# Intro

* 6 parts of the books
  + Introduction. Welcomes you to the book and clearly lays out what to expect and your learning outcomes (you are here).
  + Data Preparation. Tutorials for loading and preparing data, evaluating model predictions, estimating model skill and developing a baseline for model performance.
  + Linear Algorithms. Tutorials on linear machine learning algorithms such as linear regression, multivariate linear regression, logistic regression and the Perceptron algorithm.
  + Nonlinear Algorithms. Tutorials on nonlinear machine learning algorithms such as Naive Bayes, k-Nearest Neighbors, Learning Vector Quantization, Back-propagation and Decision Trees.
  + Ensemble Algorithms. Tutorials on ensemble machine learning algorithms such as Bootstrap Aggregation, Random Forest and Stacked Generalization.
  + Conclusions. A review of how far you have come and resources for getting help and further reading
* Tutorial structure
  + 1. Overview summarizes the tutorial
  + 2. Description: describes both the technique and problem that you will be applying it to
  + 3. Tutorial: from an idea to a fully working implementation
  + 4. Case Study: working case study on a real-world predictive modeling problem
  + 5. Extensions: ideas to extend the example
  + 6. Review

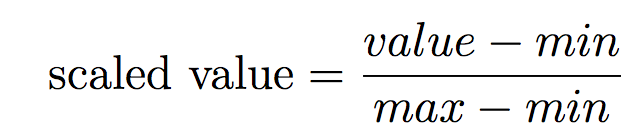
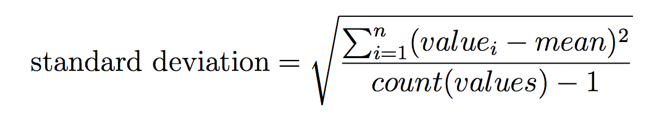
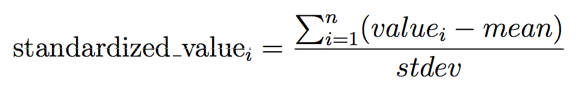
# Data preparation

* Just read through this part

## 1. Load Data from CSV

* 1. Description
     1. Comma Separated Values (CSV)
     2. Prima Indians Diabetes dataset (768 observations with 8 input variables, baseline 65%, top 77%)
     3. Iris Flower Species dataset (baseline 26%, 150 observations with 4 input variables)
  2. Tutorial
     1. Load CSV file
     2. Convert String to Floats
     3. Convert String to Integers
  3. Extensions
* NumPy and Pandas
* NumPy offers the loadtxt() function for loading data files as NumPy arrays.
* Pandas offers the read csv()2 function that offers a lot of flexibility regarding data types, file headers and more.

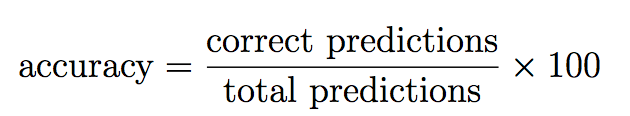
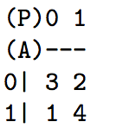
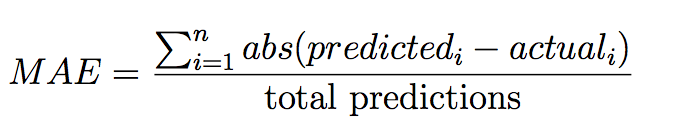
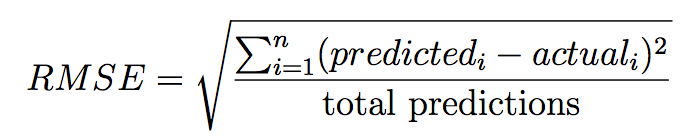
## Scale ML data

* Idea: when to normalize / standardize your data
  1. Description
     1. Prima Indians Diabetes dataset
  2. Tutorial
     1. Normalize Data:
  + can refer to di↵erent techniques depending on context.
  + Here, we use normalization to refer to rescaling an input variable to the range between 0 and 1
  + Step:
    - Find min, max for each columns
    - 
    1. Standardize Data
  + Standardization is a rescaling technique that refers to centering the distribution of the data on the value 0 and the standard deviation to the value 1.
  + Together, the mean and the standard deviation can be used to summarize a normal distribution, also called the Gaussian distribution or bell curve.
  + Step
    - Calculate mean, standard deviation. 
    - standardize the values. 
    1. When to Normalize and Standardize
  + Standardization is a scaling technique that assumes your data conforms to a normal distribution.
    - If a given data attribute is normal or close to normal, this is probably the scaling method to use.
    - The data calculated (mean, standard deviation) is useful for future use.
  + Normalization is a scaling technique that does not assume any specific distribution.
    - If your data is not normally distributed, consider normalizing it prior to applying your machine learning algorithm.
  1. Extensions
  + Normalization that permits a configurable range, such as -1 to 1 and more.
  + Standardization that permits a configurable spread, such as 1, 2 or more standard deviations from the mean.
  + Exponential transforms such as logarithm, square root and exponents.
  + Power transforms such as Box-Cox for fixing the skew in normally distributed data.

## Algorithm evaluation methods

* 1. Description
* How to implement a train and test split of your data.
* How to implement a k-fold cross-validation split of your data.
  1. Tutorial
     1. Train and test split
* Pro: easiest resampling method
* Con: noisy estimate of algorithm performance
  + 1. k-fold Cross-Validation Split.
* Resampling method that provides a more accurate estimate of algorithm performance.
* First splitting the data into k groups.
* The algorithm is then trained and evaluated k times and the performance summarized by taking the mean performance score
  + 1. How to Choose a Resampling Method.
* K-fold
  + Pro:
    - gold standard for estimating the performance of machine learning algorithms on new data is k-fold cross-validation
    - When well-configured, k-fold cross-validation gives a robust estimate of performance compared to other methods
  + Cons: time-consuming to run, requiring k different models to be trained and evaluated.
    - This is a problem if you have a very large dataset or if you are evaluating a model that takes a long time to train.
* Train and test split
  + Pro:
    - most widely used. This is because it is easy to understand and implement,
    - and because it gives a quick estimate of algorithm performance. Only a single model is constructed and evaluated
  + cons: Although the train and test split method can give a noisy or unreliable estimate of the performance of a model on new data, this becomes less of a problem if you have a very large dataset.
* Large datasets are those in the hundreds of thousands or millions of records, large enough that splitting it in half results in two datasets that have nearly equivalent statistical properties. In such cases, there may be little need to use k-fold cross-validation as an evaluation of the algorithm and a train and test split may be just as reliable.
  1. Extension
* Other methods:
  + Repeated Train and Test. This is where the train and test split is used, but the process is repeated many times.
  + LOOCV or Leave One Out Cross-Validation. This is a form of k-fold cross-validation where the value of k is fixed at 1.
  + Stratification. In classification problems, this is where the balance of class values in each group is forced to match the original dataset.
  1. Review

## Evaluation metrics

* 1. Description
* How to implement classification accuracy.
* How to implement and interpret a confusion matrix.
* How to implement mean absolute error for regression.
* How to implement root mean squared error for regression.
  1. Tutorial
     1. Classification Accuracy.
* 
  + 1. Confusion Matrix.
* summary of all of the predictions made compared to the expected actual values
* 
  + 1. Mean Absolute Error.
* 
  + 1. Root Mean Squared Error.
* 
  1. Extension
* Other performance matrix:
  + Precision for classification.
  + Recall for classification.
  + F1 for classification.
  + Area Under ROC Curve or AUC for classification.
  + Goodness of Fit or R2 (R squared) for regression.
  1. Review

## Baseline models

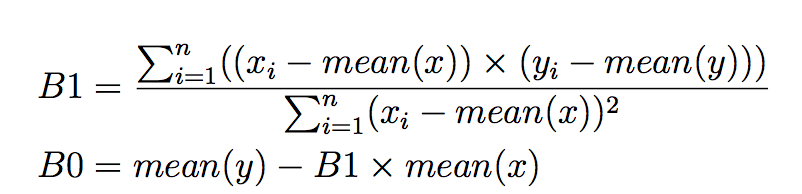
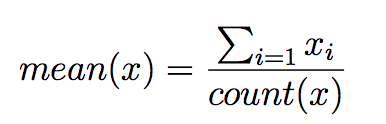
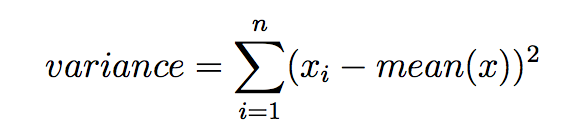
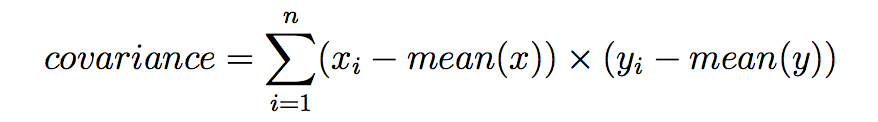
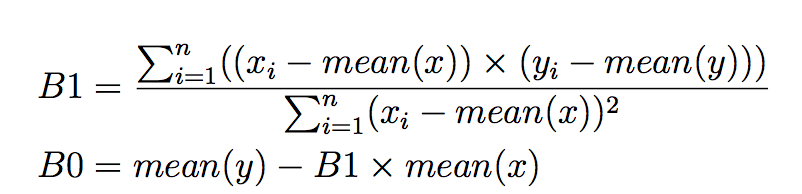
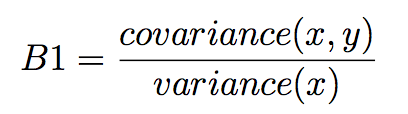
* 1. Description
* A baseline provides a point of comparison for the more advanced methods that you evaluate later
* wo most commonly used baseline algorithms are:
  + How to implement the random prediction algorithm.
  + How to implement the zero rule prediction algorithm.
  1. Tutorial
     1. Random Prediction Algorithm.
* predicts a random outcome as observed in the training data.
* It is perhaps the simplest algorithm to implement
  + 1. Zero Rule Algorithm.
* The Zero Rule Algorithm is a better baseline than the random algorithm. It uses more information about a given problem to create one rule in order to make predictions
* For classification problems, the one rule is to predict the class value that is most common in the training dataset.
* Regression problems require the prediction of a real value. A good default prediction for real values is to predict the central tendency.
  1. Extension
* Alternate Central Tendency where the median, mode or other central tendency calculations are predicted instead of the mean.
* Moving Average for time series problems where the mean of the last n records is predicted.
  1. Review

# II. Linear algorithms

## Algorithm Test Harnesses

* 1. Description
* How to implement a train-test algorithm test harness.
* How to implement a k-fold cross-validation algorithm test harness.
* test harness provides a consistent way to evaluate machine learning algorithms on a dataset.
* It involves 3 elements:
  + 1. The resampling method to split-up the dataset.
  + 2. The machine learning algorithm to evaluate.
  + 3. The performance measure by which to evaluate predictions.
* Pima Indians Diabetes Dataset
* The loading and preparation of a dataset is a prerequisite step that must have been completed prior to using the test harness.
* The test harness must allow for different machine learning algorithms to be evaluated, whilst the dataset, resampling method and performance measures are kept constant
  1. Tutorial
     1. Train-Test Algorithm Test Harness.
* We can assume the prior development of a function to split a dataset into train and test sets and a function to evaluate the accuracy of a set of predictions.
  + 1. Cross-Validation Algorithm Test Harness.
* algorithm must be evaluated on di↵erent subsets of the dataset many times => we need additional loops within our evaluate\_algorithm() function
* Unlike the train-test algorithm test harness, a list of scores is returned, one for each cross-validation fold.
  1. Extension
* Parameterized Evaluation. Pass in the function used to evaluate predictions, allowing you to seamlessly work with regression problems.
* Parameterized Resampling. Pass in the function used to calculate resampling splits, allowing you to easily switch between the train-test and cross-validation methods.
* Standard Deviation Scores. Calculate the standard deviation to get an idea of the spread of scores when evaluating algorithms using cross-validation.
  1. Review

## Simple Linear regression

* 1. Description
     1. Simple Linear Regression
* y = b0 + b1 \* x
* calculate statistical properties from the data such as mean, variance and covariance.
* 
  + 1. Swedish Auto Insurance Dataset
* Need to convert the European comma (,) to the decimal dot (.)
* You will also need change the file from white-space-separated variables to CSV format.
  + 1. Idea:
* How to estimate statistical quantities from training data.
* How to estimate linear regression coefficients from data.
* How to make predictions using linear regression for new data.
  1. Tutorial
     1. Calculate Mean and Variance.
* 
* 
  + 1. Calculate Covariance.
* The covariance of two groups of numbers describes how those numbers change together.
* Correlation describes the relationship between two groups of numbers, whereas covariance can describe the relationship between two or more groups of numbers.
* Additionally, covariance can be normalized to produce a correlation value.
* 
  + 1. Estimate Coefficients.
* 
* 
  + 1. Make Predictions.
    2. Swedish Auto Insurance Case Study.
  1. Extension
  2. Review

## Multivariate Linear regression

* 1. Description
  2. Tutorial
  3. Extension
  4. Review

## Logistic regression

* 1. Description
  2. Tutorial
  3. Extension
  4. Review

## Perception

* 1. Description
  2. Tutorial
  3. Extension
  4. Review

# III. Nonlinear algorithms

# IV. Ensemble algorithms

# V. Conclusion

# Code

## 1. Load file + data process

* Ex: 1a, 1b

"""  
-----------------------------------------------------------------  
Groups load\_file  
Purpose: extract dataset from file  
Functions: load\_csv(filename) => list dataset  
 str\_column\_to\_float(dataset, column) => list dataset   
 str\_column\_to\_int(dataset, column) => list dataset, list lookup   
-----------------------------------------------------------------  
"""

# Load a CSV file  
**def load\_csv2**(filename):  
 dataset = list()  
 **with** open(filename, 'r') **as** file:  
 csv\_reader = reader(file)  
 **for** row **in** csv\_reader:  
 **if not** row:  
 **continue** dataset.append(row)  
 **return** dataset

# Convert string column to float  
**def str\_column\_to\_float**(dataset, column):  
 **for** row **in** dataset:  
 row[column] = float(row[column].strip())  
  
# Convert string column to integer  
**def str\_column\_to\_int**(dataset, column):  
 class\_values = [row[column] **for** row **in** dataset]  
 unique = set(class\_values)  
 lookup = dict()  
 **for** i, value **in** enumerate(unique):  
 lookup[value] = i  
 **for** row **in** dataset:  
 row[column] = lookup[row[column]]  
 **return** lookup

### Normalize data

* Ex: 2b

"""  
-----------------------------------------------------------------  
Groups: normalize  
Purpose: rescaling an input variable to the range between 0 and 1  
Functions: normalize\_dataset(dataset, minmax) => dataset   
 dataset\_minmax(dataset) => list minmax (used in normalize\_dataset)   
-----------------------------------------------------------------  
"""  
  
#Find the min and max values for each column  
**def dataset\_minmax**(dataset):  
 minmax = list()  
 **for** i **in** range(len(dataset[0])):  
 col\_values = [row[i] **for** row **in** dataset]  
 value\_min = min(col\_values)  
 value\_max = max(col\_values)  
 minmax.append([value\_min, value\_max])  
 **return** minmax  
  
# Rescale dataset columns to the range 0-1  
**def normalize\_dataset**(dataset, minmax):  
 **for** row **in** dataset:  
 **for** i **in** range(len(row)):  
 row[i] = (row[i] - minmax[i][0]) / (minmax[i][1] - minmax[i][0])

### standardize data

* Example: 2c

"""  
-----------------------------------------------------------------  
Groups: standardize\_data  
Purpose: centering the distribution of the data on the value 0 and the standard deviation to the value 1  
Functions: standardize\_dataset(dataset, means, stdevs) => list dataset  
 column\_means(dataset) => list means  
 column\_stdevs(dataset, means) => list stdevs  
-----------------------------------------------------------------  
"""  
  
# calculate column means  
**def column\_means**(dataset):  
 means = [0 **for** i **in** range(len(dataset[0]))]  
 **for** i **in** range(len(dataset[0])):  
 col\_values = [row[i] **for** row **in** dataset]  
 means[i] = sum(col\_values) / float(len(dataset))  
 **return** means  
  
# calculate column standard deviations  
**def column\_stdevs**(dataset, means):  
 stdevs = [0 **for** i **in** range(len(dataset[0]))]  
 **for** i **in** range(len(dataset[0])):  
 variance = [pow(row[i]-means[i], 2) **for** row **in** dataset]  
 stdevs[i] = sum(variance)  
 stdevs = [sqrt(x/(float(len(dataset)-1))) **for** x **in** stdevs]  
 **return** stdevs  
  
# standardize dataset  
**def standardize\_dataset**(dataset, means, stdevs):  
 **for** row **in** dataset:  
 **for** i **in** range(len(row)):  
 row[i] = (row[i] - means[i]) / stdevs[i]

### Split datasets for training and testing purpose

#### Method 1: train test

"""  
-----------------------------------------------------------------  
Function Name: train\_test\_split  
Parameters: dataset - list data type  
 split - ratio  
Return Value: train dataset  
 test dataset  
Purpose: split dataset into a train and test set  
-----------------------------------------------------------------  
"""

Evaluation- Method 1: train test split

"""  
-----------------------------------------------------------------  
Function Name: evaluate\_algorithm  
Parameters: dataset - no need train test dataset, function will split  
 split - split ratio  
 algorithm - prediction model  
 \*args - more options  
Return Value: accuracy percentage  
Purpose: Evaluate how effective an algorithm is using a train/test split  
-----------------------------------------------------------------  
"""

#### Method 2: k-fold validation

"""  
-----------------------------------------------------------------  
Function Name: cross\_validation\_split  
Parameters: dataset - list data type  
 n\_folds - number of folds we want to use  
Return Value: dataset\_split  
Purpose: k-fold Cross-Validation Split  
-----------------------------------------------------------------  
"""

Evaluation- Method 2: Cross validation

"""  
-----------------------------------------------------------------  
Function Name: cross\_validation\_split  
Parameters: dataset,   
 folds  
Return Value: dataset\_split  
Purpose: k-fold Cross-Validation Split  
-----------------------------------------------------------------  
"""  
# Split a dataset into k folds  
**def cross\_validation\_split**(dataset, folds=3):  
 dataset\_split = list()  
 dataset\_copy = list(dataset)  
 fold\_size = int(len(dataset) / folds)  
 **for** i **in** range(folds):  
 fold = list()  
 **while** len(fold) < fold\_size:  
 index = randrange(len(dataset\_copy))  
 fold.append(dataset\_copy.pop(index))  
 dataset\_split.append(fold)  
 **return** dataset\_split

**from** random **import** seed  
**from** random **import** randrange

## 3. Algorithms

### 3a. Baseline algorithms

* Ex: 5a, 5b

For classification

"""  
-----------------------------------------------------------------  
Group: baseline algorithms  
Parameters: dataset train  
 dataset test  
Return Value: predicted list  
Purpose: make prediction using basic functions (there is no ML here)  
-----------------------------------------------------------------  
"""  
# Generate random predictions  
**def random\_algorithm**(train, test):  
 output\_values = [row[-1] **for** row **in** train]  
 unique = list(set(output\_values))  
 predicted = list()  
 **for** row **in** test:  
 index = randrange(len(unique))  
 predicted.append(unique[index])  
 **return** predicted

# zero rule algorithm for classification  
**def zero\_rule\_algorithm\_classification**(train, test):  
 output\_values = [row[-1] **for** row **in** train]  
 prediction = max(set(output\_values), key=output\_values.count)  
 predicted = [prediction **for** i **in** range(len(test))]  
 **return** predicted

For regression

""  
-----------------------------------------------------------------  
Group: baseline algorithms  
Parameters: dataset train  
 dataset test  
Return Value: predicted list  
Purpose: make prediction using basic functions (there is no ML here)  
-----------------------------------------------------------------  
"""

# zero rule algorithm for regression  
**def zero\_rule\_algorithm\_regression**(train, test):  
 output\_values = [row[-1] **for** row **in** train]  
 prediction = sum(output\_values) / float(len(output\_values))  
 predicted = [prediction **for** i **in** range(len(test))]  
 **return** predicted

## 4. Evaluation

### 4a. Classification metric

* Ex: 4a, 4b

"""  
-----------------------------------------------------------------  
Groups: classification\_metric  
Purpose: evaluate for classifications  
Functions: accuracy\_metric(actual, predicted) => float %  
 confusion\_matrix(actual, predicted) => set unique, 2d-list matrix  
 print\_confusion\_matrix(unique, matrix) (used in confusion\_matrix)  
-----------------------------------------------------------------  
"""  
# Calculate accuracy percentage between two lists  
**def accuracy\_metric**(actual, predicted):  
 correct = 0  
 **for** i **in** range(len(actual)):  
 **if** actual[i] == predicted[i]:  
 correct += 1  
 **return** correct / float(len(actual)) \* 100.0

# calculate a confusion matrix  
**def confusion\_matrix**(actual, predicted):  
 unique = set(actual)  
 matrix = [list() **for** x **in** range(len(unique))]  
 **for** i **in** range(len(unique)):  
 matrix[i] = [0 **for** x **in** range(len(unique))]  
 lookup = dict()  
 **for** i, value **in** enumerate(unique):  
 lookup[value] = i  
 **for** i **in** range(len(actual)):  
 x = lookup[actual[i]]  
 y = lookup[predicted[i]]  
 matrix[x][y] += 1  
 **return** unique, matrix  
  
# pretty print a confusion matrix  
**def print\_confusion\_matrix**(unique, matrix):  
 print( '(P)' + ' '.join(str(x) **for** x **in** unique))  
 print('(A) ---')  
 **for** i, x **in** enumerate(unique):  
 print("%s| %s" % (x,' '.join(str(x) **for** x **in** matrix[i])))

### 4b. Regression metric

* Example: 4c, 4d

"""  
-----------------------------------------------------------------  
Groups: regression\_metric  
Purpose: evaluate for regression  
Functions: mae\_metric(actual, predicted) => float  
 rmse\_metric(actual, predicted) => float  
-----------------------------------------------------------------  
"""  
# Calculate mean absolute error  
**def mae\_metric**(actual, predicted):  
 sum\_error = 0.0  
 **for** i **in** range(len(actual)):  
 sum\_error += abs(predicted[i] - actual[i])  
 **return** sum\_error / float(len(actual))

# Calculate root mean squared error  
**def rmse\_metric**(actual, predicted):  
 sum\_error = 0.0  
 **for** i **in** range(len(actual)):  
 prediction\_error = predicted[i] - actual[i]  
 sum\_error += (prediction\_error \*\* 2)  
 mean\_error = sum\_error / float(len(actual))  
 **return** sqrt(mean\_error)