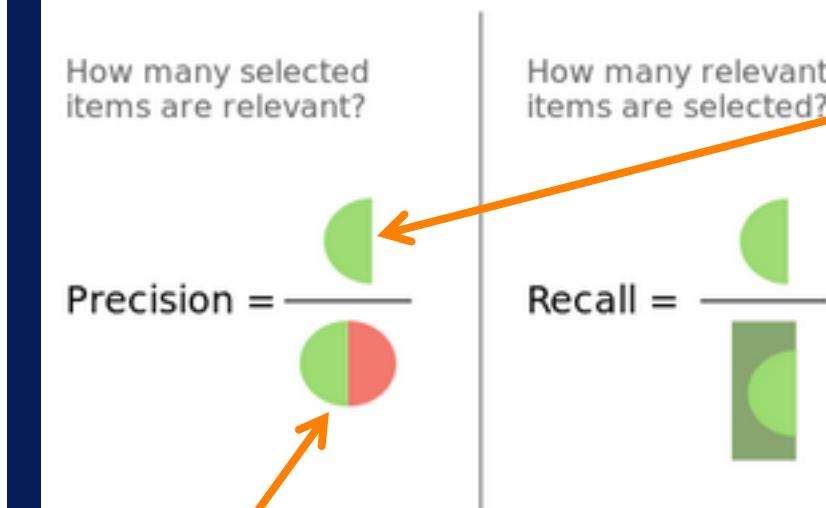
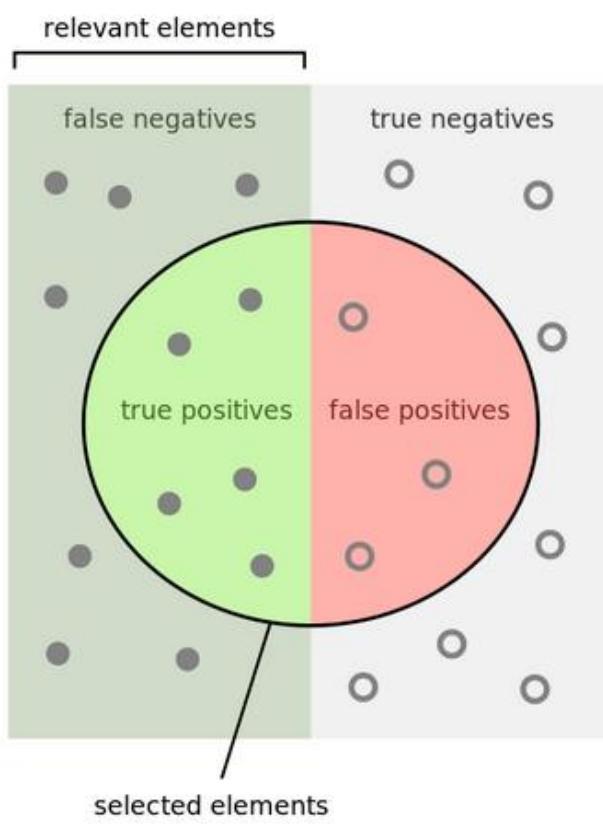
$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$



Samples correctly predicted as Positive

Samples actually Positive

# Recall



Samples correctly predicted as Positive

# Recall

# Precision & Recall

True	Predicted	Error
Yes	Yes	True Positive (TP)
No	No	True Negative (TN)
No	Yes	False Positive (FP)
Yes	No	False Negative (FN)

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} = \frac{\text{Positive samples correctly predicted}}{\text{All samples predicted as Positive}}$$

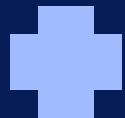
Measure of exactness

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} = \frac{\text{Positive samples correctly predicted}}{\text{All samples with true label Positive}}$$

Measure of completeness

# Precision & Recall

Precision

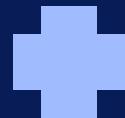


Recall

- Use together
- Goal: Maximize both

# F-Measure

Precision



Recall

$$F_1 = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

- $F_1$  : evenly weighted
- $F_2$  : weights Recall more
- $F_{0.5}$ : weights Precision more

# Evaluation Metrics

*Accuracy  
Rate*

*Precision  
& Recall*

True	Predicted	Error
Yes	Yes	True Positive (TP)
No	No	True Negative (TN)
No	Yes	False Positive (FP)
Yes	No	False Negative (FN)

*Error  
Rate*

*F<sub>1</sub>-Measure*

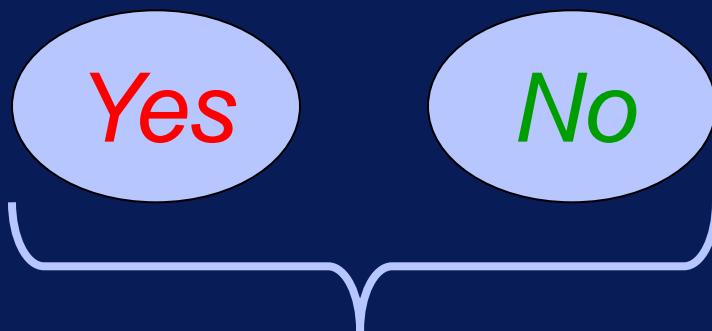
# Confusion Matrix

# After this video you will be able to..

- Describe how a confusion matrix can be used to evaluate a classifier
- Interpret the confusion matrix of a model
- Relate accuracy to values in a confusion matrix

# Classification

Is this animal a mammal?

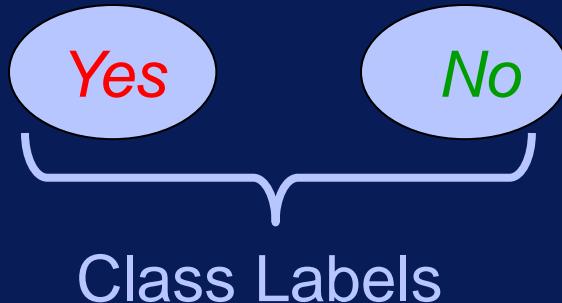


Class Labels

# Types of Classification Errors

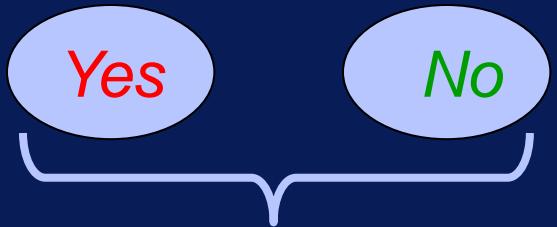
Is this animal a mammal?

True Label	Predicted Label	Error Type
Yes	Yes	True Positive (TP)
No	No	True Negative (TN)
No	Yes	False Positive (FP)
Yes	No	False Negative (FN)



# Confusion Matrix

Is this animal a mammal?



	Predicted Class Label		Class Labels
True Class Label			Yes      No
	Yes	True Positive (TP)	False Negative (FN)
	No	False Positive (FP)	True Negative (TN)

True Label	Predicted Label
Yes	No
No	No
No	No
Yes	Yes
Yes	Yes
No	No
Yes	No
Yes	Yes
No	No
No	Yes

	Predicted Class Label		
True Class Label	Yes	No	
	Yes	TP	FN
	No	FP	TN

# Confusion Matrix

True Label	Predicted Label
Yes	No
No	No
No	No
Yes	Yes
Yes	Yes
No	No
Yes	No
Yes	Yes
No	No
No	Yes

	Predicted Class Label	
True Class Label	Yes	No
Yes	TP = 3	
No		

# Confusion Matrix

TP



True Label	Predicted Label
Yes	No
No	No
No	No
Yes	Yes
Yes	Yes
No	No
Yes	No
Yes	Yes
No	No
No	Yes

	Predicted Class Label	
True Class Label	Yes	No
Yes	TP = 3	
No		TN = 4

# Confusion Matrix

True Label	Predicted Label
Yes	No
No	No
No	No
Yes	Yes
Yes	Yes
No	No
Yes	No
Yes	Yes
No	No
No	Yes

	Predicted Class Label	
True Class Label	Yes	No
Yes	TP = 3	FN = 2
No		TN = 4

# Confusion Matrix

FN

True Label	Predicted Label
Yes	No
No	No
No	No
Yes	Yes
Yes	Yes
No	No
Yes	No
Yes	Yes
No	No
No	Yes

	Predicted Class Label	
True Class Label	Yes	No
Yes	TP = 3	FN = 2
No	FP = 1	TN = 4

FP



# Confusion Matrix

True Label	Predicted Label
Yes	No
No	No
No	No
Yes	Yes
Yes	Yes
No	No
Yes	No
Yes	Yes
No	No
No	Yes

	Predicted Class Label	
True Class Label	Yes	No
Yes	TP = 3	FN = 2
No	FP = 1	TN = 4

# Confusion Matrix

True Label	Predicted Label
Yes	No
No	No
No	No
Yes	Yes
Yes	Yes
No	No
Yes	No
Yes	Yes
No	No
No	Yes

	Predicted Class Label	
True Class Label	Yes	No
Yes	TP = 3	FN = 2
No	FP = 1	TN = 4

Correct Predictions :  
7 out of 10 = 0.7

True Label	Predicted Label
Yes	No
No	No
No	No
Yes	Yes
Yes	Yes
No	No
Yes	No
Yes	Yes
No	No
No	Yes

	Predicted Class Label	
True Class Label	Yes	No
Yes	TP = 3	FN = 2
No	FP = 1	TN = 4

Incorrect Predictions :  
 3 out of 10 = 0.3

# Confusion Matrix & Accuracy Rate

	Predicted Class Label		
True Class Label	Yes	No	
	Yes	TP = 3	FN = 2
	No	FP = 1	TN = 4

$$\text{Accuracy Rate} = \frac{\# \text{ correct predictions}}{\# \text{ total predictions}}$$

$$= \frac{TP + TN}{TP + TN + FP + FN}$$

$$= (3 + 4) / 10 = 7 / 10 = 0.7$$

# Confusion Matrix & Error Rate

	Predicted Class Label		
True Class Label		Yes	No
	Yes	TP = 3	FN = 2
	No	FP = 1	TN = 4

$$\begin{aligned}\text{Error Rate} &= \frac{\# \text{ incorrect predictions}}{\# \text{ total predictions}} \\ &= 1 - \text{Accuracy Rate} \\ &= 1 - 0.7 = 0.3\end{aligned}$$

# Misclassifications in Confusion Matrix

		Predicted Class Label	
True Class Label		Yes	No
	Yes	TP = 3	FN = 2
	No	FP = 1	TN = 4

High value means  
classifying Negative  
class is problematic

High value means  
classifying Positive  
class is problematic

# Confusion Matrix

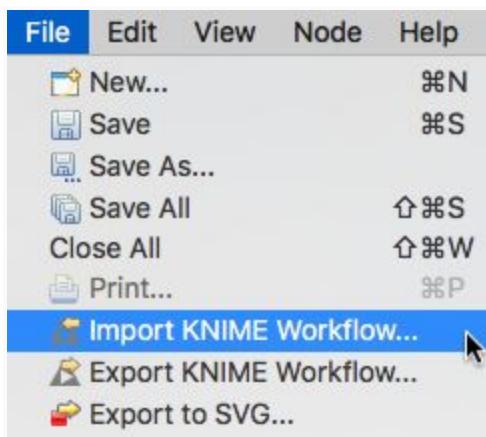
		Predicted Class Label	
		Yes	No
True Class Label	Yes	TP	FN
	No	FP	TN

# Completed KNIME Workflows

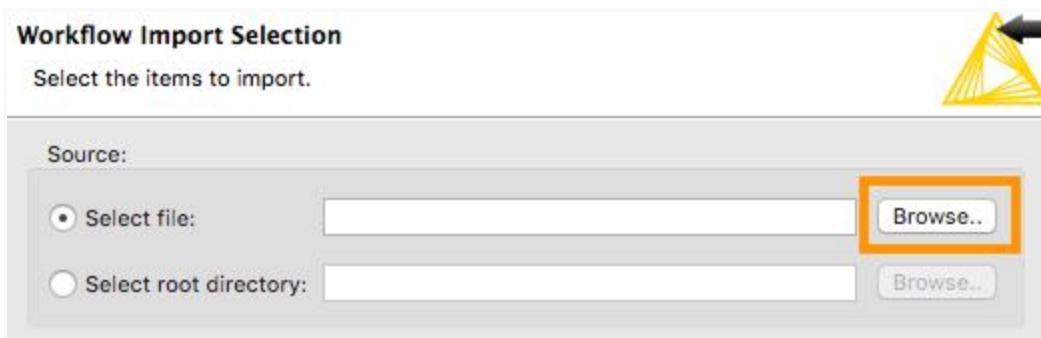
The completed KNIME workflows for this course are available here:

[KNIME-workflows.zip](#)

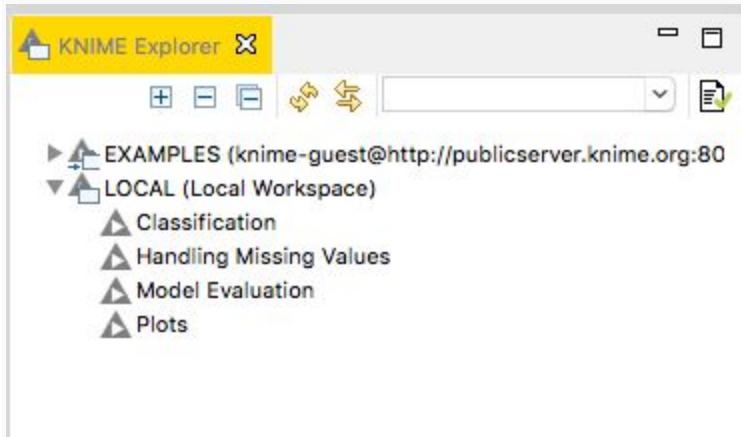
To load these in KNIME, select *File -> Import KNIME Workflow*:



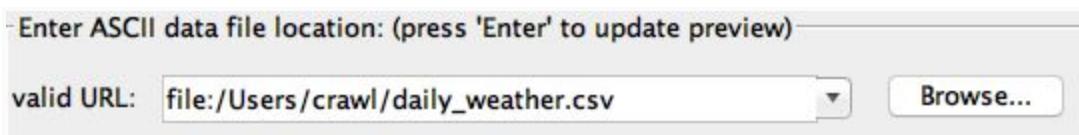
Next, click on the *Browse* button and select the workflow to import:



Finally, click on *Finish*. The workflow will be added to your *LOCAL* Workspace, and you can load it by double-clicking on the name:



Finally, double-click on the *File Reader* Node and change the *valid URL* parameter to the location of *daily\_weather.csv* on your computer:



# Comparing Classification Results for KNIME and Spark

You may have noticed that the classification results from KNIME and Spark MLlib decision trees are not exactly the same. This document describes the differences in the setup that can lead to this disparity.

## Random Seed for Partitioning

In the step to partition data into training and test sets, random sampling is used in both KNIME and Spark MLlib. However, note that different random seed values are used. A value of 12345 is used for KNIME, and 13234 is used for Spark. Different seed values will generate different random number sequences. Since the selection of which samples go into the training vs. the test partition is based on these random number sequences, the contents of the training and test datasets will be different with different seed values. Consequently, different training and test sets will change the performance results of the classifier. Cross validation is a common approach to address this variability in performance with a single partitioning of the data.

## Partitioning into Training and Test Sets

You also may have noticed that the sizes of the training and test sets are different between KNIME and Spark. This is due to the different sampling methods that they use to partition data.

Spark uses a sampling method that does not generate a fixed sample size. So if we specify that the test size should be 20% of the available data, the resulting test size is only approximately 20%, not necessarily exactly.

KNIME uses a different way to randomly select samples to be placed in either training and test set. Due to the different sampling techniques, the contents of the training and test sets are different for KNIME and Spark MLlib.

## Decision Tree

KNIME and Spark MLlib use different methods to constrain the complexity of the decision tree model. In both, the minimum number of samples in a node can be specified as a stopping criterion for growing the tree. If the number of samples reaches this threshold, the node is not split.

In Spark, you can also specify the maximum depth of the tree as another stopping criterion (the `maxDepth` parameter). KNIME does not have this option in its Decision Tree Learner node.

KNIME does have an option for pruning the tree, however. The default setting in the Decision Tree Learner node uses ‘Reduced error pruning’. This prunes the tree by replacing a node with the class of the majority of the samples in that node, if doing so does not decrease the accuracy of the classifier. Spark’s `DecisionTreeClassifier` method does not have an option to prune the tree.

## Performance Results

As we can see, there are several differences in the setup for a decision tree classifier in KNIME and Spark MLlib. These differences result in different trained models, and consequently, different classification performance numbers.

# Evaluation of Decision Tree in KNIME

## Learning Objectives

At the end of this activity, you will be able to perform the following operations in KNIME:

1. Create and interpret a confusion matrix for a decision tree
2. Determine the accuracy rate of a decision tree model
3. Use highlighting to analyze classification errors

## Problem Description

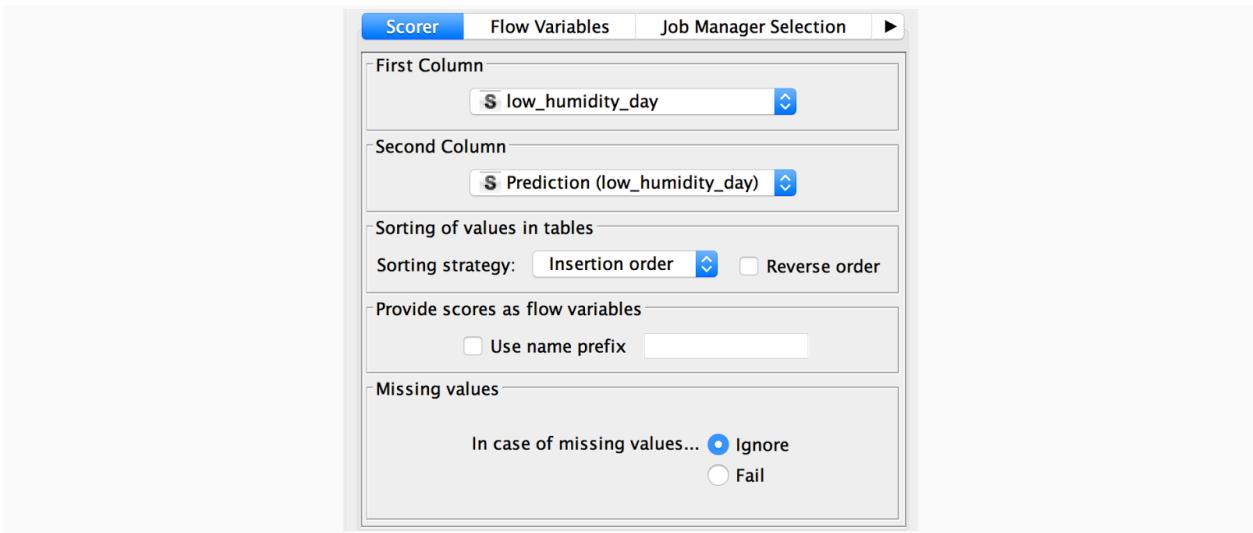
With the decision tree classifier built, we now need to evaluate its performance.

## Steps

### Generate a Confusion Matrix and Determine Accuracy Rate

A confusion matrix shows the type of errors and correct classifications that a classifier makes. It can be generated using a **Scorer** node.

1. Open the Decision Tree Workflow that you created from the Classification Hands-On reading.
2. Connect a **Scorer** node to the existing **Decision Tree Predictor**.
3. The Scorer Configure Dialog should look like this by default. Click OK.



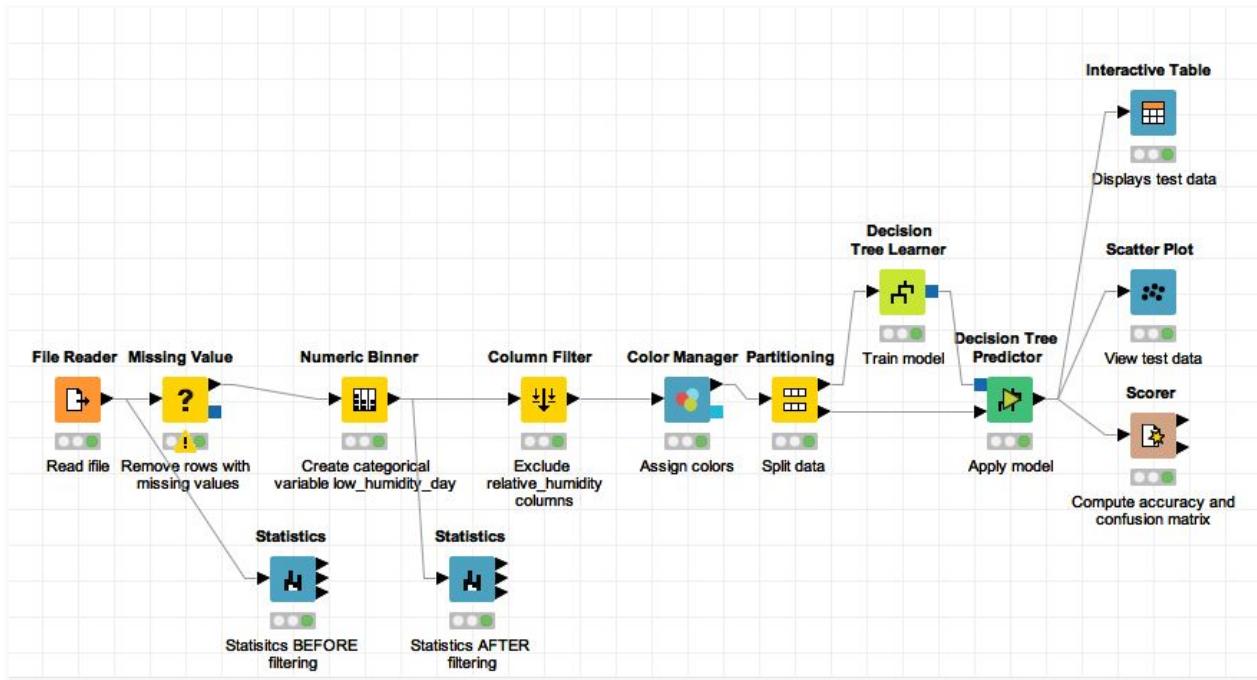
Execute and view the **Scorer** node. It shows the confusion matrix, along with the accuracy of the prediction. Here you should see an accuracy rate of 80.282% if you followed all the hands-on instructions.

low_humidity_day \ Prediction (low_humidity_day)		humidity_low	humidity_not_low
humidity_low	76	24	
humidity_not_low	18	95	
Correct classified: 171		Wrong classified: 42	
Accuracy: 80.282 %		Error: 19.718 %	
Cohen's kappa ( $\kappa$ ) 0.603			

From the confusion matrix, we see the following:

- There are 213 samples in the test data set (the sum of all the values in the confusion matrix)
- 76 humidity\_low samples were correctly classified
- 95 humidity\_not\_low samples were correctly classified
- The accuracy rate is  $(76 + 95) / 213 = 171 / 213 = 80.282\%$
- 24 humidity\_low samples were incorrectly classified as humidity\_not\_low
- 18 humidity\_not\_low samples were incorrectly classified as humidity\_low
- The error rate is  $(24 + 18) / 213 = 42 / 213 = 19.718\%$

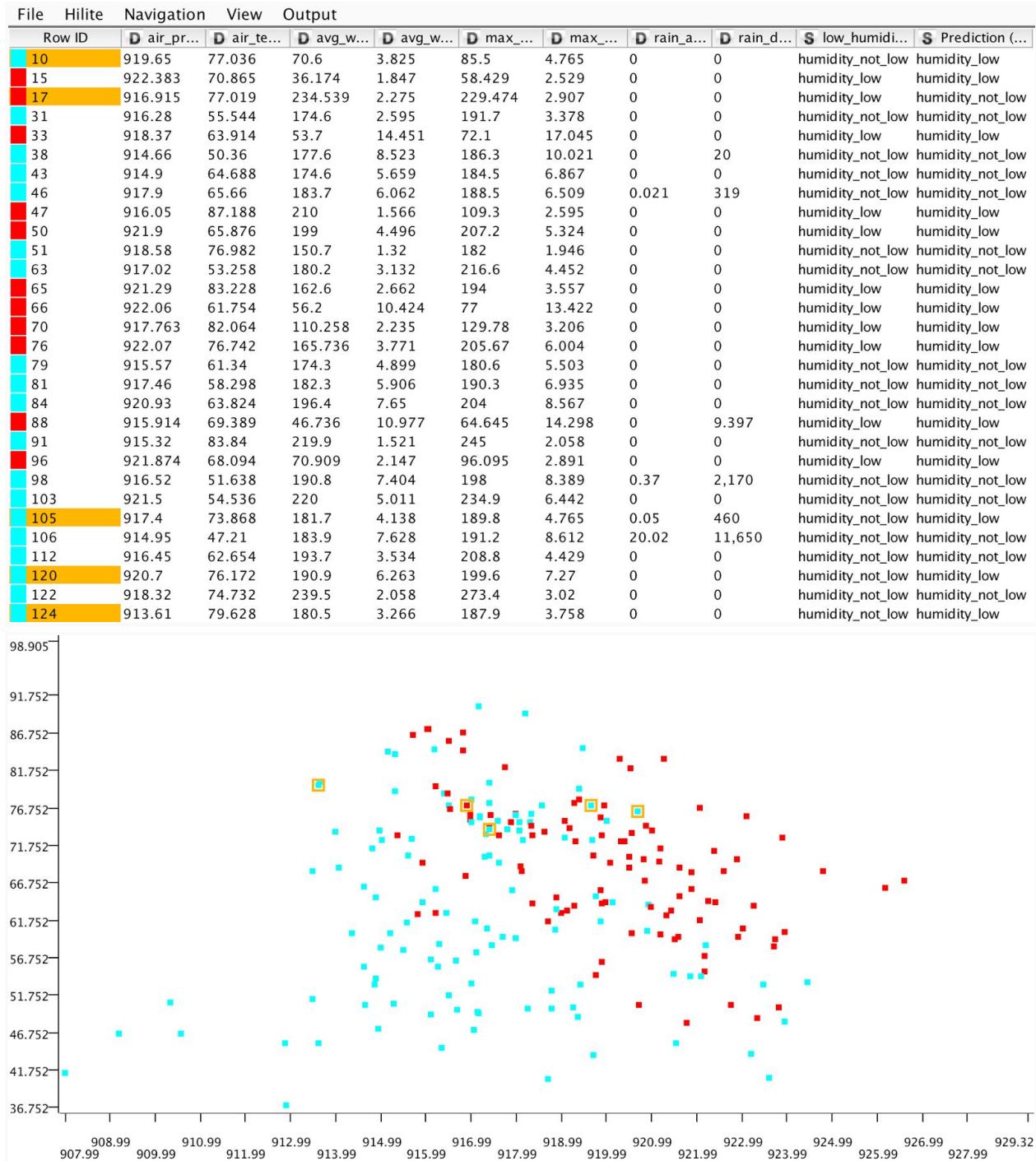
# Use Highlighting and Scatter Plot to Analyze Classification Errors



A good way to enhance analysis of incorrect predictions is to visualize them. This can be accomplished using a feature called **hiliting**, and viewing the data in a **Scatter Plot** node.

1. Connect an **Interactive Table** node to the **Decision Tree Predictor**.
2. Execute and view this **Interactive Table** to see the input values for each sample (row), along with the ACTUAL/TRUE low\_humidity\_day value and the PREDICTED low\_humidity\_day value. The red and blue squares next to the Row ID color-codes the actual/true label (low or not). You can use this table to analyze samples whose true value differs from the predicted value (incorrect prediction).
3. Connect a **Scatter Plot** node to the **Decision Tree Predictor**.
4. Execute and view the **Scatter Plot** node, and place the window side-by-side with the **Interactive Table** window.
5. Go through the the table looking for rows with predictions that are different from the true value.
6. When you find such a row, click anywhere on that row. At the top of the window click **Hilite > Hilite Selected**. This will make that row yellow in the table and in the Scatter Plot. It may be easiest to use the up and down arrow keys to navigate the rows of the table. In this example, we are just going to highlight the first 5 misclassifications.
7. Do this for any row with a misclassification. This allows you to pinpoint the misclassified samples and analyze them further. Analyzing the misclassified samples can bring insight into how to improve model performance. For example, if many samples with

avg\_temperature\_9am between 60 and 70 degrees are misclassified, this suggests that more samples with these values for avg\_temperature\_9am are needed to train the model.



Save Your Workflow

Save your workflow using <control>-s on Windows or <command>-s on Mac, or selecting File>Save or File>Save As.

# Evaluation of Decision Tree in Spark

By the end of this activity, you will be able to perform the following in Spark:

1. Determine the accuracy of a classifier model
  2. Display the confusion matrix for a classifier model

In this activity, you will be programming in a Jupyter Python Notebook. If you have not already started the Jupyter Notebook server, see the instructions in the Reading *Instructions for Starting Jupyter*.

**Step 1. Open Jupyter Python Notebook.** Open a web browser by clicking on the web browser icon at the top of the toolbar:



Navigate to `localhost:8889/tree/Downloads/big-data-4`:



Open the model evaluation notebook by clicking on *model-evaluation.ipynb*:



**Step 2. Load predictions.** Execute the first cell to load the classes used in this activity:

```
In [1]: from pyspark.sql import SQLContext
        from pyspark.ml.evaluation import MulticlassClassificationEvaluator
        from pyspark.mllib.evaluation import MulticlassMetrics
```

Execute the next cell to load the predictions CSV file that we created at the end of the [Week 3 Hands-On Classification in Spark](#) into a DataFrame:

Step 3. **Compute accuracy.** Let's create an instance of *MulticlassClassificationEvaluator* to determine the accuracy of the predictions:

```
In [3]: evaluator = MulticlassClassificationEvaluator(  
    labelCol="label", predictionCol="prediction", metricName="precision")
```

The first two arguments specify the names of the label and prediction columns, and the third argument specifies that we want the overall precision.

We can compute the accuracy by calling *evaluate()*:

```
In [4]: accuracy = evaluator.evaluate(predictions)  
print("Accuracy = %g " % (accuracy))
```

```
Accuracy = 0.809524
```

Step 4. **Display confusion matrix.** The *MulticlassMetrics* class can be used to generate a confusion matrix of our classifier model. However, unlike *MulticlassClassificationEvaluator*, *MulticlassMetrics* works with RDDs of numbers and not DataFrames, so we need to convert our *predictions* DataFrame into an RDD.

If we use the *rdd* attribute of *predictions*, we see this is an *RDD* of *Rows*:

```
In [5]: predictions.rdd.take(2)
```

```
Out[5]: [Row(prediction=1.0, label=1.0), Row(prediction=1.0, label=1.0)]
```

Instead, we can map the RDD to *tuple* to get an RDD of numbers:

```
In [6]: predictions.rdd.map(tuple).take(2)
```

```
Out[6]: [(1.0, 1.0), (1.0, 1.0)]
```

Let's create an instance of *MulticlassMetrics* with this RDD:

```
In [7]: metrics = MulticlassMetrics(predictions.rdd.map(tuple))
```

**NOTE:** the above command can take longer to execute than most Spark commands when first run in the notebook.

The `confusionMatrix()` function returns a Spark *Matrix*, which we can convert to a Python Numpy array, and transpose to view:

```
In [8]: metrics.confusionMatrix().toArray().transpose()
```

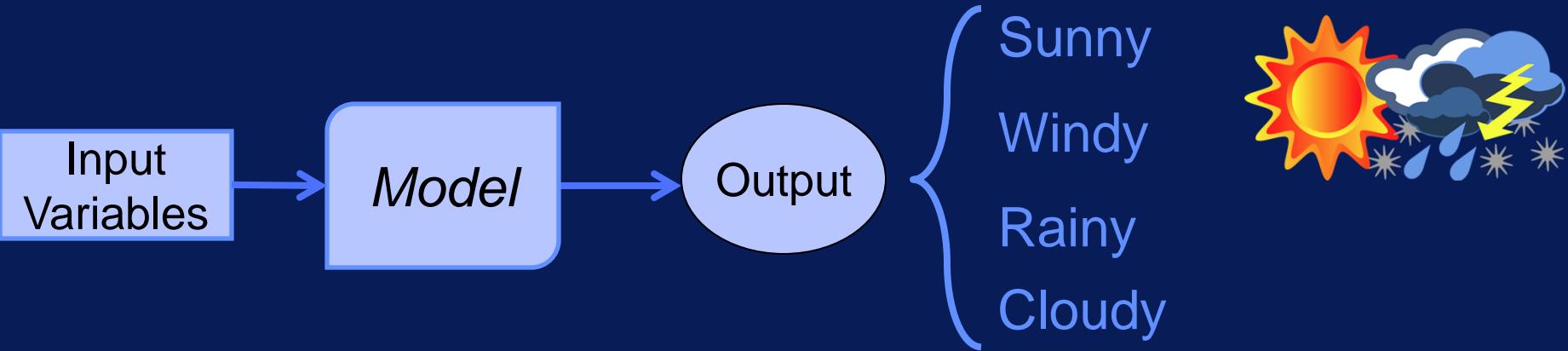
```
Out[8]: array([[ 87.,  26.],
   [ 14.,  83.]])
```

# Regression

# After this video you will be able to..

- Define what regression is
- Explain the difference between regression and classification
- Name some applications of regression

# Classification Review



Classification:  
Given input variables,  
predict category

# Regression



Regression:  
Given input variables,  
predict numeric value

# Regression Examples

- **Forecast** high temperature for next day
- **Estimate** average house price for a region
- **Determine** demand for a new product
- **Predict** power usage



# Regression is Supervised

Input Variables

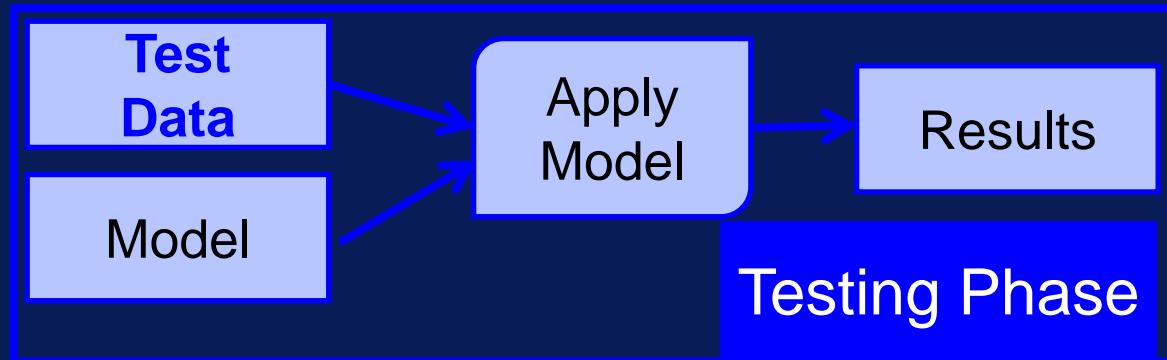
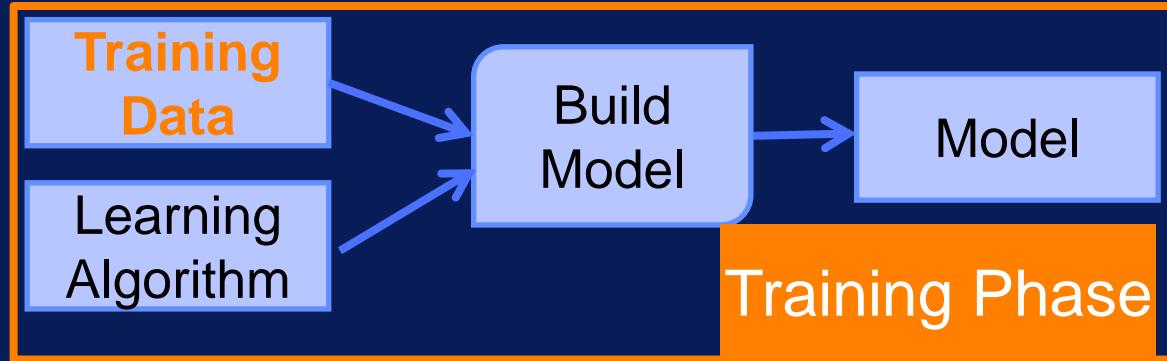
Target Variable

Target is provided

The diagram illustrates a regression model. At the top, two boxes are labeled: 'Input Variables' in blue and 'Target Variable' in red. A blue bracket groups the first three columns ('Today's High', 'Today's Low', and 'Month') under 'Input Variables'. A red arrow points from the 'Target Variable' box down to the fourth column, 'Tomorrow's High', in the table below. The table contains four rows of data.

Today's High	Today's Low	Month	Tomorrow's High
79	64	July	81
60	45	October	58
68	49	May	65
57	47	January	54

# Training vs. Testing Phases



# Datasets

## Training Data

Adjust model parameters

## Validation Data

Determine when to stop training (avoid overfitting)

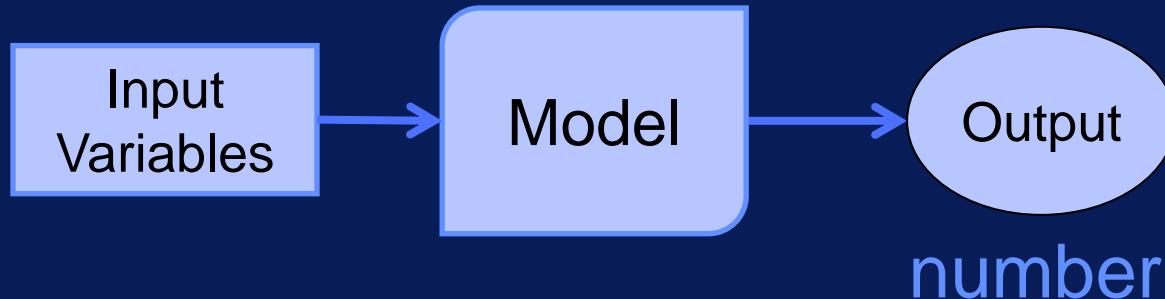
Estimate generalization performance

## Test Data

Evaluate performance on new data

# Regression Main Points

- Predict number from input variables
- Regression is a supervised task
- Target variable is numerical



# Linear Regression

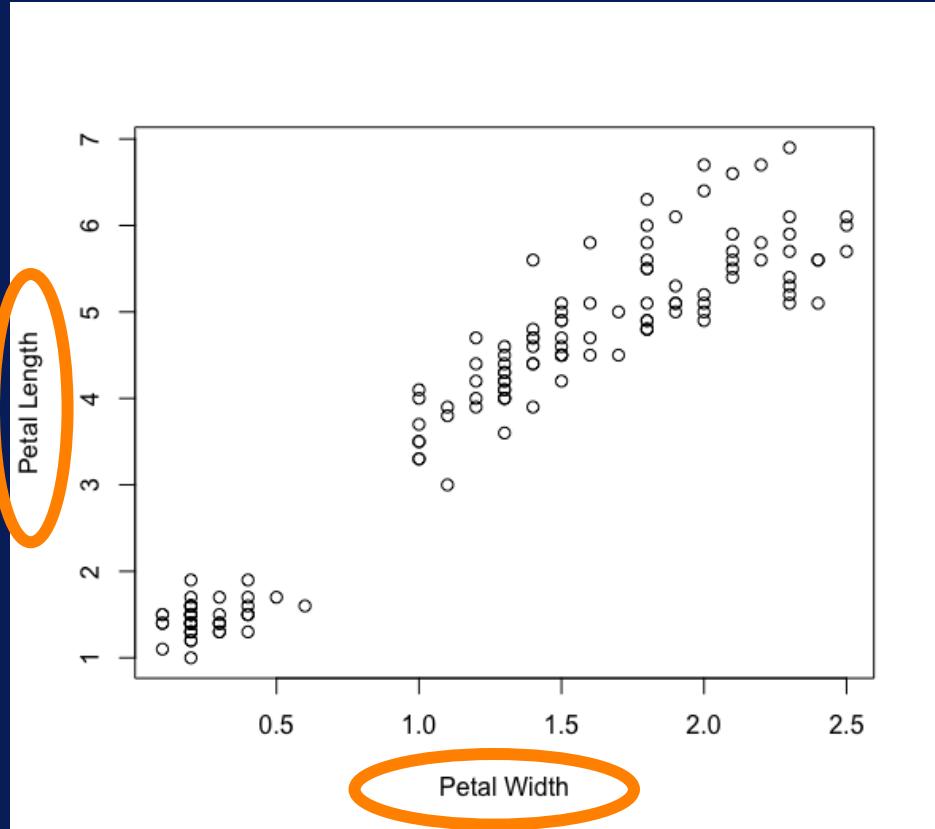
# After this video you will be able to..

- Describe how linear regression works
- Discuss how least squares is used in linear regression
- Define simple and multiple linear regression

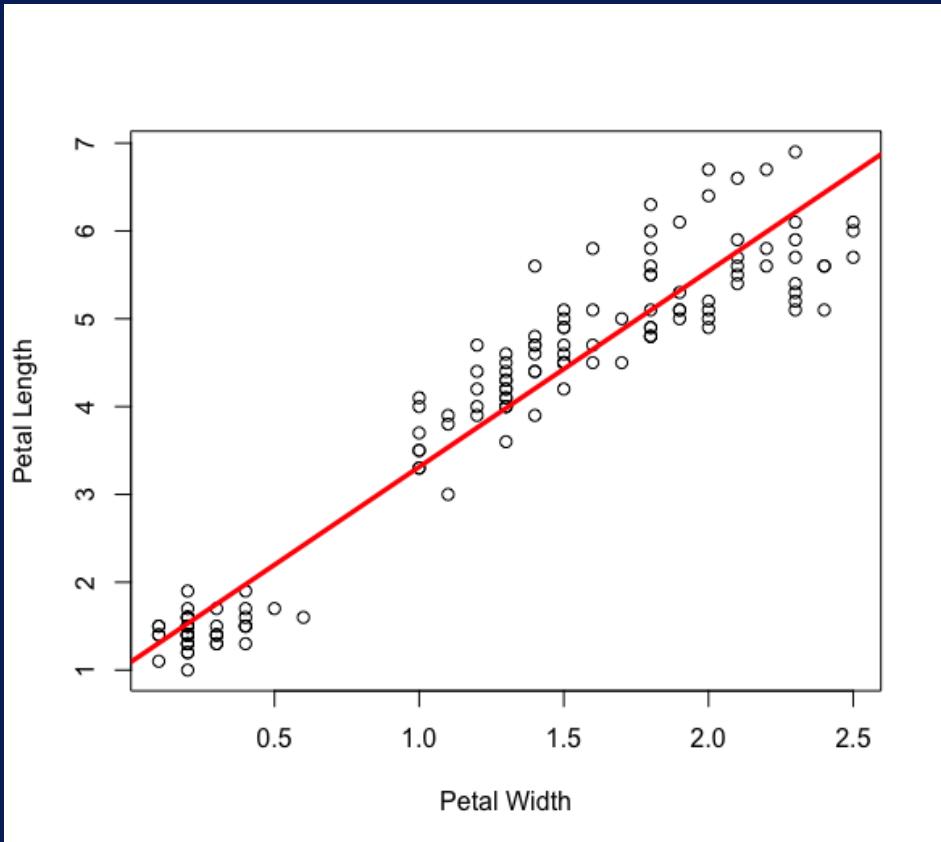
# Linear Regression

- Captures relationship between numerical output and input variables
- Relationship is modeled as linear

# Linear Regression Model



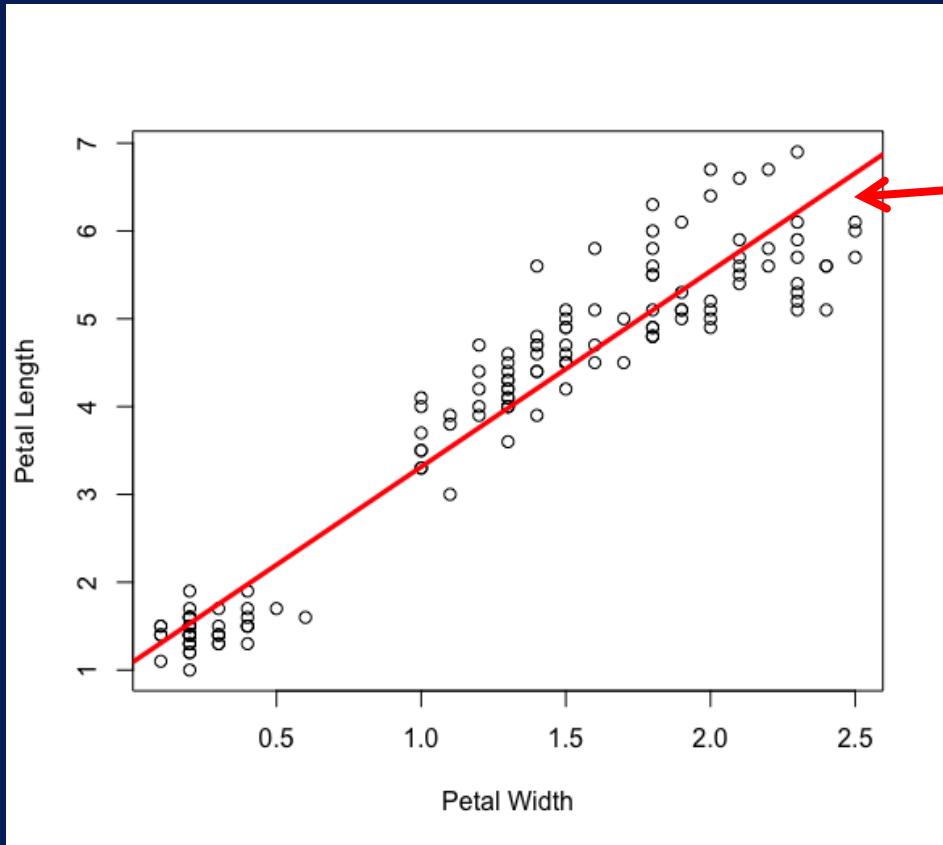
# Linear Regression Model



# Regression Task:

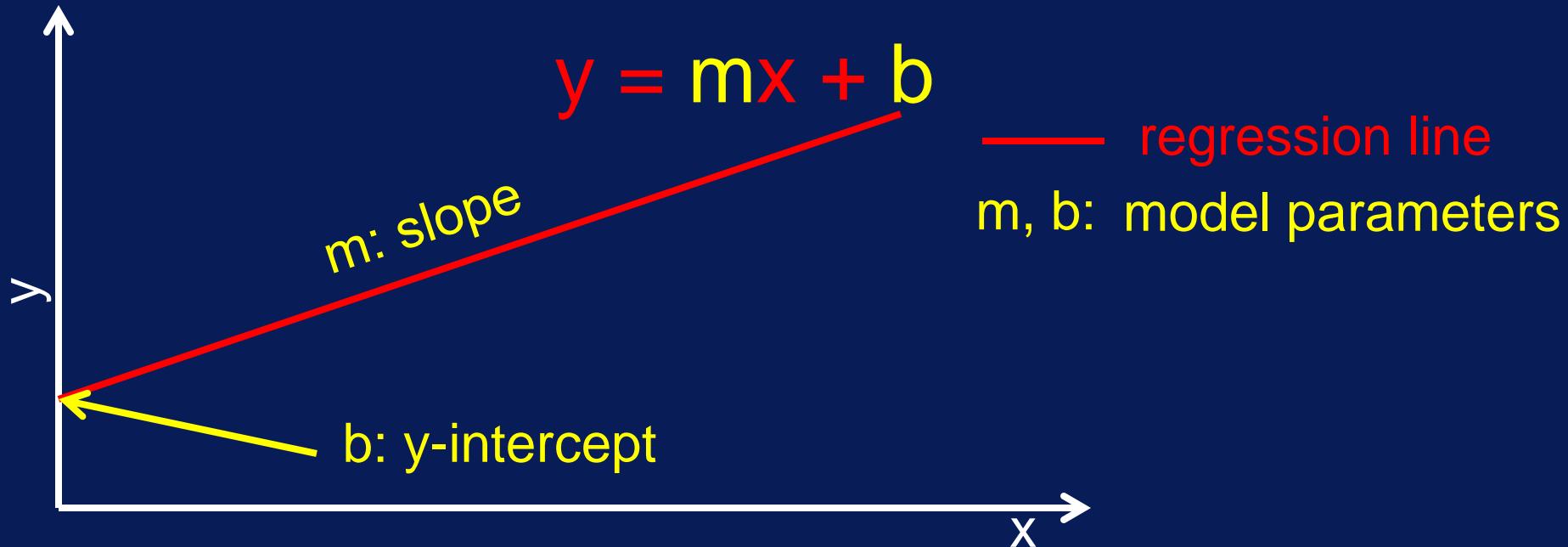
Given petal width, predict petal length.

# Linear Regression Model



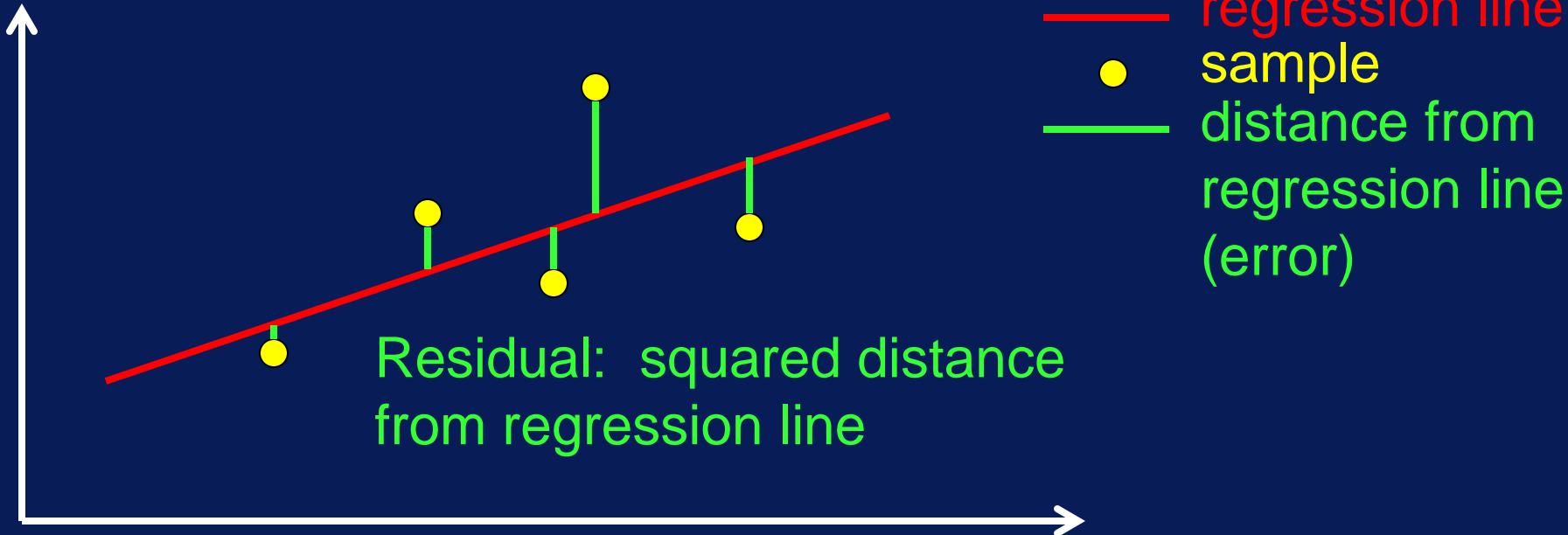
regression line

# Least Squares Algorithm



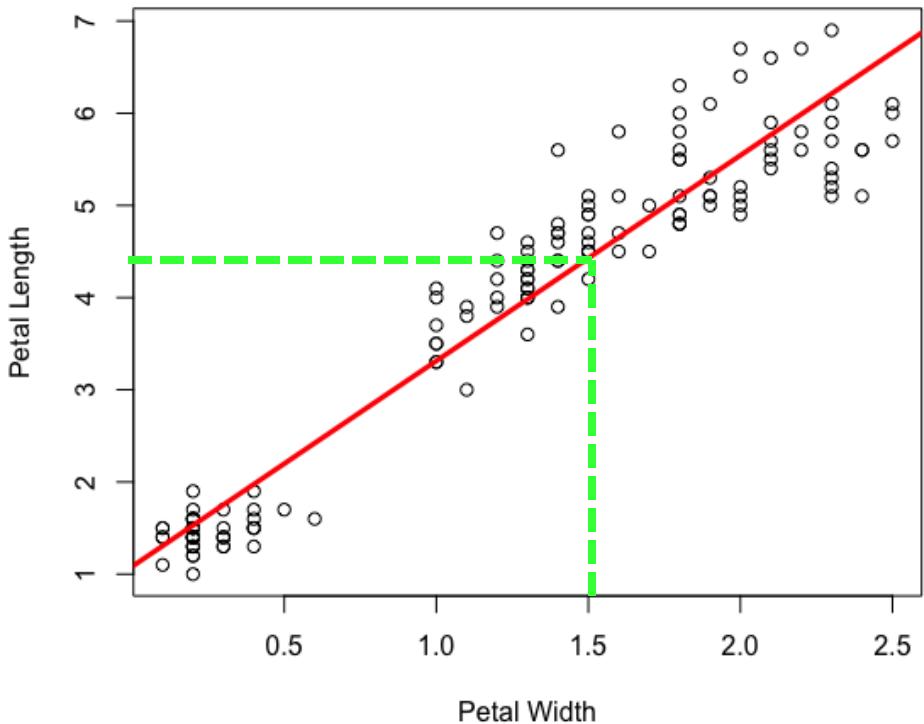
Training linear regression model adjusts  
model parameters to fit samples

# Least Squares Method



**Goal:** Find regression line that makes sum of residuals as small as possible

# Linear Regression Model



**Applying model:**

Given petal width = 1.5,  
prediction is

petal length = 4.5

# Types of Linear Regression

**Simple Linear  
Regression**

**Multiple Linear  
Regression**

var1

var1

var2

...  
varN

Input has one  
variable

Input has >1  
variables

# Linear Regression Summary

- Captures linear relationship between numerical output and input variables
- Model can be fitted using least squares

# Cluster Analysis

# After this video you will be able to..

- Articulate the goal of cluster analysis
- Discuss whether cluster analysis is supervised or unsupervised.
- List some ways that cluster results can be applied

# Cluster Analysis Overview

Goal: Organize similar items into groups.

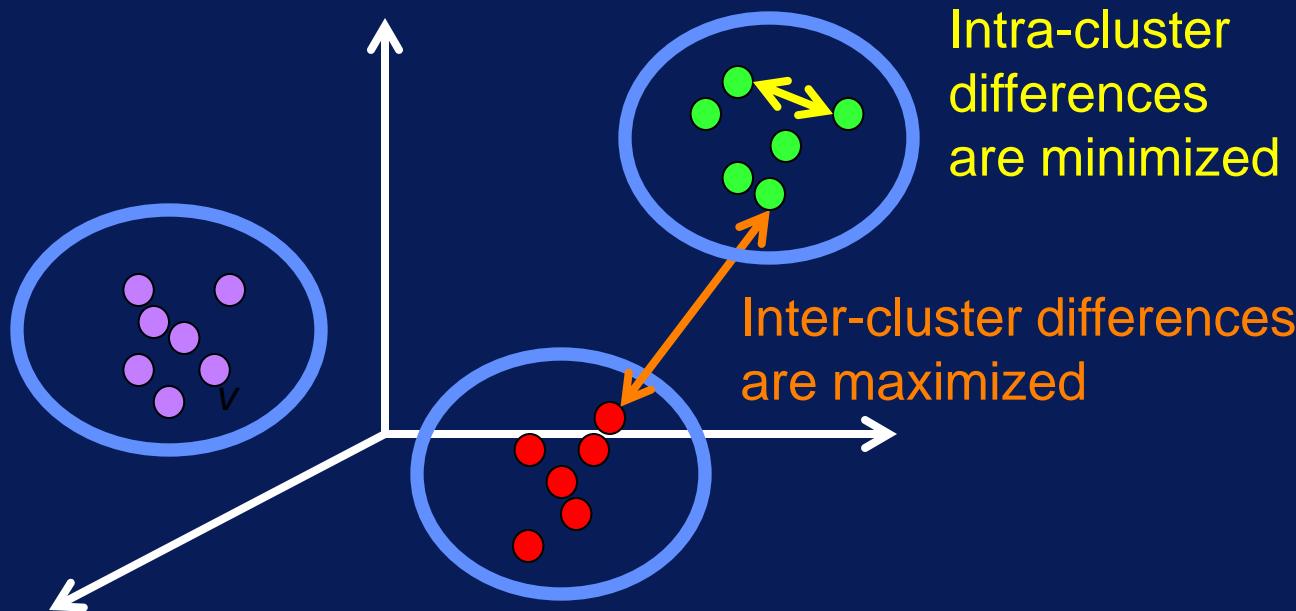


# Cluster Analysis Examples

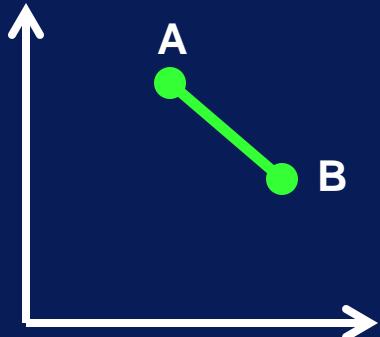
- Segment customer base into groups
- Characterize different weather patterns for a region
- Group news articles into topics
- Discover crime hot spots

# Cluster Analysis

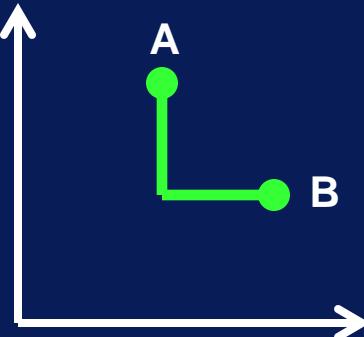
- Divides data into clusters
- Similar items are placed in same cluster



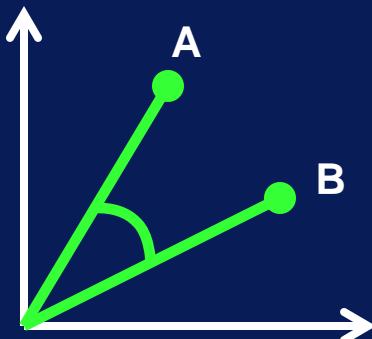
# Similarity Measures



Euclidean Distance



Manhattan Distance

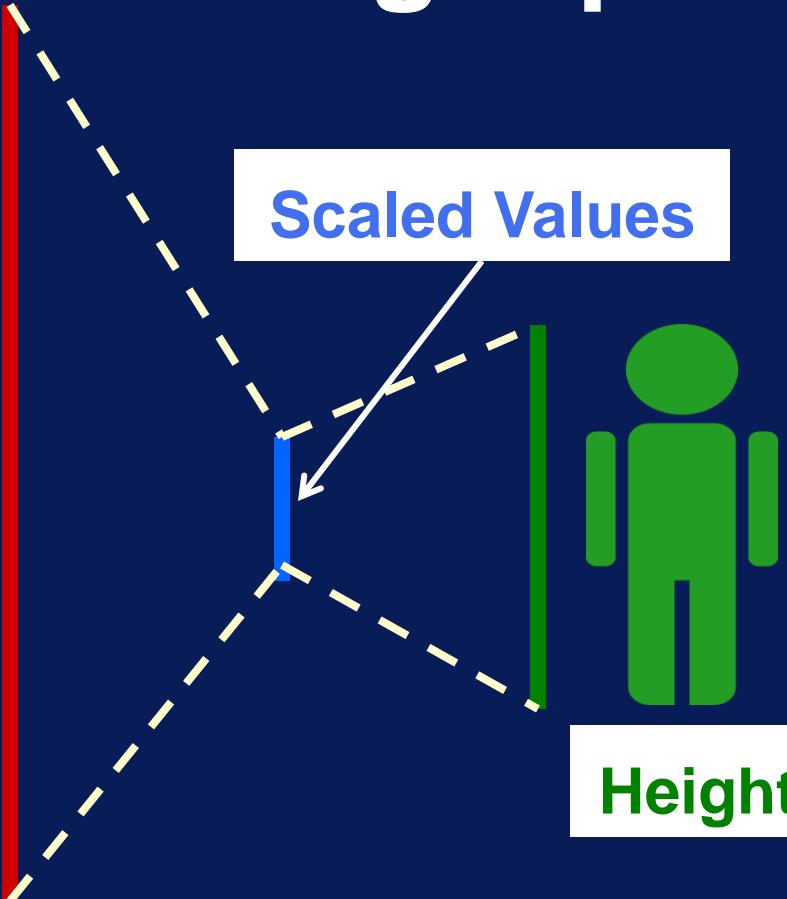


Cosine Similarity

# Normalizing Input Variables



Weight



Height

# Cluster Analysis Notes

Unsupervised

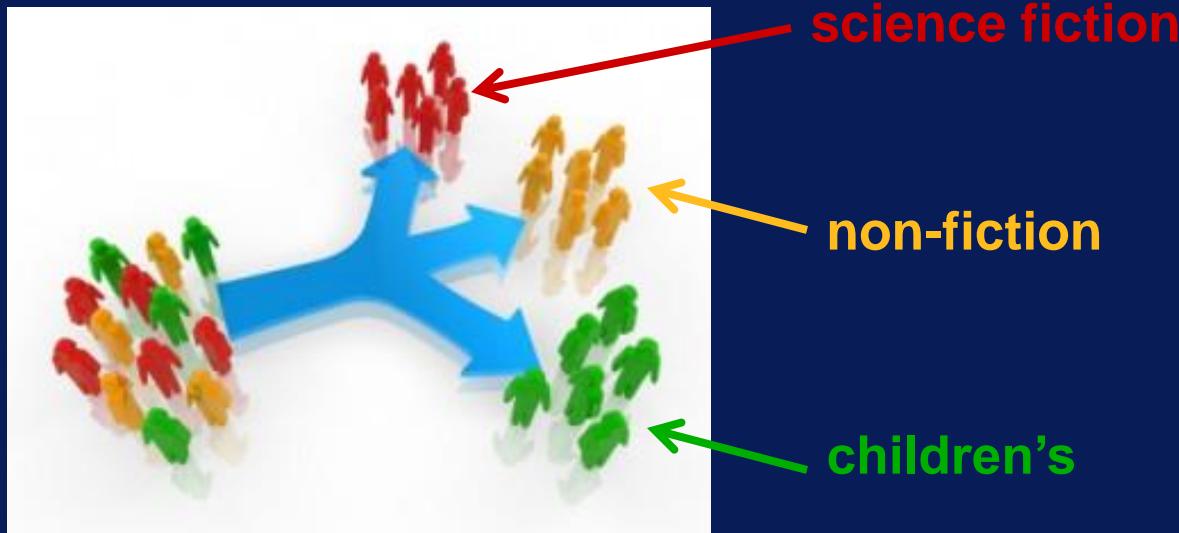
There is no  
'correct' clustering

Clusters don't  
come with labels

Interpretation and analysis required to make sense of clustering results!

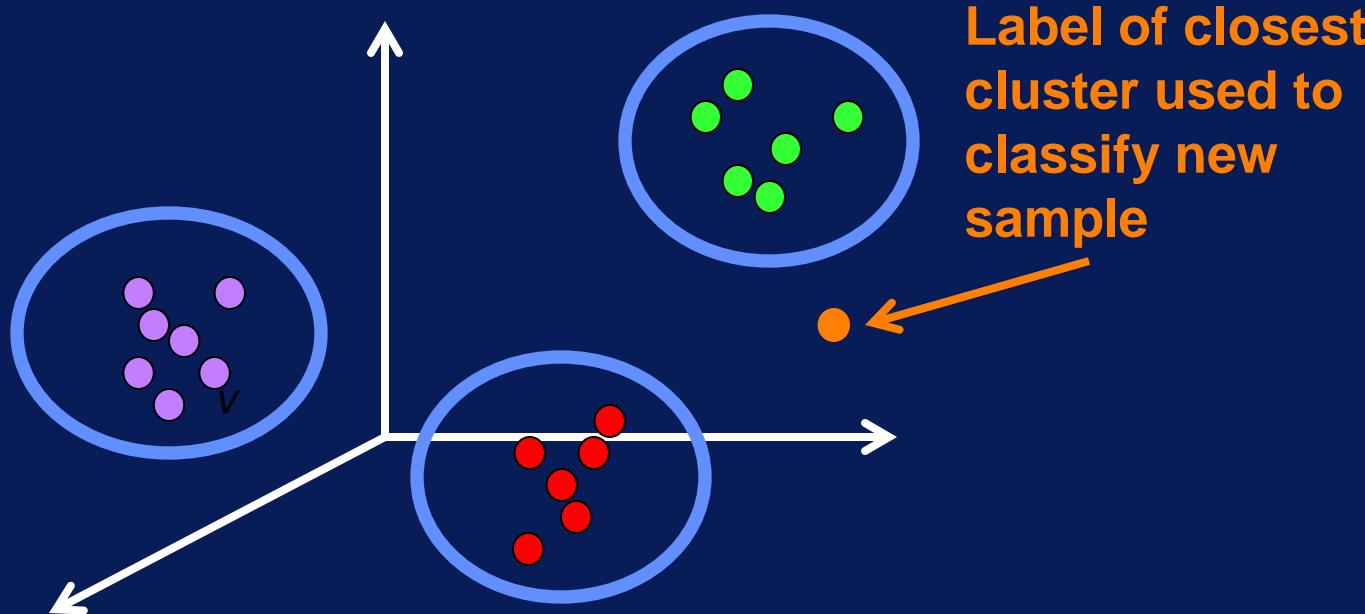
# Uses of Cluster Results

- **Data segmentation**
  - Analysis of each segment can provide insights



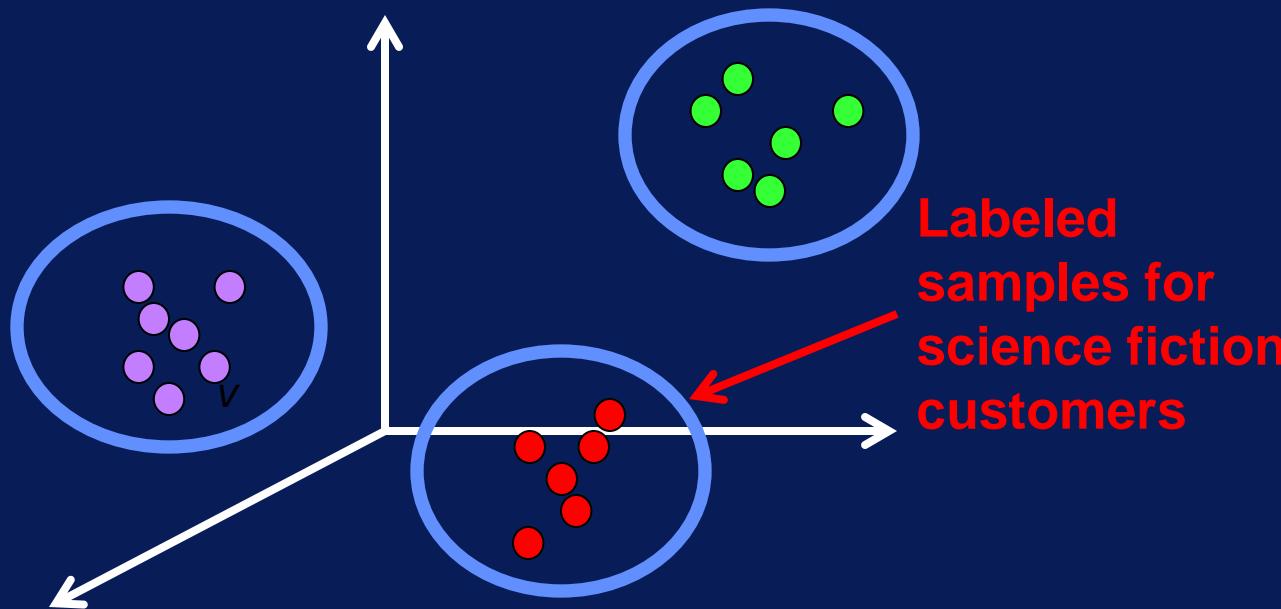
# Uses of Cluster Results

- Categories for classifying new data
  - New sample assigned to closest cluster



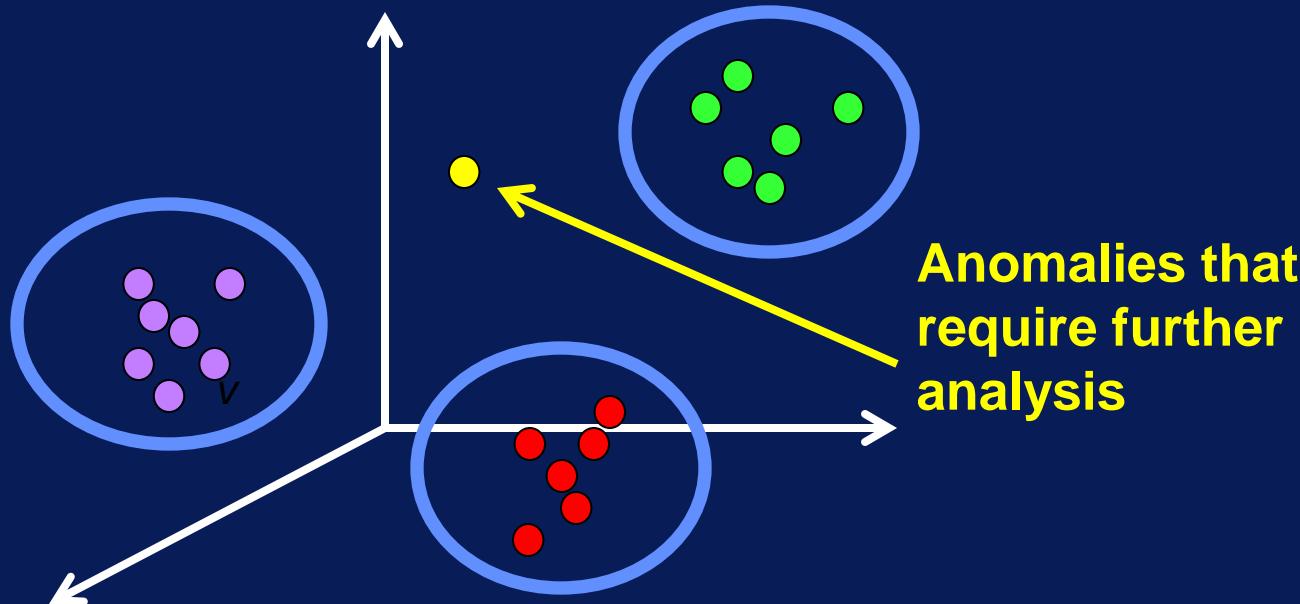
# Uses of Cluster Results

- Labeled data for classification
  - Cluster samples used as labeled data



# Uses of Cluster Results

- Basis for anomaly detection
  - Cluster outliers are anomalies



# Cluster Analysis Summary

- Organize similar items into groups
- Analyzing clusters often leads to useful insights about data
- Clusters require analysis and interpretation

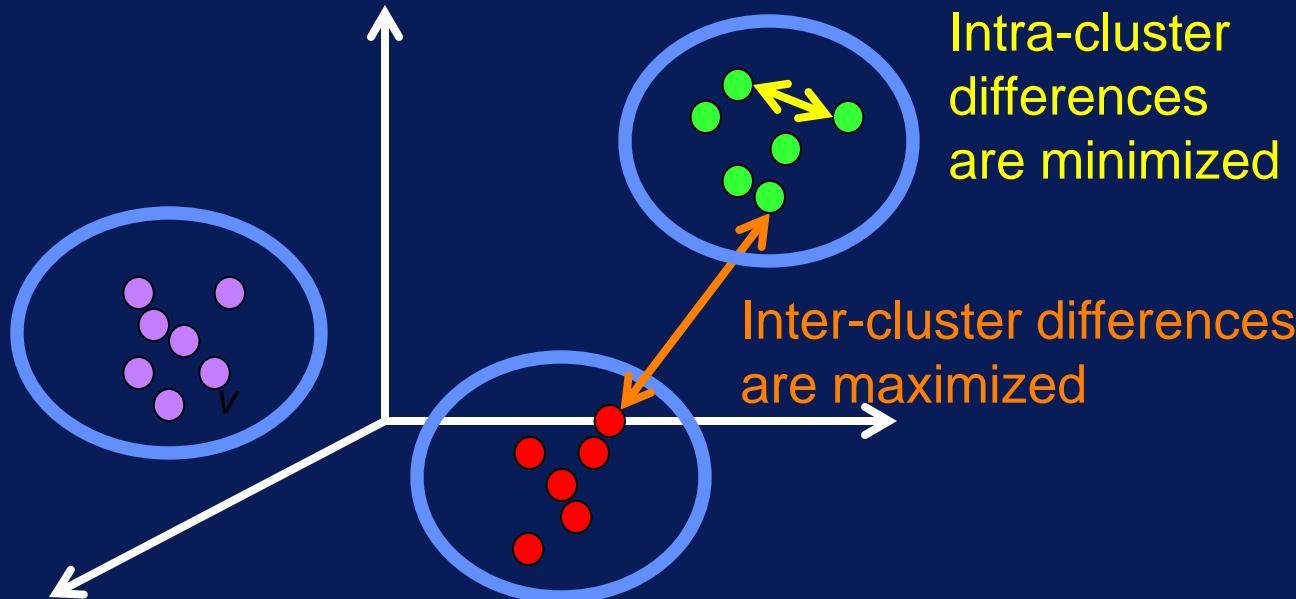
# k-Means Clustering

# After this video you will be able to..

- Describe the steps in the k-means algorithm
- Explain what the ‘k’ stands for in k-means
- Define what a cluster centroid is

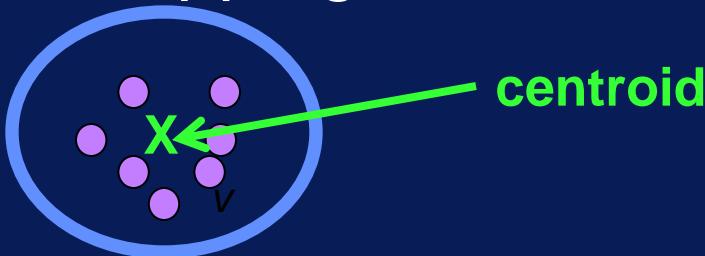
# Cluster Analysis

- Divides data into clusters
- Similar items are in same cluster

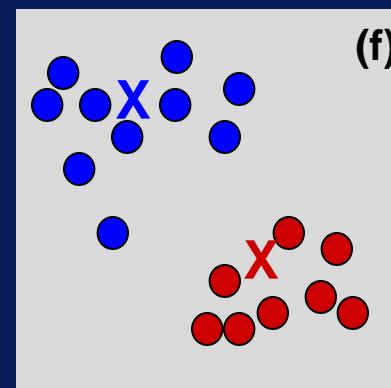
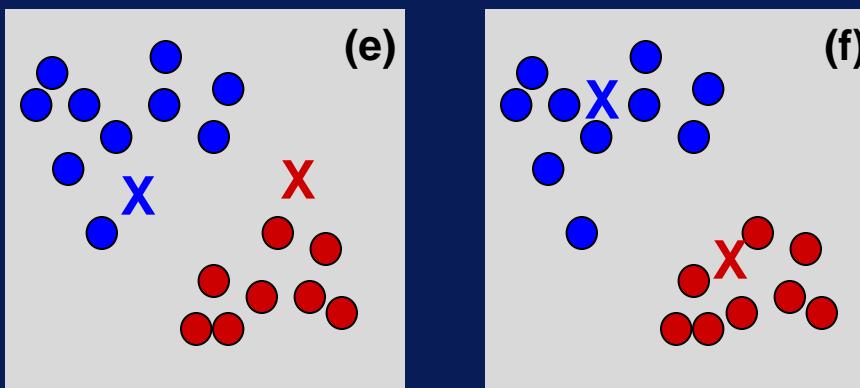
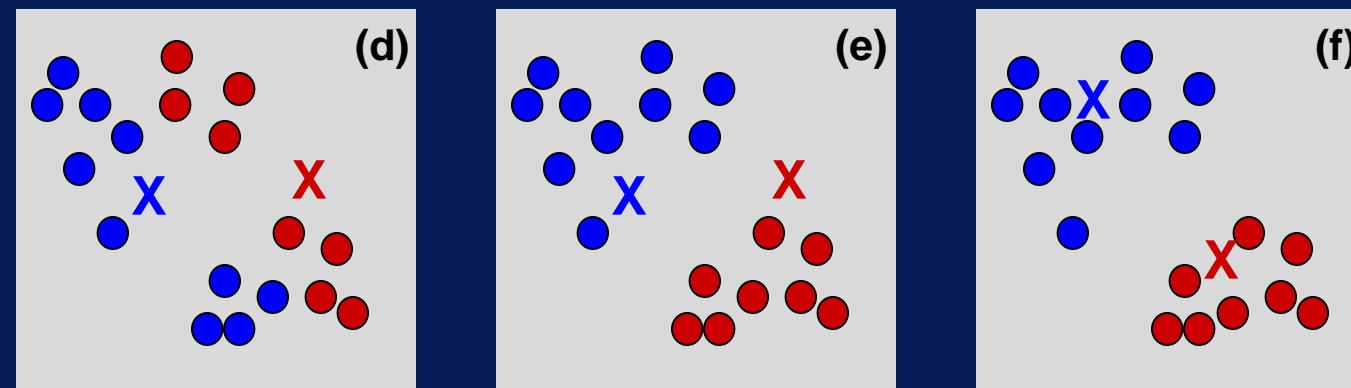
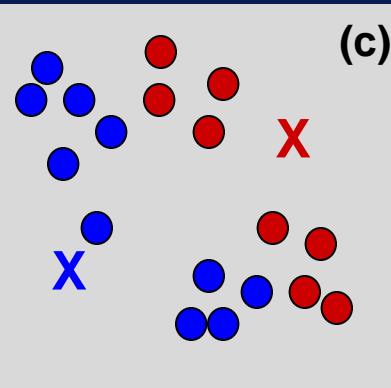
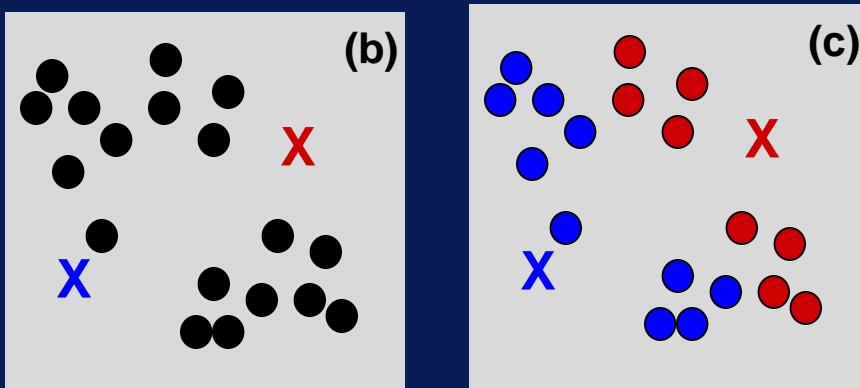
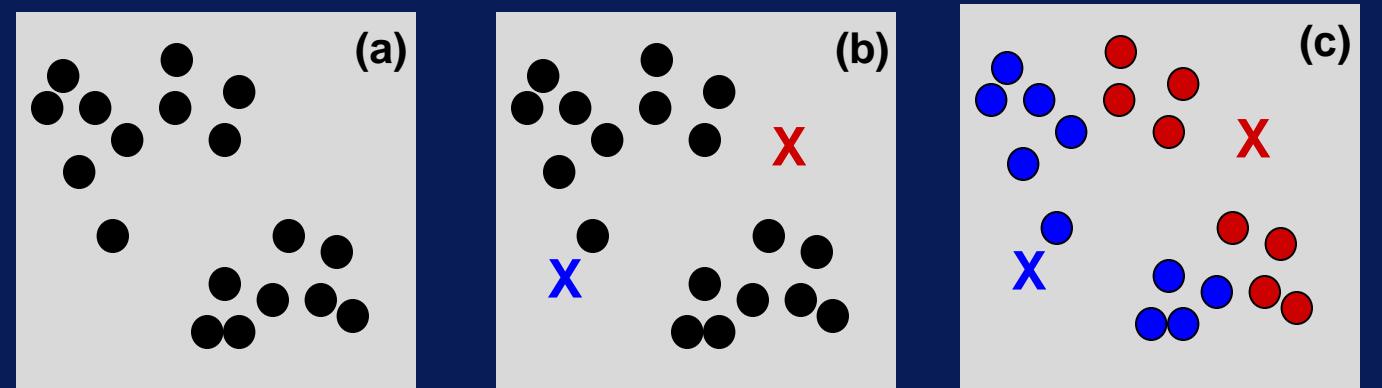


# k-Means Algorithm

- Select k initial centroids (cluster centers)
- Repeat
  - Assign each sample to closest centroid
  - Calculate mean of cluster to determine new centroid
- Until some stopping criterion is reached



# k-Means



Re-calculate centroids      Assign samples      Re-calculate centroids

# Choosing Initial Centroids

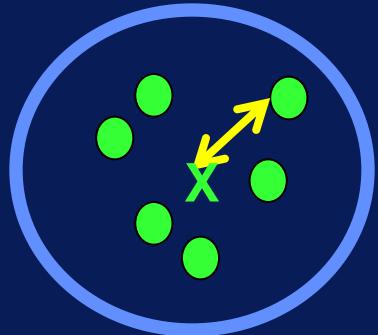
**Issue:**

Final clusters are sensitive to initial centroids

**Solution:**

Run k-means multiple times with  
different random initial centroids,  
and choose best results

# Evaluating Cluster Results



error = distance between sample & centroid

squared error =  $\text{error}^2$

Sum of squared errors  
between all samples & centroid

Sum over all clusters



WSSE

Within-Cluster Sum  
of Squared Error

# Using WSSE

$WSSE_1 < WSSE_2$



WSSE1 is better *numerically*

Caveats:

- Does not mean that cluster set 1 is more 'correct' than cluster set 2
- Larger values for k will always reduce WSSE

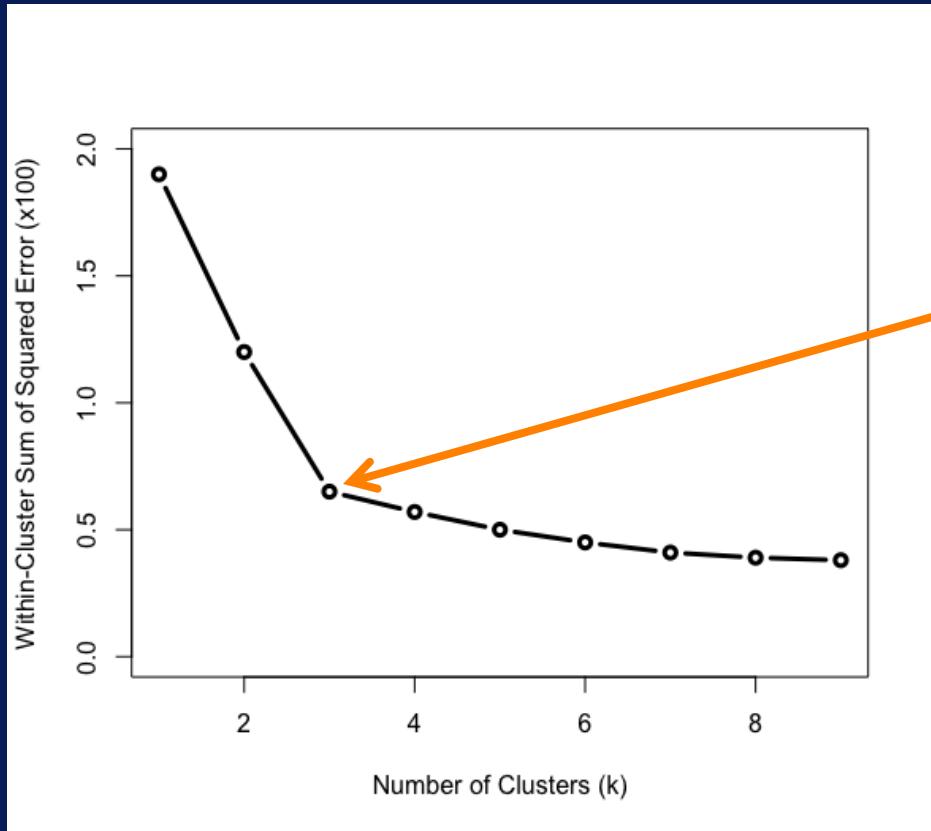
# Choosing Value for k

- **Approaches:**

- Visualization
- Application-Dependent
- Data-Driven

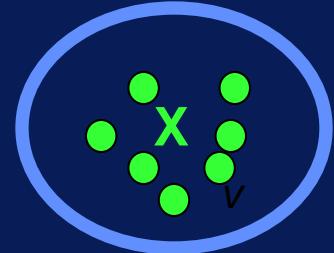
k = ?

# Elbow Method for Choosing k



“Elbow” suggests value  
for  $k$  should be 3

# Stopping Criteria

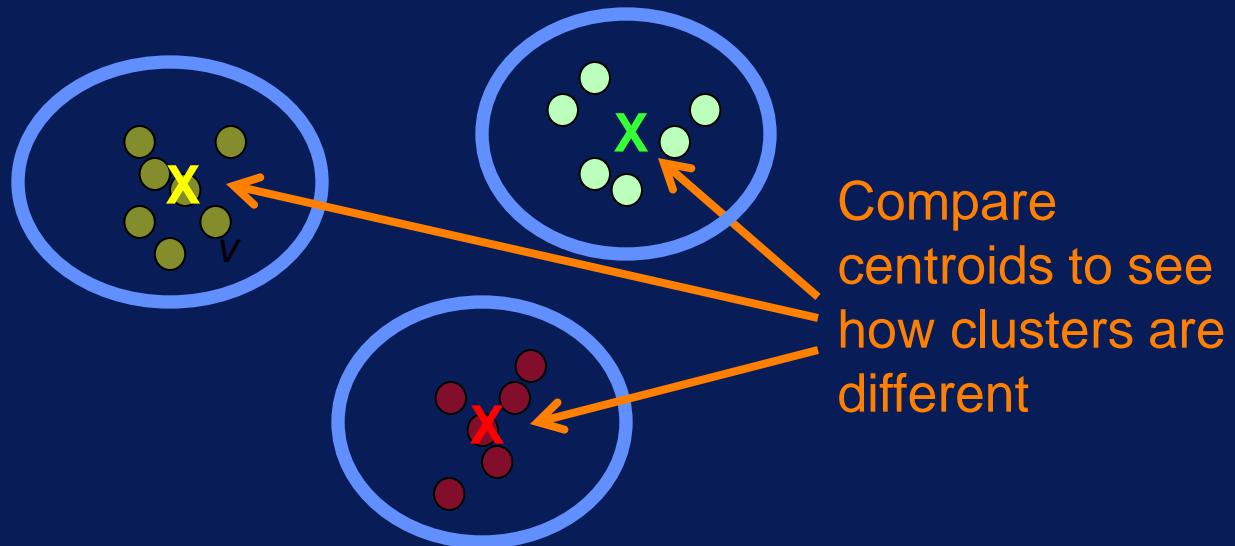


**When to stop iterating?**

- No changes to centroids
- Number of samples changing clusters is below threshold

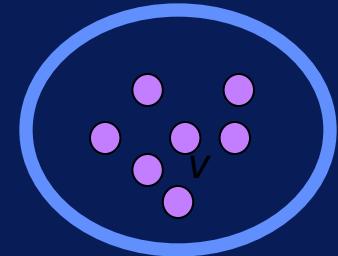
# Interpreting Results

- **Examine cluster centroids**
  - How are clusters different?



# K-Means Summary

- **Classic algorithm for cluster analysis**
- **Simple to understand and implement and is efficient**
- **Value of k must be specified**
- **Final clusters are sensitive to initial centroids**

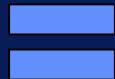


# Association Analysis

# After this video you will be able to..

- Explain what association analysis entails
- List some applications of association analysis
- Define what an item set is

# Association Analysis



Goal: Find rules to capture associations between items.

# Association Analysis Examples

- Have sales on related items often purchased together
- Recommend items based on purchase/browsing history
- Identify effective treatments to patients

# Analysis Association Overview

ID	Items
1	diapers, bread, milk
2	bread, diapers, beer, eggs
3	milk, diapers, beer, butter
4	bread, milk, diapers, beer
5	bread, milk, diapers, butter

Item Sets

$\{bread, milk\} \Rightarrow \{diapers\}$   
 $\{milk\} \Rightarrow \{bread\}$

Rules

If bread and milk  
are bought, then  
diaper is also  
bought

# Association Analysis Steps

## 1. Create item sets

{bread}

{butter}

{bread, milk}

{milk, beer}

## 2. Identify frequent item sets

{bread}

{bread, milk}

## 3. Generate rules

{bread, milk} => {diapers}

# Association Analysis Notes

Unsupervised

Usefulness of rules  
is subjective

Need to determine  
how to use rules



Interpretation and analysis required to  
make sense of resulting rules!

# Association Analysis Summary

- Find rules to capture associations between items
- Rules have intuitive appeal
- Resulting rules require analysis and interpretation using domain knowledge



# Association Analysis in Detail

# After this video you will be able to..

- Define the terms ‘support’ and ‘confidence’
- Describe the steps in association analysis
- Explain how association rules are formed from item sets

# Association Analysis Steps

## 1. Create item sets

{bread}

{butter}

{bread, milk}

{bread, beer}

## 2. Identify frequent item sets

{bread}

{bread, beer}

## 3. Generate rules

{bread, milk} => {diapers}

# Analysis Association Dataset

ID	Items
1	diapers, bread, milk
2	bread, diapers, beer, eggs
3	milk, diapers, beer, butter
4	bread, milk, diapers, beer
5	bread, milk, diapers, butter

Item Sets

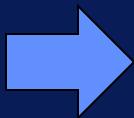
$\{bread, milk\} \Rightarrow \{diapers\}$   
 $\{milk\} \Rightarrow \{bread\}$

If bread and milk  
are bought, then  
diapers are also  
bought

Rules

# Create Item Sets

ID	Items
1	diaper, bread, milk
2	bread, diaper, beer, eggs
3	milk, diaper, beer, butter
4	bread, milk, diaper, beer
5	bread, milk, diaper, butter



## 1-Item Sets

Item	Support
bread	4/5
butter	2/5
milk	4/5
beer	3/5
diaper	5/5
eggs	1/5

Support =  
frequency of  
item set

'diaper' occurs in all transactions

'eggs' occurs only once, in transaction 2

# Create Item Sets

ID	Items
1	diaper, bread, milk
2	bread, diaper, beer, eggs
3	milk, diaper, beer, butter
4	bread, milk, diaper, beer
5	bread, milk, diaper, butter

## 1-Item Sets

minimum support = 3/5

Item	Support
{bread}	4/5
{butter}	2/5
{milk}	4/5
{beer}	3/5
{diaper}	5/5
{eggs}	1/5

Remove these item sets since they have low support.

# Create Item Sets

ID	Items
1	diaper, bread, milk
2	bread, diaper, beer, eggs
3	milk, diaper, beer, butter
4	bread, milk, diaper, beer
5	bread, milk, diaper, butter



## 2-Item Sets

Item	Support
{bread,milk}	3/5
{bread,beer}	2/5
{bread,diaper}	4/5
{milk,beer}	2/5
{milk,diaper}	4/5
{beer,diaper}	3/5

minimum support = 3/5

1-item sets:  
{bread}, {milk}, {diaper}

'beer' and 'diaper' occur  
together 3 times, in  
transactions 2, 3, & 4

# Create Item Sets

ID	Items
1	diaper, bread, milk
2	bread, diaper, beer, eggs
3	milk, diaper, beer, butter
4	bread, milk, diaper, beer
5	bread, milk, diaper, butter

## 2-Item Sets

Item	Support
{bread,milk}	3/5
{bread,beer}	2/5
{bread,diaper}	4/5
{milk,beer}	2/5
{milk,diaper}	4/5
{beer,diaper}	3/5

minimum support = 3/5

## 1-item sets:

{bread}, {milk}, {diaper}

Remove these item sets  
since they have low support.

# Create Item Sets

ID	Items
1	diaper, bread, milk
2	bread, diaper, beer, eggs
3	milk, diaper, beer, butter
4	bread, milk, diaper, beer
5	bread, milk, diaper, butter

1-item sets:

{bread},  
{milk},  
{diaper}

2-item sets:

{bread,milk},  
{bread,diaper},  
{milk,diaper},  
{beer,diaper}

3-Item Sets

Item	Support
{bread,milk, diaper}	3/5

minimum support = 3/5

Only 3-item set with  
support > minimum support

# Frequent Item Sets

ID	Items
1	diaper, bread, milk
2	bread, diaper, beer, eggs
3	milk, diaper, beer, butter
4	bread, milk, diaper, beer
5	bread, milk, diaper, butter

## 1-Item Sets

Item	Support
{bread}	4/5
{milk}	4/5
{diaper}	5/5

minimum support = 3/5

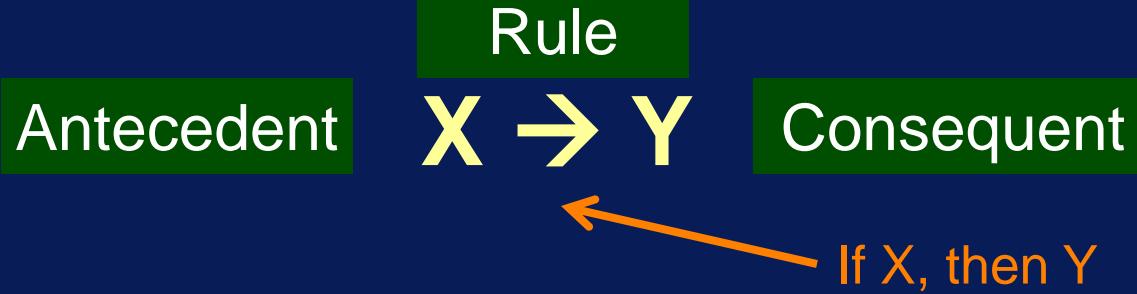
## 2-Item Sets

Item	Support
{bread,milk}	3/5
{bread,diaper}	4/5
{milk,diaper}	4/5
{beer,diaper}	3/5

## 3-Item Sets

Item	Support
{bread,milk, diaper}	3/5

# Rule Terms



Rule Confidence

$$\text{conf } (X \rightarrow Y) = \frac{\text{supp } (X \cup Y)}{\text{supp } (X)}$$

support for X & Y together  
support for X

Itemset Support

$$\text{supp } (X) = \frac{\# \text{ transactions with } X}{\text{total } \# \text{ transactions}}$$

# Rule Generation & Pruning

frequent item sets  association rules

each k-item set   $2^k - 2$  rules!

frequent item sets  significant rules

Use rule confidence to  
constrain rule generation

Keep rule if confidence > minimum confidence

# Rule Example

min confidence = 0.95

$$\text{conf } (X \rightarrow Y) = \frac{\text{supp } (X \cup Y)}{\text{supp } (X)}$$



ID	Items
1	diaper, bread, milk
2	bread, diaper, beer, eggs
3	milk, diaper, beer, butter
4	bread, milk, diaper, beer
5	bread, milk, diaper, butter

## 3-Item Sets

Item	Support
{bread,milk, diaper}	3/5

Candidate rule: {bread,milk}  $\rightarrow$  {diaper}

$$\text{conf} = \frac{\text{supp } (\text{bread,milk,diaper})}{\text{supp } (\text{bread,milk})} = \frac{3/5}{3/5} = \frac{3}{3} = 1.0$$



Candidate rule: {bread,diaper}  $\rightarrow$  {milk}

$$\text{conf} = \frac{\text{supp } (\text{bread,diaper,milk})}{\text{supp } (\text{bread,diaper})} = \frac{3/5}{4/5} = \frac{3}{4} = 0.75$$

# Association Analysis Algorithms

- Use different methods to make efficient:
  - item set creation
  - rule generation efficient
- Algorithms:  
**Apriori      FP Growth      Eclat**

# Association Analysis Steps

- Item sets created from data
- Frequent item sets identified using support
- Rules generated from frequent item sets and pruned using confidence



# Description of Minute Weather Dataset

The minute weather dataset comes from the same source as the daily weather dataset that was used for the hands-on activities in the previous modules. The difference is that the minute weather dataset contains raw sensor measurements captured at one-minute intervals, not processed like the daily weather dataset. The data is in the file *minute\_weather.csv*, which is a comma-separated file.

As with the daily weather data, this data comes from a weather station located in San Diego, California. The weather station is equipped with sensors that capture weather-related measurements such as air temperature, air pressure, and relative humidity. Data was collected for a period of three years, from September 2011 to September 2014, to ensure that sufficient data for different seasons and weather conditions is captured.

Each row in *minute\_weather.csv* contains weather data captured for a one-minute interval. Each row, or sample, consists of the following variables:

Variable	Description	Unit of Measure
rowID	unique number for each row	NA
hpwren_timestamp	timestamp of measure	year-month-day hour:minute:second
air_pressure	air pressure measured at the timestamp	hectopascals
air_temp	air temperature measure at the timestamp	degrees Fahrenheit
avg_wind_direction	wind direction averaged over the minute before the timestamp	degrees, with 0 means coming from the North, and increasing clockwise

avg_wind_speed	wind speed averaged over the minute before the timestamp	meters per second
max_wind_direction	highest wind direction in the minute before the timestamp	degrees, with 0 being North and increasing clockwise
max_wind_speed	highest wind speed in the minute before the timestamp	meters per second
min_wind_direction	smallest wind direction in the minute before the timestamp	degrees, with 0 being North and increasing clockwise
min_wind_speed	smallest wind speed in the minute before the timestamp	meters per second
rain_accumulation	amount of accumulated rain measured at the timestamp	millimeters
rain_duration	length of time rain has fallen as measured at the timestamp	seconds
relative_humidity	relative humidity measured at the timestamp	percent

# Cluster Analysis in Spark

## Problem Description

This activity guides you through the process of performing cluster analysis on a dataset using k-means. In this activity, we will perform cluster analysis on the *minute-weather.csv* dataset using the k-means algorithm. Recall that this dataset contains weather measurements such as temperature, relative humidity, etc., from a weather station in San Diego, California, collected at one-minute intervals. The goal of cluster analysis on this data is to identify different weather patterns for this weather station.

## Learning Objectives

By the end of this activity, you will be able to:

1. Scale all features so that each feature is zero-normalized
2. Create an "elbow" plot, the number of clusters vs. within-cluster sum-of-squared errors, to determine a value for k, the number of clusters in k-means
3. Perform cluster analysis on a dataset using k-means
4. Create parallel coordinates plots to analyze cluster centers

In this activity, you will be programming in a Jupyter Python Notebook. If you have not already started the Jupyter Notebook server, see the instructions in the Reading *Instructions for Starting Jupyter*.

**Step 1. Open Jupyter Python Notebook.** Open a web browser by clicking on the web browser icon at the top of the toolbar:



Navigate to `localhost:8889/tree/Downloads/big-data-4`:



Open the clustering notebook by clicking on `clustering.ipynb`:



Step 2. **Load minute weather data.** Execute the first cell to load the classes used in this activity:

```
In [1]: from pyspark.sql import SQLContext
from pyspark.ml.clustering import KMeans
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.feature import StandardScaler
from notebooks import utils
%matplotlib inline
```

Execute the second cell to load the minute weather data in *minute\_weather.csv*:

```
In [2]: sqlContext = SQLContext(sc)
df = sqlContext.read.load('file:///home/cloudera/Downloads/big-data-4/minute_weather.csv',
                         format='com.databricks.spark.csv',
                         header='true',inferSchema='true')
```

Step 3. **Subset and remove unused data.** Let's count the number of rows in the DataFrame:

```
In [3]: df.count()
```

```
Out[3]: 1587257
```

There are over 1.5 million rows in the DataFrame. Clustering this data on your computer in the Cloudera VM can take a long time, so let's only one-tenth of the data. We can subset by calling *filter()* and using the *rowID* column:

```
In [4]: filteredDF = df.filter((df.rowID % 10) == 0)
filteredDF.count()
```

```
Out[4]: 158726
```

Let's compute the summary statistics using *describe()*:

```
In [5]: filteredDF.describe().toPandas().transpose()
```

Out[5]:

	0	1	2	3	4
<b>summary</b>	count	mean	stddev	min	max
<b>rowID</b>	158726	793625.0	458203.9375103623	0	1587250
<b>air_pressure</b>	158726	916.8301614102434	3.051716552830638	905.0	929.5
<b>air_temp</b>	158726	61.851589153636304	11.833569210641757	31.64	99.5
<b>avg_wind_direction</b>	158680	162.15610032770354	95.27820101905898	0.0	359.0
<b>avg_wind_speed</b>	158680	2.775214897907747	2.057623969742642	0.0	31.9
<b>max_wind_direction</b>	158680	163.46214393748426	92.45213853838689	0.0	359.0
<b>max_wind_speed</b>	158680	3.400557726241518	2.4188016208098886	0.1	36.0
<b>min_wind_direction</b>	158680	166.77401688933702	97.44110914784567	0.0	359.0
<b>min_wind_speed</b>	158680	2.1346641038568754	1.7421125052424393	0.0	31.6
<b>rain_accumulation</b>	158725	3.178453299732825E-4	0.011235979086039813	0.0	3.12
<b>rain_duration</b>	158725	0.4096267128681682	8.665522693479772	0.0	2960.0
<b>relative_humidity</b>	158726	47.609469778108206	26.214408535062027	0.9	93.0

The weather measurements in this dataset were collected during a drought in San Diego. We can count the how many values of rain accumulation and duration are 0:

```
In [6]: filteredDF.filter(filteredDF.rain_accumulation == 0.0).count()
```

Out[6]: 157812

```
In [7]: filteredDF.filter(filteredDF.rain_duration == 0.0).count()
```

Out[7]: 157237

Since most the values for these columns are 0, let's drop them from the DataFrame to speed up our analyses. We can also drop the *hpwren\_timestamp* column since we do not use it.

```
In [8]: workingDF = filteredDF.drop('rain_accumulation').drop('rain_duration').drop('hpwren_timestamp')
```

Let's drop rows with missing values and count how many rows were dropped:

```
In [9]: before = workingDF.count()
workingDF = workingDF.na.drop()
after = workingDF.count()
before - after
```

```
Out[9]: 46
```

Step 4. **Scale the data.** Since the features are on different scales (e.g., air pressure values are in the 900's, while relative humidities range from 0 to 100), they need to be scaled. We will scale them so that each feature will have a value of 0 for the mean, and a value of 1 for the standard deviation.

First, we will combine the columns into a single vector column. Let's look at the columns in the DataFrame:

```
In [10]: workingDF.columns
```

```
Out[10]: ['rowID',
          'air_pressure',
          'air_temp',
          'avg_wind_direction',
          'avg_wind_speed',
          'max_wind_direction',
          'max_wind_speed',
          'min_wind_direction',
          'min_wind_speed',
          'relative_humidity']
```

We do not want to include *rowID* since it is the row number. The minimum wind measurements have a high correlation to the average wind measurements, so we will not include them either. Let's create an array of the columns we want to combine, and use *VectorAssembler* to create the vector column:

```
In [11]: featuresUsed = ['air_pressure', 'air_temp', 'avg_wind_direction', 'avg_wind_speed', 'max_wind_direction',
                      'max_wind_speed', 'relative_humidity']
assembler = VectorAssembler(inputCols=featuresUsed, outputCol="features_unscaled")
assembled = assembler.transform(workingDF)
```

Next, let's use *StandardScaler* to scale the data:

```
In [12]: scaler = StandardScaler(inputCol="features_unscaled", outputCol="features", withStd=True, withMean=True)
scalerModel = scaler.fit(assembled)
scaledData = scalerModel.transform(assembled)
```

The *withMean* argument specifies to center the data with the mean before scaling, and *withStd* specifies to scale the data to the unit standard deviation.

Step 5. **Create elbow plot.** The k-means algorithm requires that the value of  $k$ , the number of clusters, to be specified. To determine a good value for  $k$ , we will use the “elbow” method. This method involves applying k-means, using different values for  $k$ , and calculating the within-cluster sum-of-squared error (WSSE). Since this means applying k-means multiple times, this process can be very compute-intensive. To speed up the process, we will use only a subset of the dataset. We will take every third sample from the dataset to create this subset:

```
In [13]: scaledData = scaledData.select("features", "rowID")
    elbowset = scaledData.filter((scaledData.rowID % 3) == 0).select("features")
    elbowset.persist()
```

The last line calls the `persist()` method to tell Spark to keep the data in memory (if possible), which will speed up the computations.

Let's compute the k-means clusters for  $k = 2$  to 30 to create an elbow plot:

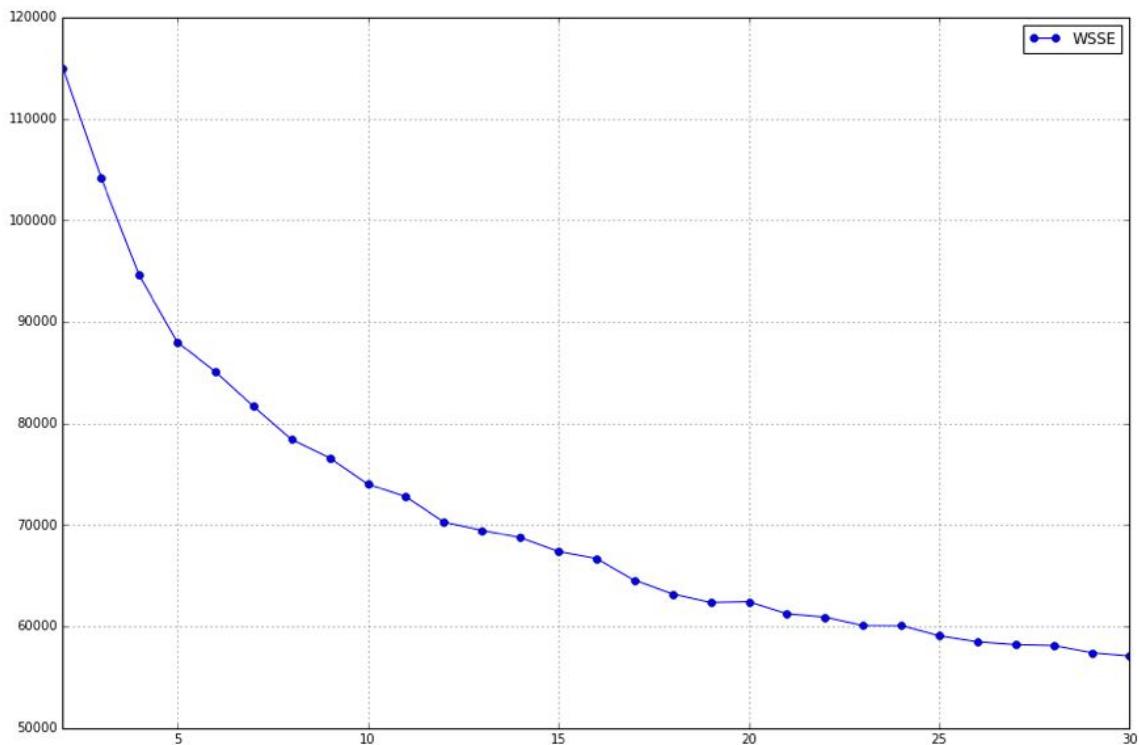
```
In [14]: clusters = range(2,31)
wsseList = utils.elbow(elbowset, clusters)

Training for cluster size 2
.....WSSE = 114993.13181214454
Training for cluster size 3
.....WSSE = 104181.0978581738
Training for cluster size 4
.....WSSE = 94577.27151288436
Training for cluster size 5
.....WSSE = 87993.46098415818
Training for cluster size 6
.....WSSE = 85084.23922296542
Training for cluster size 7
.....WSSE = 81664.96024487517
Training for cluster size 8
```

The first line creates an array with the numbers 2 through 30, and the second line calls the `elbow()` function defined in the `utils.py` library to perform clustering. The first argument to `elbow()` is the dataset, and the second is the array of values for  $k$ . The `elbow()` function returns an array of the WSSE for each number of clusters.

Let's plot the results by calling `elbow_plot()` in the `utils.py` library:

```
In [15]: utils.elbow_plot(wsseList, clusters)
```



The values for  $k$  are plotted against the WSSE values, and the elbow, or bend in the curve, provides a good estimate for the value for  $k$ . In this plot, we see that the elbow in the curve is between 10 and 15, so let's choose  $k = 12$ . We will use this value to set the number of clusters for k-means.

Step 6. **Cluster using selected  $k$ .** Let's select the data we want to cluster:

```
In [16]: scaledDataFeat = scaledData.select("features")
scaledDataFeat.persist()
```

Again, we call the `persist()` method to cache the data in memory for faster access.

We can perform clustering using *KMeans*:

```
In [17]: kmeans = KMeans(k=12, seed=1)
model = kmeans.fit(scaledDataFeat)
transformed = model.transform(scaledDataFeat)
```

The first line creates a new *KMeans* instance with 12 clusters and a specific seed value. (As in previous hands-on activities, we use a specific seed value for reproducible results.) The second line fits the data to the model, and the third applies the model to the data.

Once the model is created, we can determine the center measurement of each cluster:

```
In [18]: centers = model.clusterCenters()
centers

Out[18]: [array([-0.13720796,  0.6061152 ,  0.22970948, -0.62174454,  0.40604553,
   -0.63465994, -0.42215364]), array([ 1.42238994, -0.10953198, -1.10891543, -0.07335197, -0.96904335,
   -0.05226062, -0.99615617]), array([-0.63637648,  0.01435705, -1.1038928 , -0.58676582, -0.96998758,
   -0.61362174,  0.33603011]), array([-0.22385278, -1.06643622,  0.5104215 , -0.24620591,  0.68999967,
   -0.24399706,  1.26206479]), array([ 1.17896517, -0.25134204, -1.15089838,  2.11902126, -1.04950228,
   2.23439263, -1.12861666]), array([-1.14087425, -0.979473 ,  0.42483303,  1.68904662,  0.52550171,
   1.65795704,  1.03863542]), array([ 0.50746307, -1.08840683, -1.20882766, -0.57604727, -1.0367013 ,
   -0.58206904,  0.97099067]), array([ 0.14064028,  0.83834618,  1.89291279, -0.62970435, -1.54598923,
   -0.55625032, -0.75082891]), array([-0.0339489 ,  0.98719067, -1.33032244, -0.57824562, -1.18095582,
   -0.58893358, -0.81187427]), array([-0.22747944,  0.59239924,  0.40531475,  0.6721331 ,  0.51459992,
   0.61355559, -0.15474261]), array([ 0.3334222 , -0.99822761,  1.8584392 , -0.68367089, -1.53246714,
   -0.59099434,  0.91004892]), array([ 0.3051367 ,  0.67973831,  1.36434828, -0.63793718,  1.631528 ,
   -0.58807924, -0.67531539])]
```

It is difficult to compare the cluster centers by just looking at these numbers. So we will use plots in the next step to visualize them.

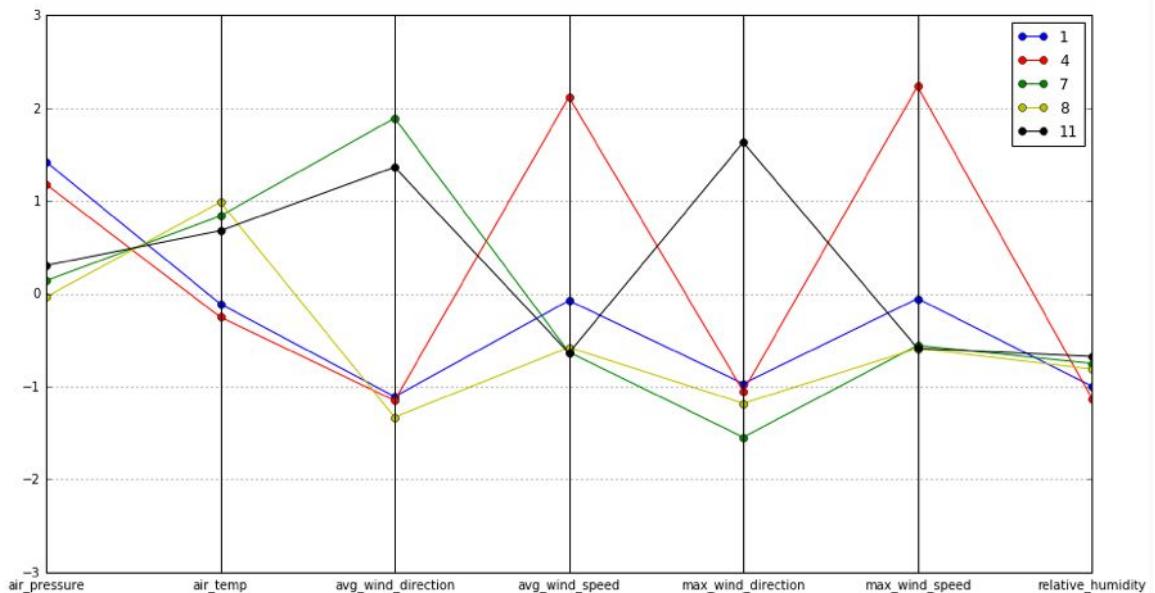
**Step 7. Create parallel plots of clusters and analysis.** A parallel coordinates plot is a great way to visualize multi-dimensional data. Each line plots the centroid of a cluster, and all of the features are plotted together. Recall that the feature values were scaled to have mean = 0 and standard deviation = 1. So the values on the y-axis of these parallel coordinates plots show the number of standard deviations from the mean. For example, +1 means one standard deviation higher than the mean of all samples, and -1 means one standard deviation lower than the mean of all samples.

We'll create the plots with *matplotlib* using a Pandas DataFrame each row contains the cluster center coordinates and cluster label. (Matplotlib can plot Pandas DataFrames, but not Spark DataFrames.) Let's use the *pd\_centers()* function in the *utils.py* library to create the Pandas DataFrame:

```
In [19]: P = utils.pd_centers(featuresUsed, centers)
```

Let's show clusters for "Dry Days", i.e., weather samples with low relative humidity:

```
In [20]: utils.parallel_plot(P[P['relative_humidity'] < -0.5], P)
```

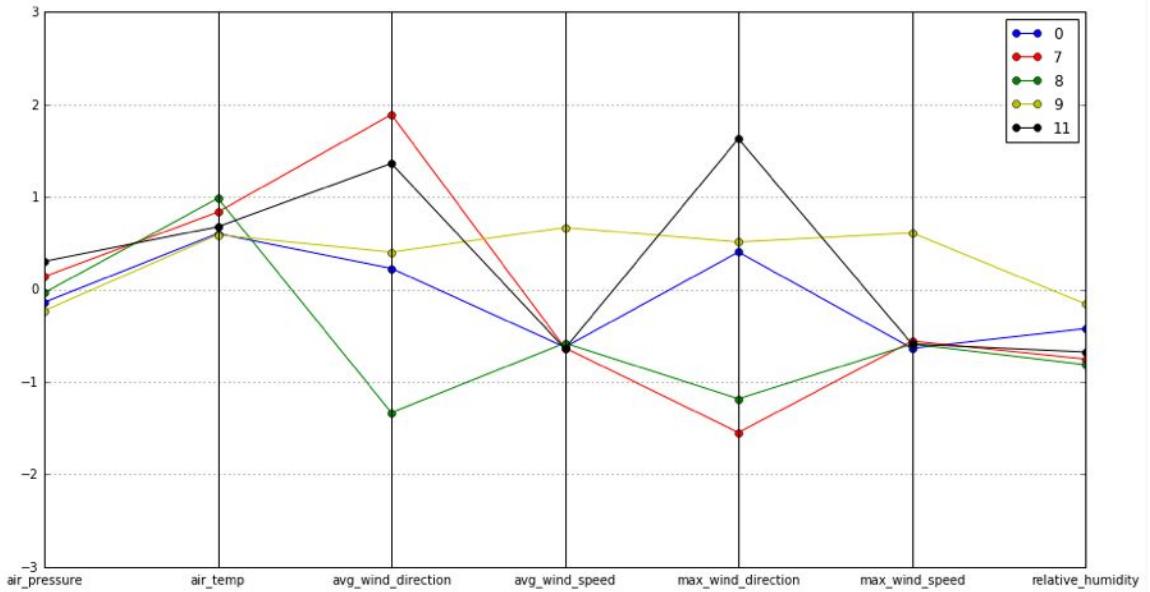


The first argument to *parallel\_plot* selects the clusters whose relative humidities are centered less than 0.5 from the mean value. All clusters in this plot have *relative\_humidity* < -0.5, but they differ in values for other features, meaning that there are several weather patterns that include low humidity.

Note in particular cluster 4. This cluster has samples with lower-than-average wind direction values. Recall that wind direction values are in degrees, and 0 means wind coming from the North and increasing clockwise. So samples in this cluster have wind coming from the N and NE directions, with very high wind speeds, and low relative humidity. These are characteristic weather patterns for Santa Ana conditions, which greatly increase the dangers of wildfires.

Let's show clusters for "Warm Days", i.e., weather samples with high air temperature:

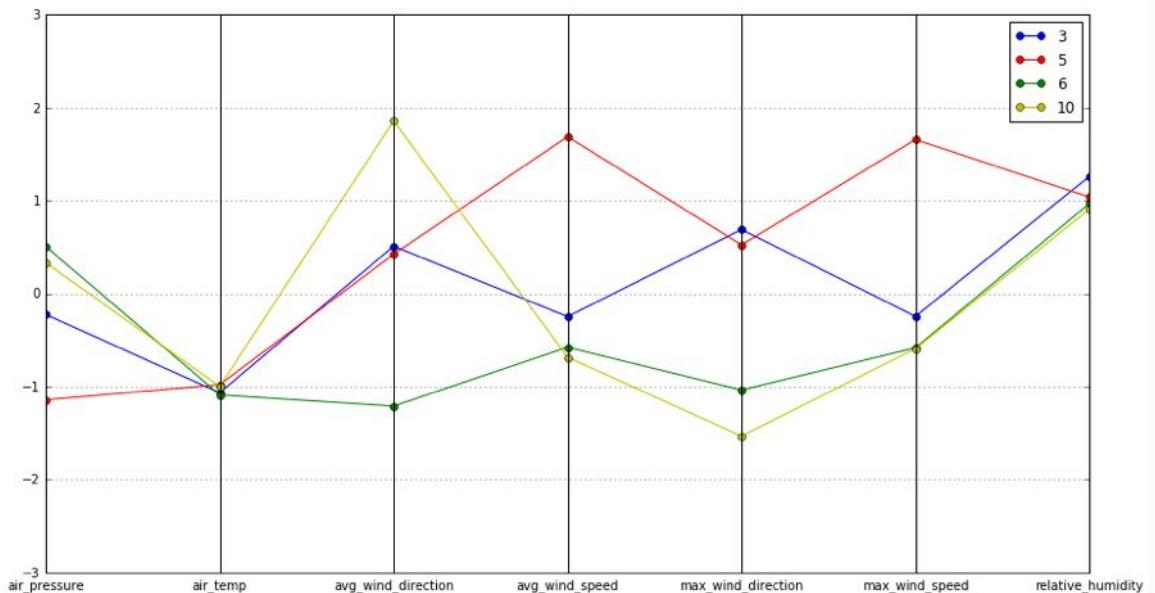
```
In [21]: utils.parallel_plot(P[P['air_temp'] > 0.5], P)
```



All clusters in this plot have `air_temp` > 0.5, but they differ in values for other features.

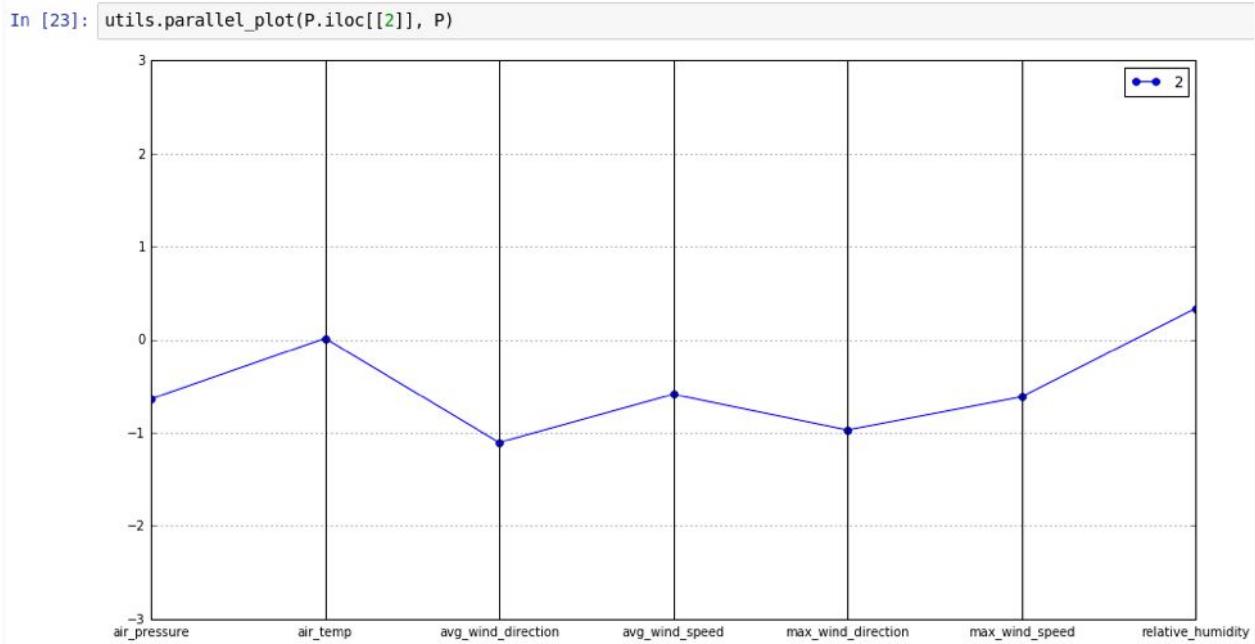
Let's show clusters for "Cool Days", i.e., weather samples with high relative humidity and low air temperature:

```
In [22]: utils.parallel_plot(P[(P['relative_humidity'] > 0.5) & (P['air_temp'] < 0.5)], P)
```



All clusters in this plot have *relative\_humidity* > 0.5 and *air\_temp* < 0.5. These clusters represent cool temperature with high humidity and possibly rainy weather patterns. For cluster 5, note that the wind speed values are high, suggesting stormy weather patterns with rain and wind.

So far, we've seen all the clusters except 2 since it did not fall into any of the other categories. Let's plot this cluster:



Cluster 2 captures days with mild weather.