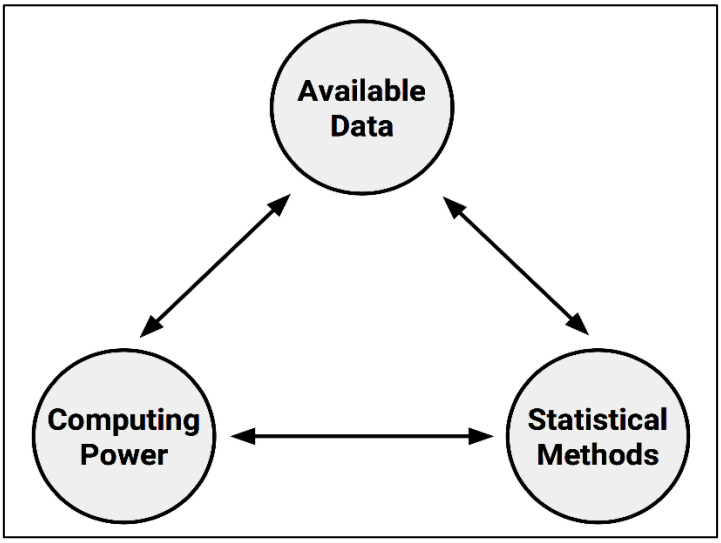
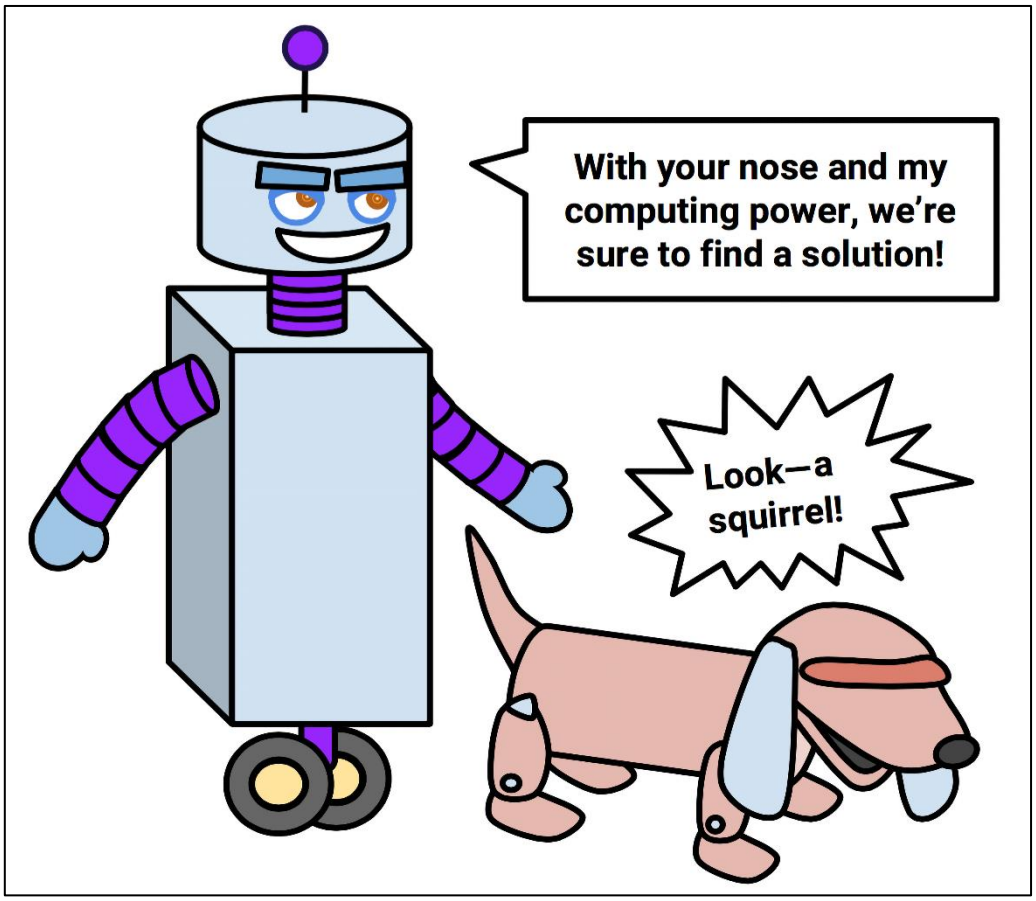
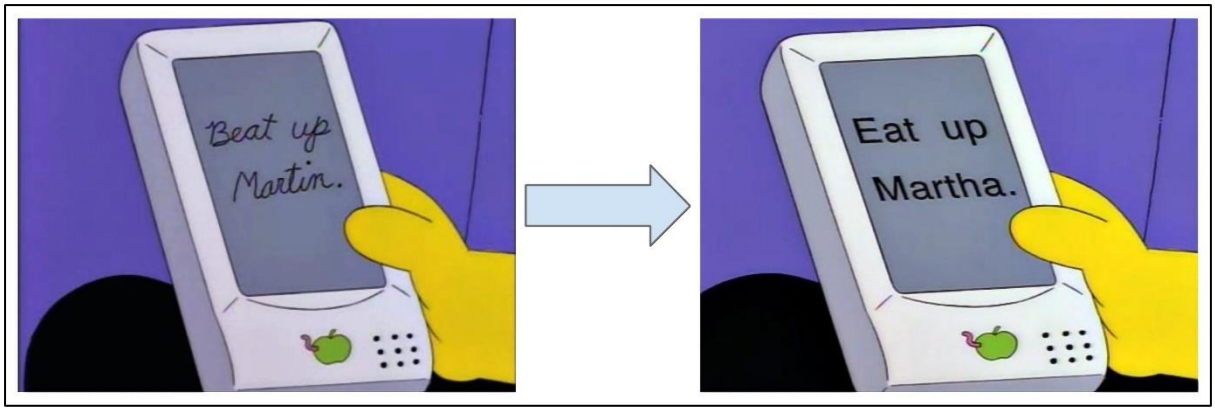
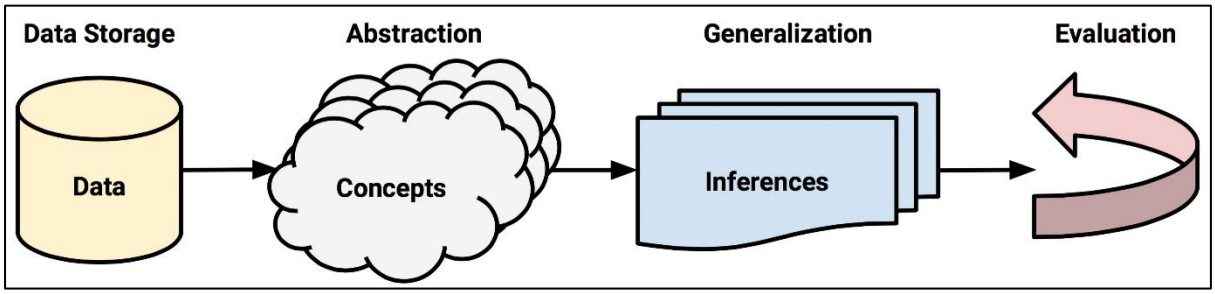
Chapter 1: Introducing Machine Learning






Beat up Martin.
Eat up Martha.


Data Storage
Abstraction
Generalization
Evaluation
Data
Concepts
Inferences


Ceci n'est pas une pipe.



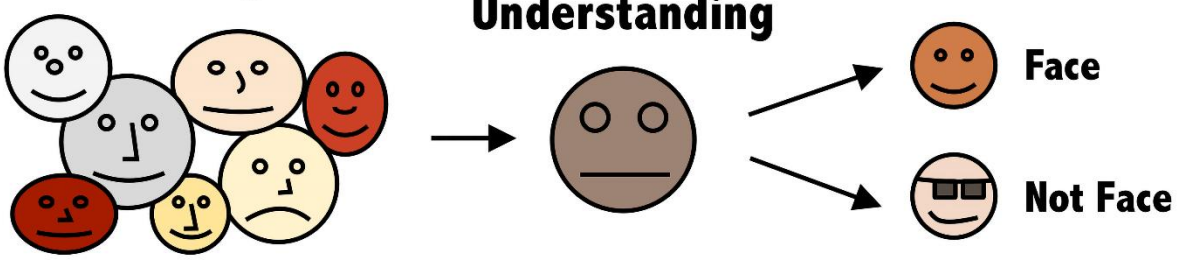
Observations → **Data** → **Model**

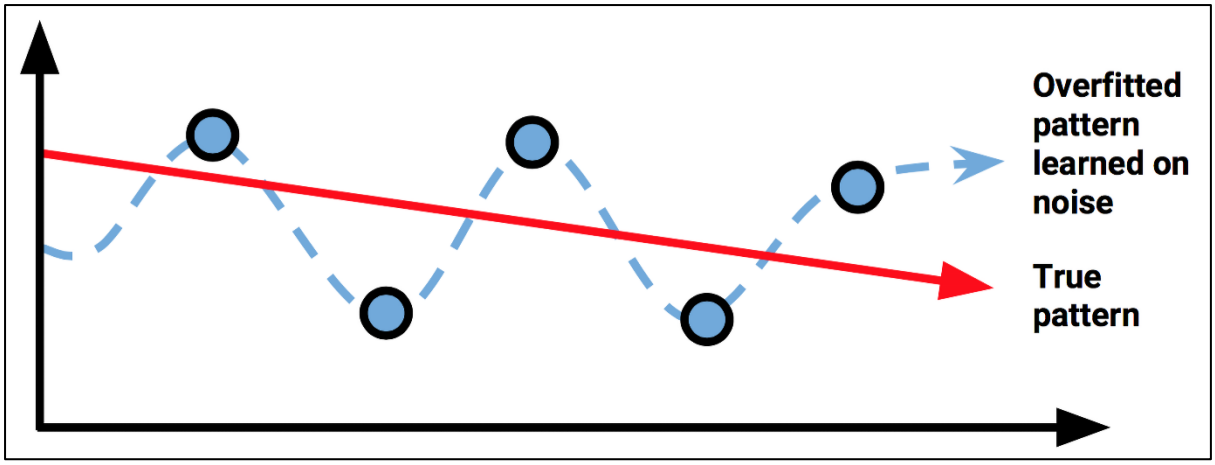


Distance	Time
4.9m	1s
19.6m	2s
44.1m	3s
78.5m	4s

$$g = 9.8m/s^2$$

Reality → **Machine's Understanding** → **Result**

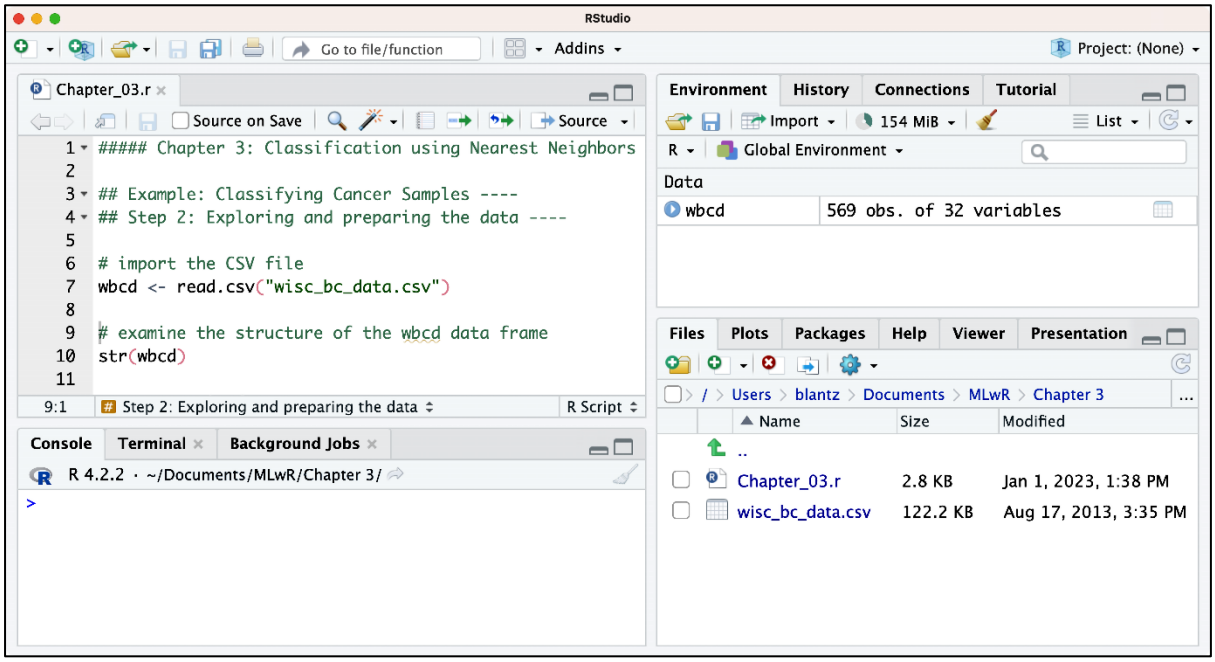




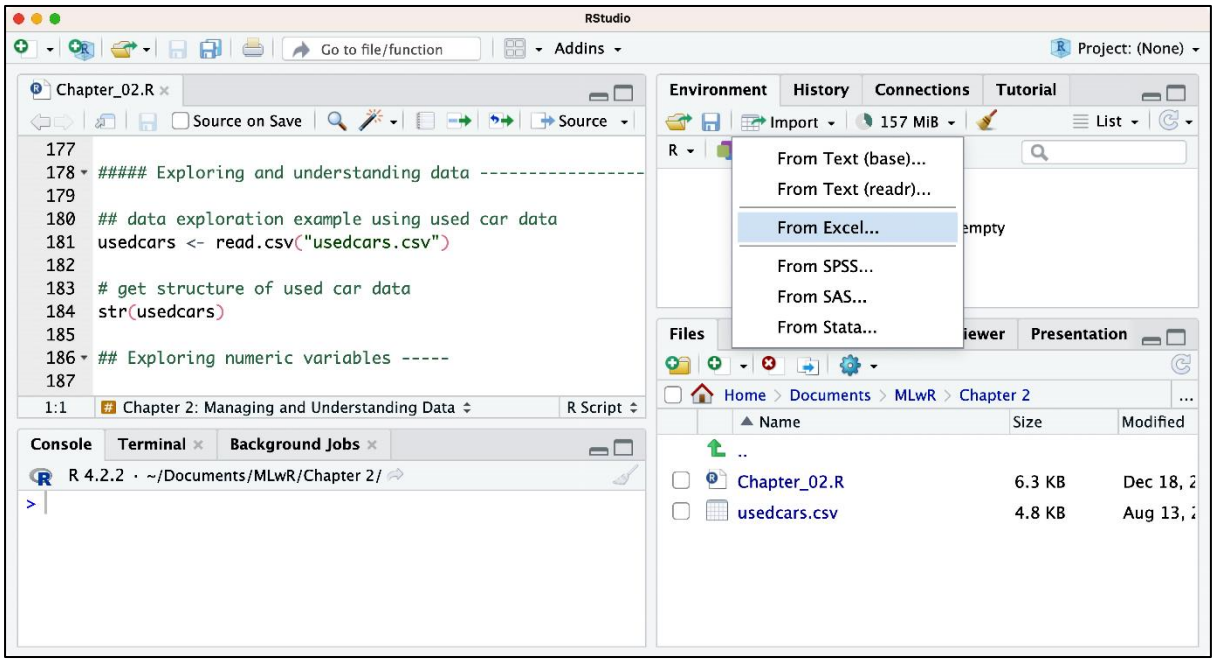
features

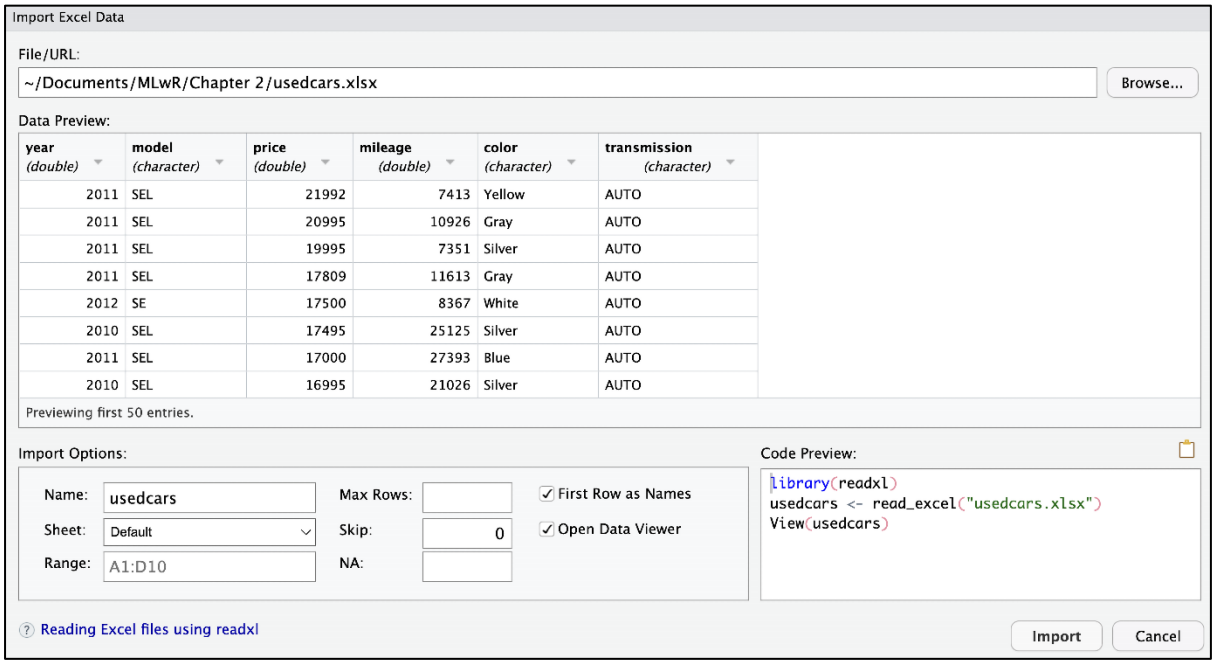
year	model	price	mileage	color	transmission
2011	SEL	21992	7413	Yellow	AUTO
2011	SEL	20995	10926	Gray	AUTO
2011	SEL	19995	7351	Silver	AUTO
2011	SEL	17809	11613	Gray	AUTO
2012	SE	17500	8367	White	MANUAL
2010	SEL	17495	25125	Silver	AUTO
2011	SEL	17000	27393	Blue	AUTO
2010	SEL	16995	21026	Silver	AUTO
2011	SES	16995	32655	Silver	AUTO

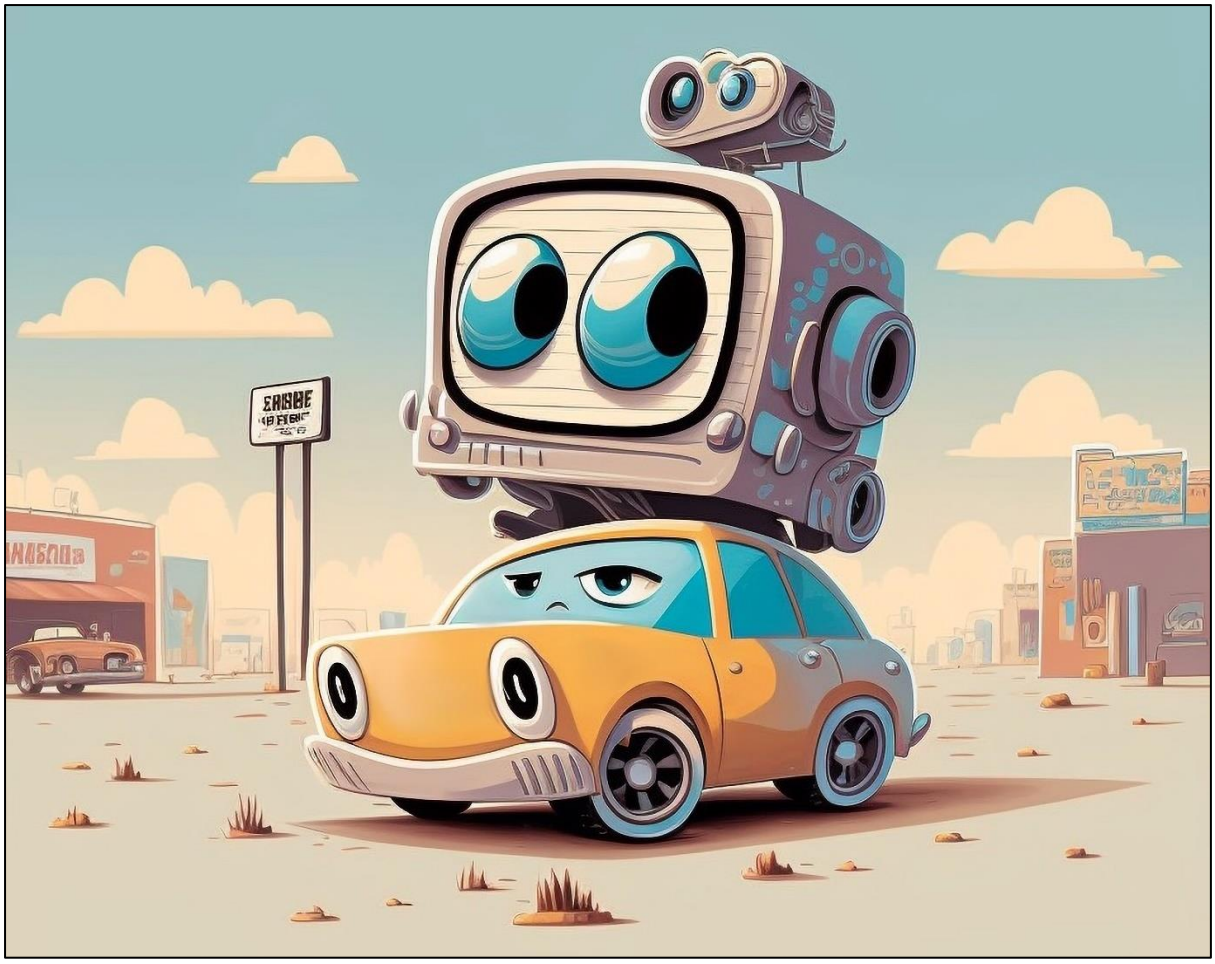
examples

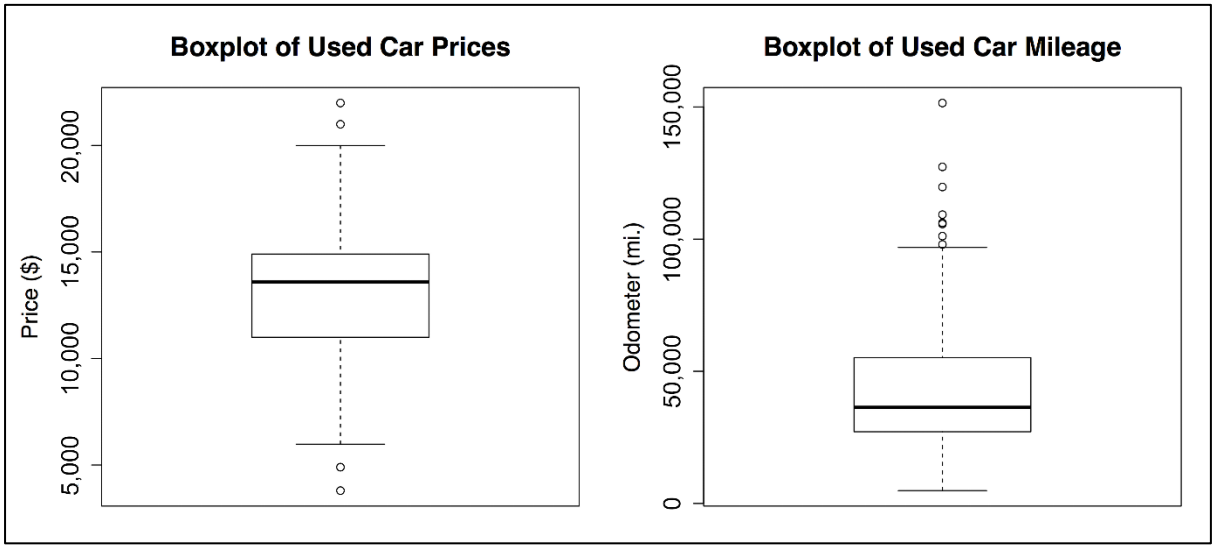


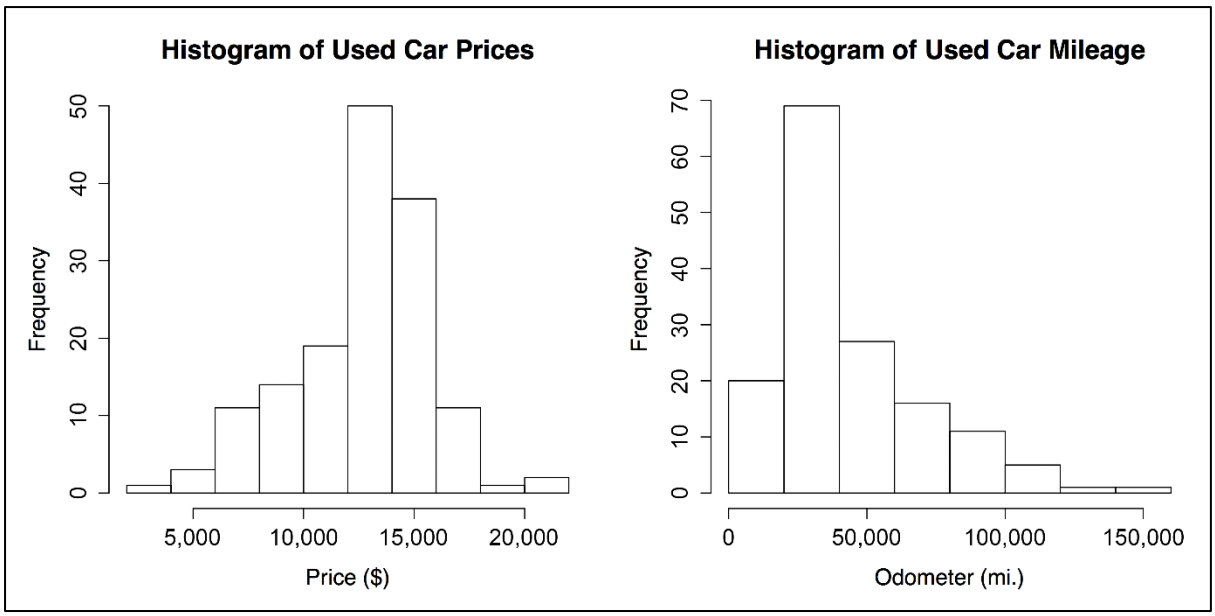
Chapter 2: Managing and Understanding Data

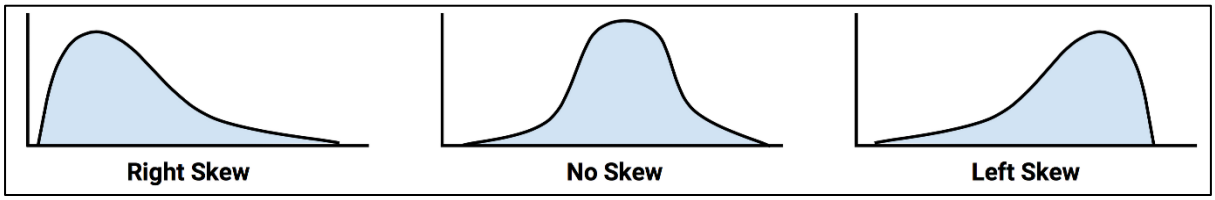


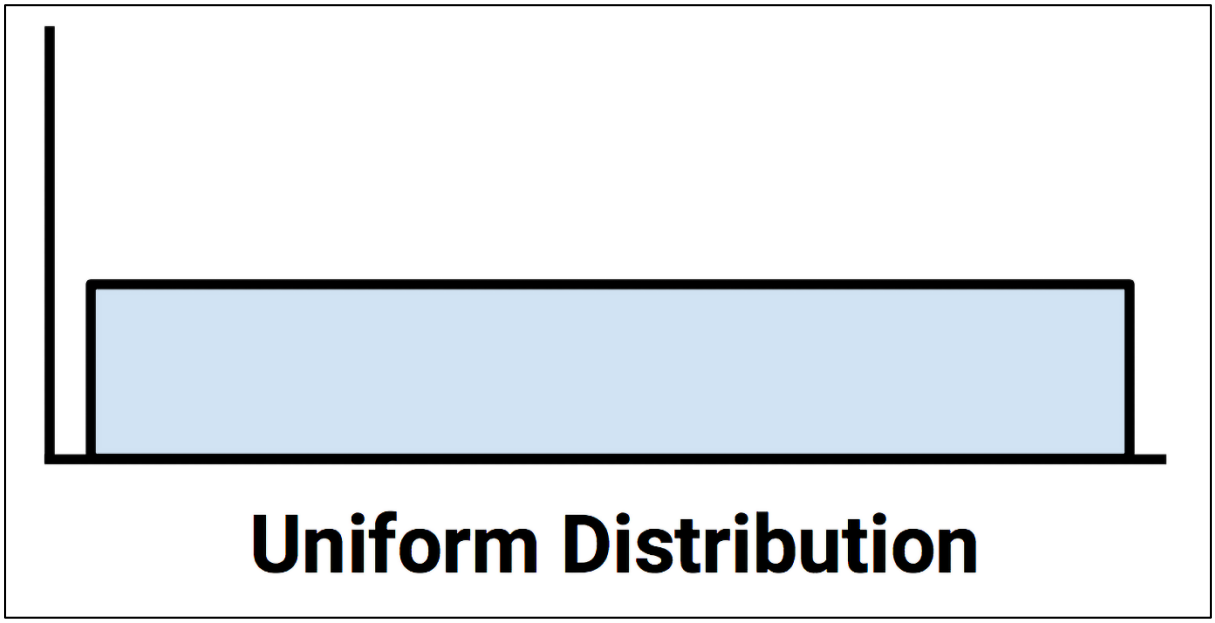


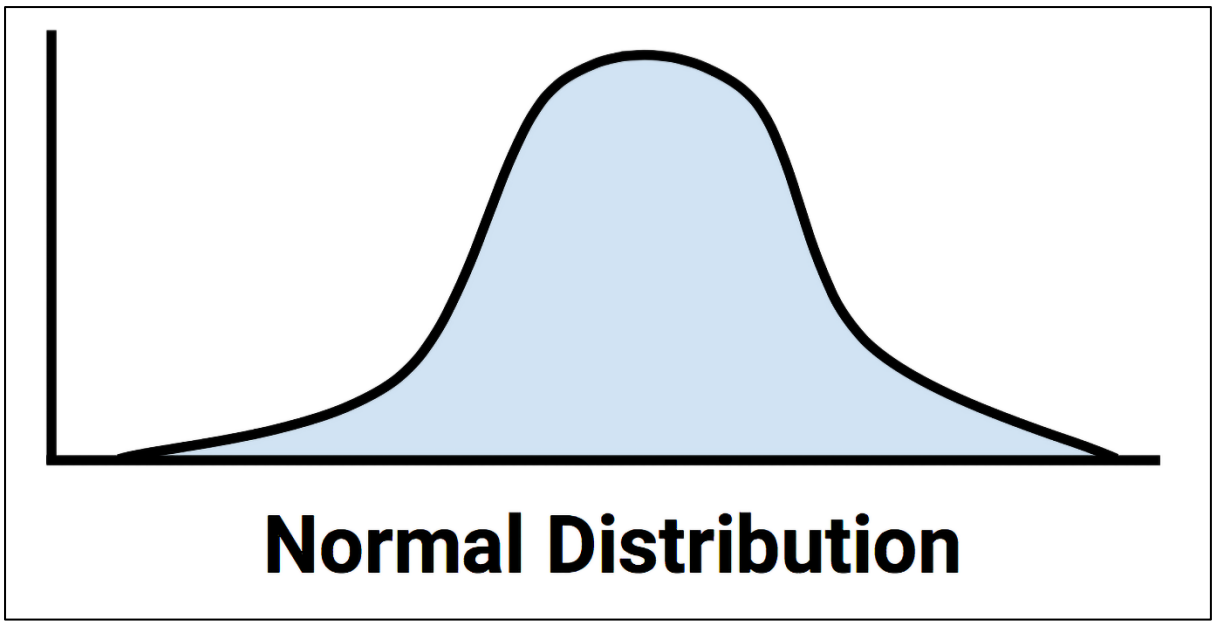



Boxplot of Used Car Prices
Price ($)
5,000
10,000
15,000
20,000
Boxplot of Used Car Mileage
Odometer (mi.)
0
50,000
100,000
150,000

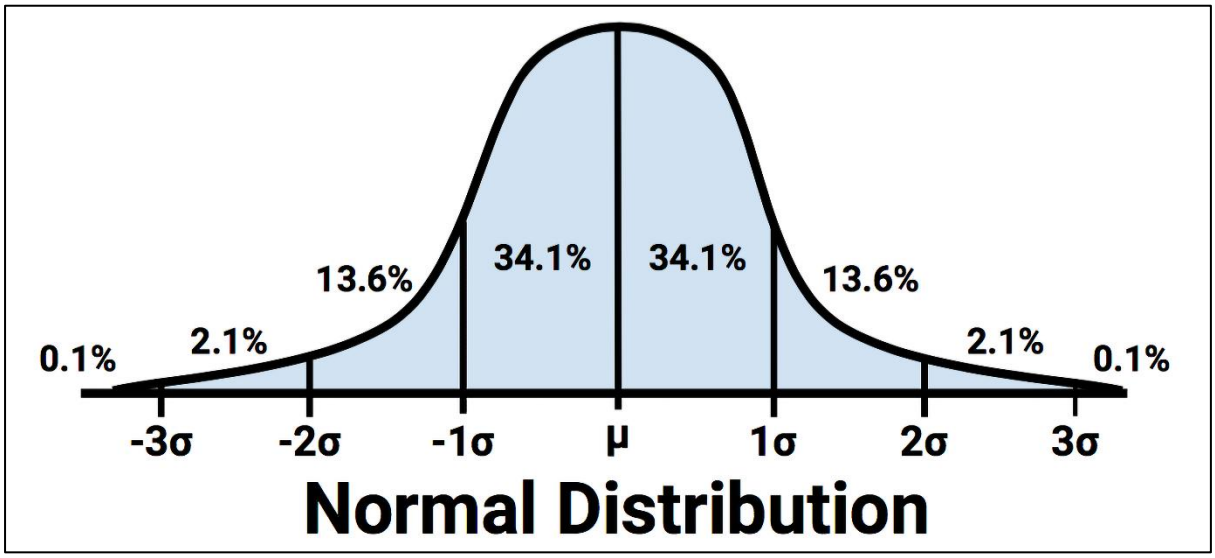

Histogram of Used Car Prices
Frequency
0
10
20
30
40
50
5,000
10,000
15,000
20,000
Price ($)
Histogram of Used Car Mileage
Frequency
0
10
20
30
40
50
60
70
0
50,000
100,000
150,000
Odometer (mi.)


Right Skew
No Skew
Left Skew

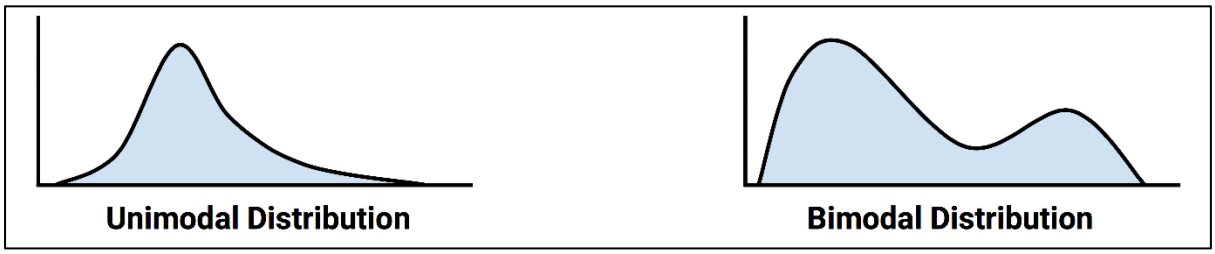

Uniform Distribution

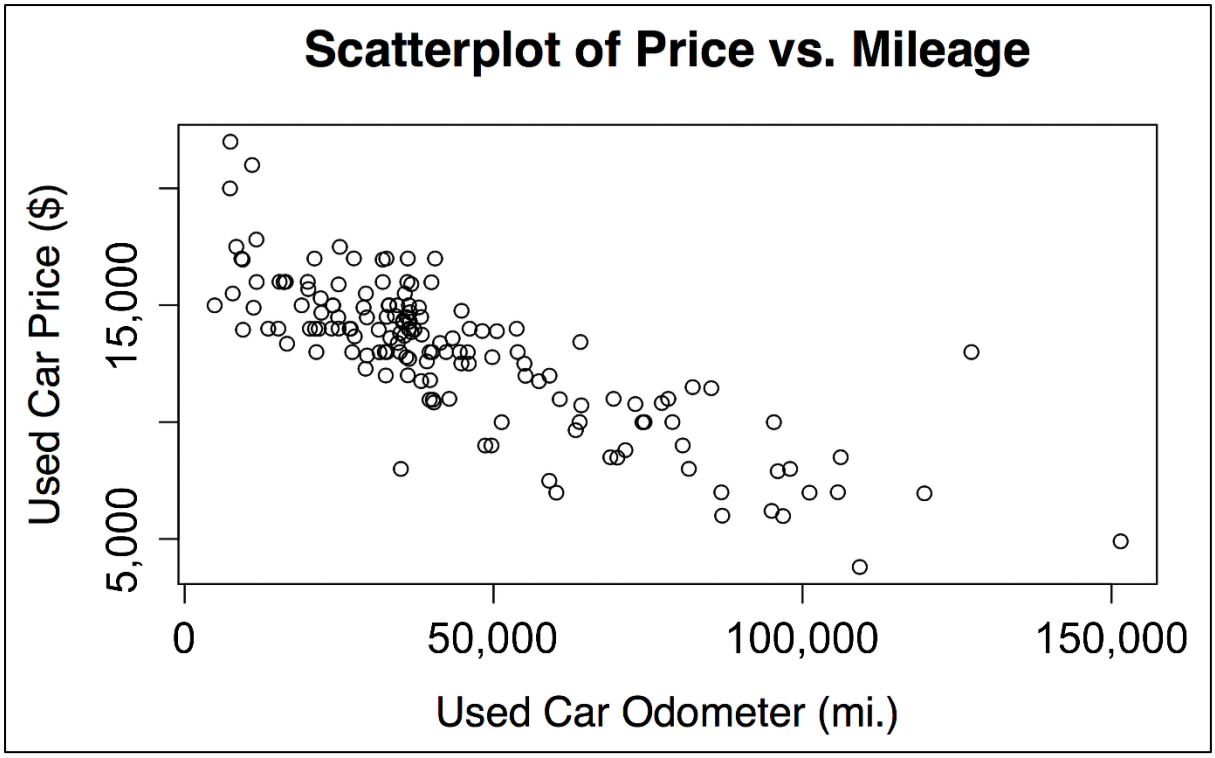


Normal Distribution

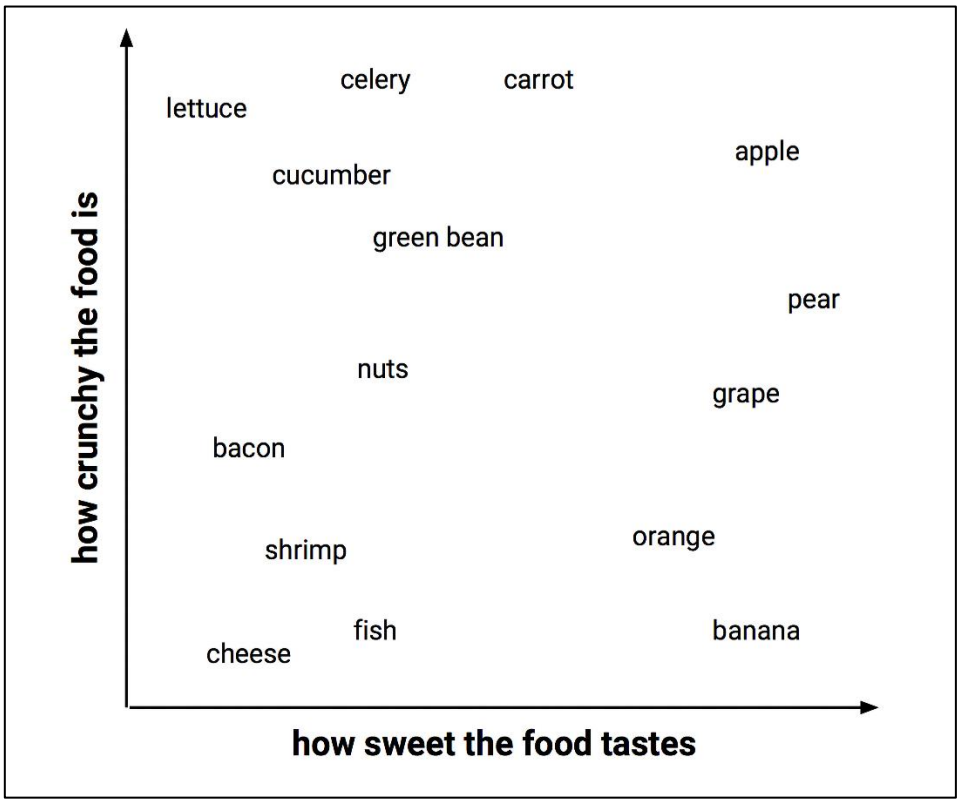


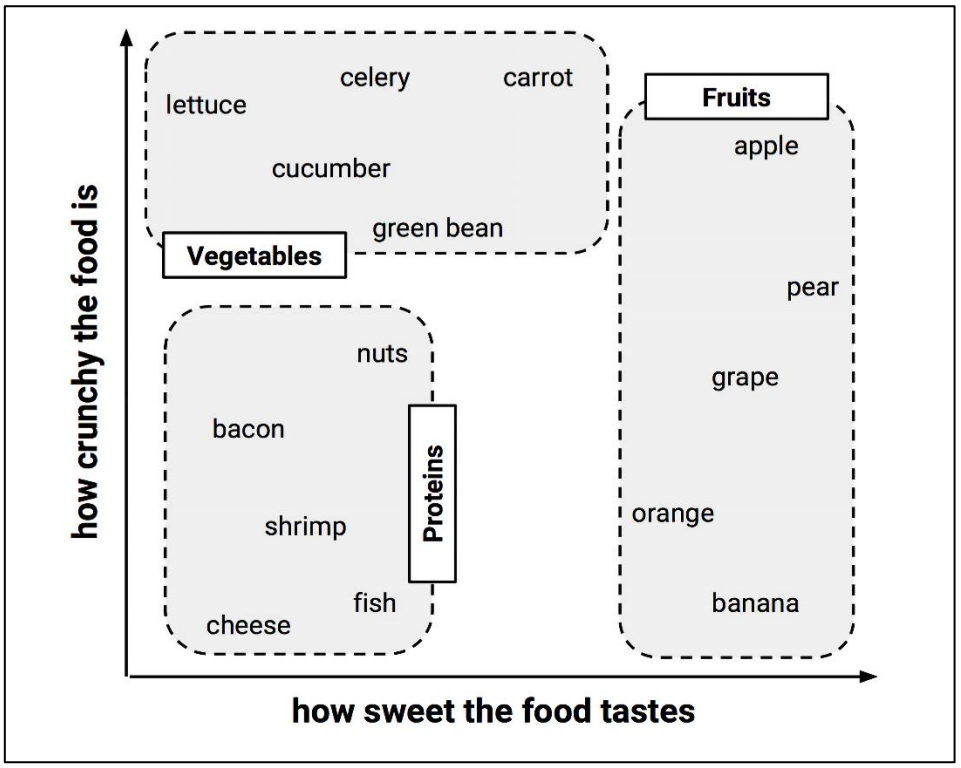
Normal Distribution

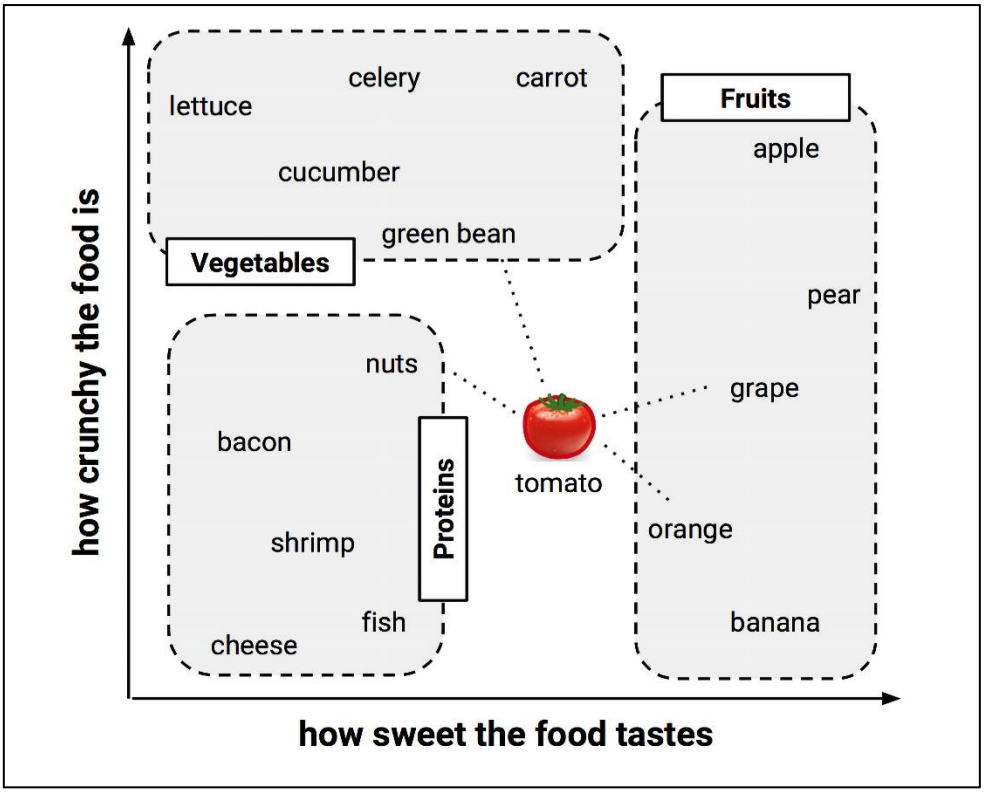


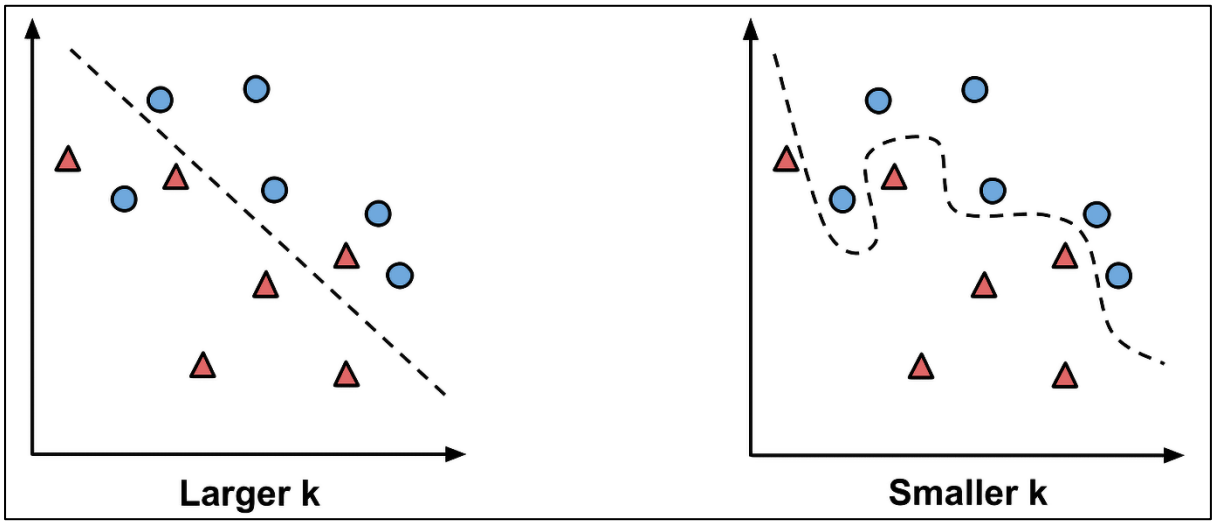

Scatterplot of Price vs. Mileage
Used Car Price ($)
5,000
15,000
0
50,000
100,000
150,000
Used Car Odometer (mi.)

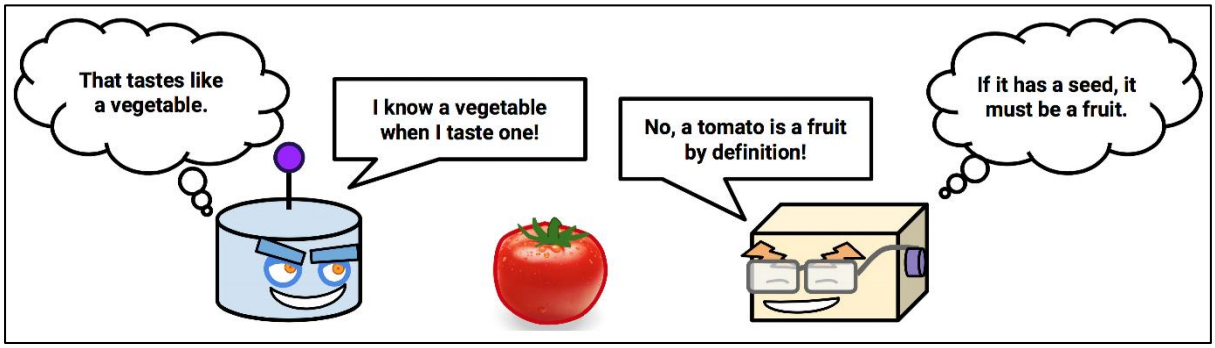
Chapter 3: Lazy Learning - Classification Using Nearest Neighbors






how crunchy the food is
how sweet the food tastes
Vegetables
Fruits
Proteins
lettuce
celery
carrot
cucumber
green bean
apple
pear
grape
orange
banana
tomato
nuts
bacon
shrimp
fish
cheese


Larger k
Smaller k


That tastes like a vegetable.
I know a vegetable when I taste one!
No, a tomato is a fruit by definition!
If it has a seed, it must be a fruit.

kNN classification syntax

using the `knn()` function in the `class` package

Building the classifier and making predictions:

```
p <- knn(train, test, class, k)
```

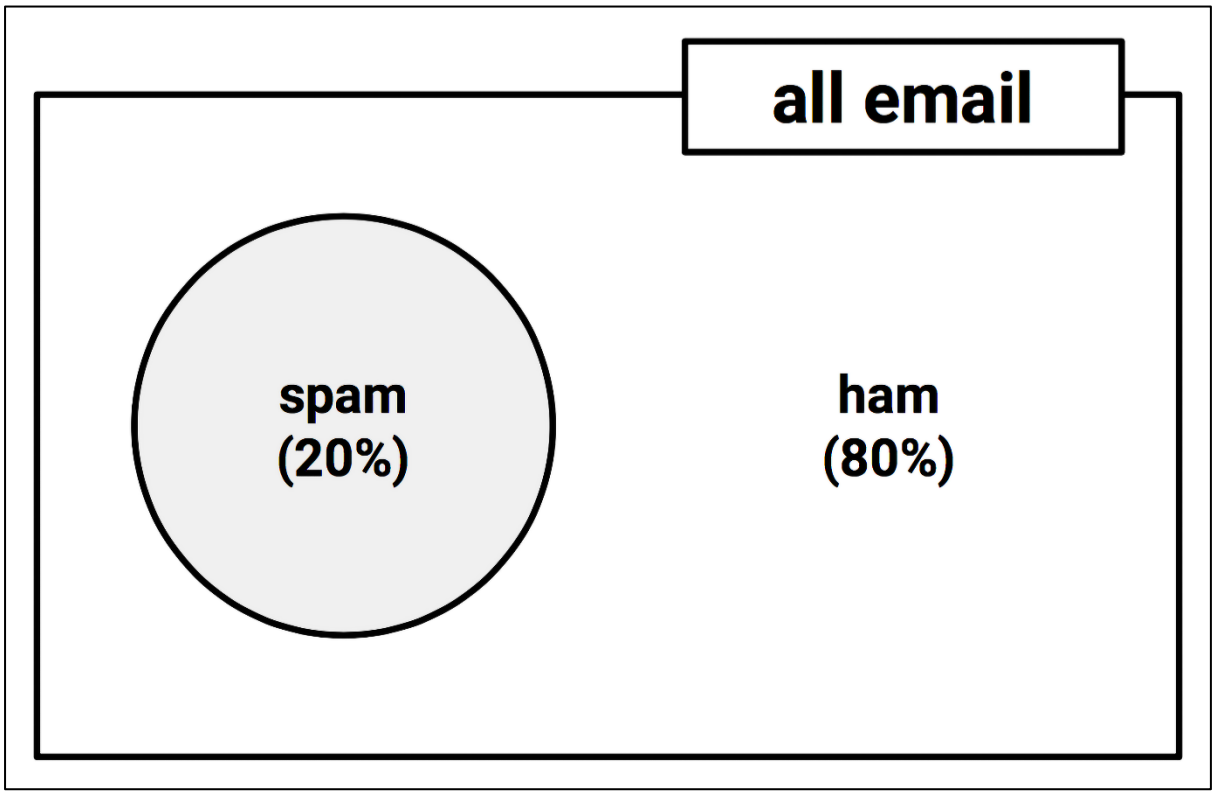
- `train` is a data frame containing numeric training data
- `test` is a data frame containing numeric test data
- `class` is a factor vector with the class for each row in the training data
- `k` is an integer indicating the number of nearest neighbors

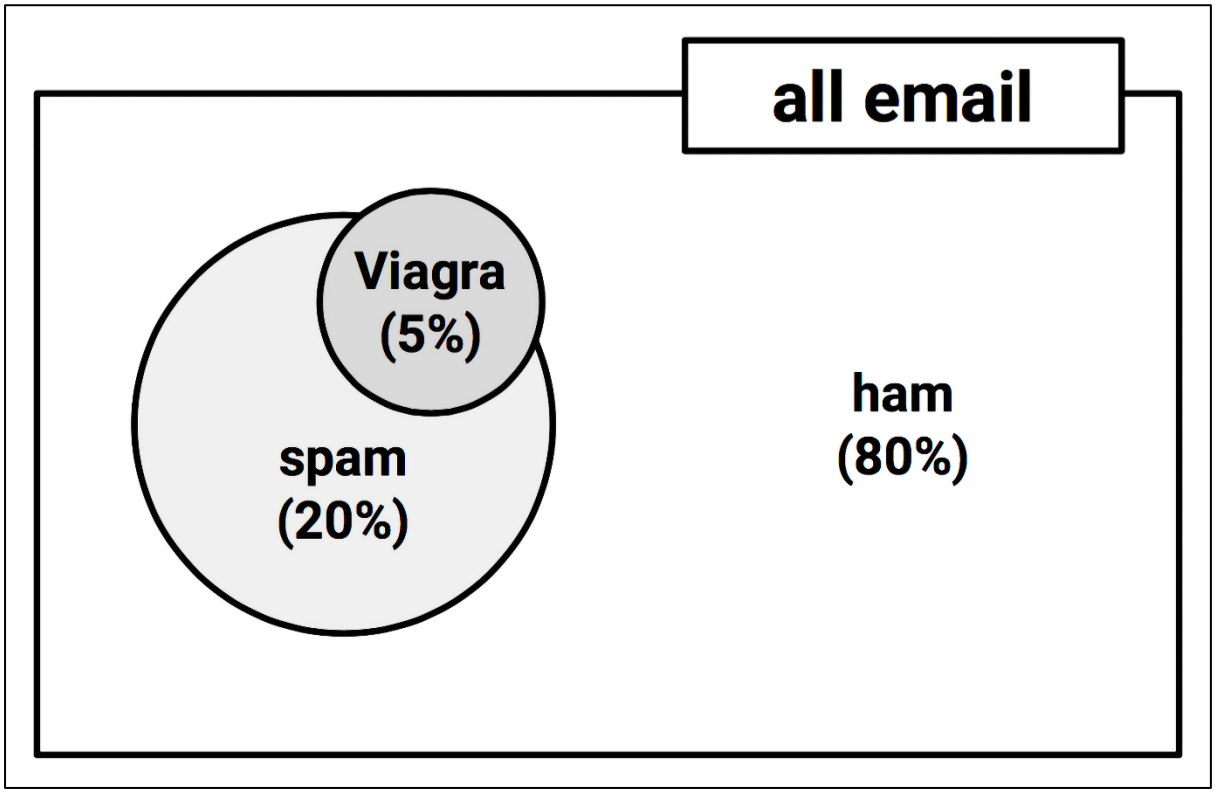
The function returns a factor vector of predicted classes for each row in the test data frame.

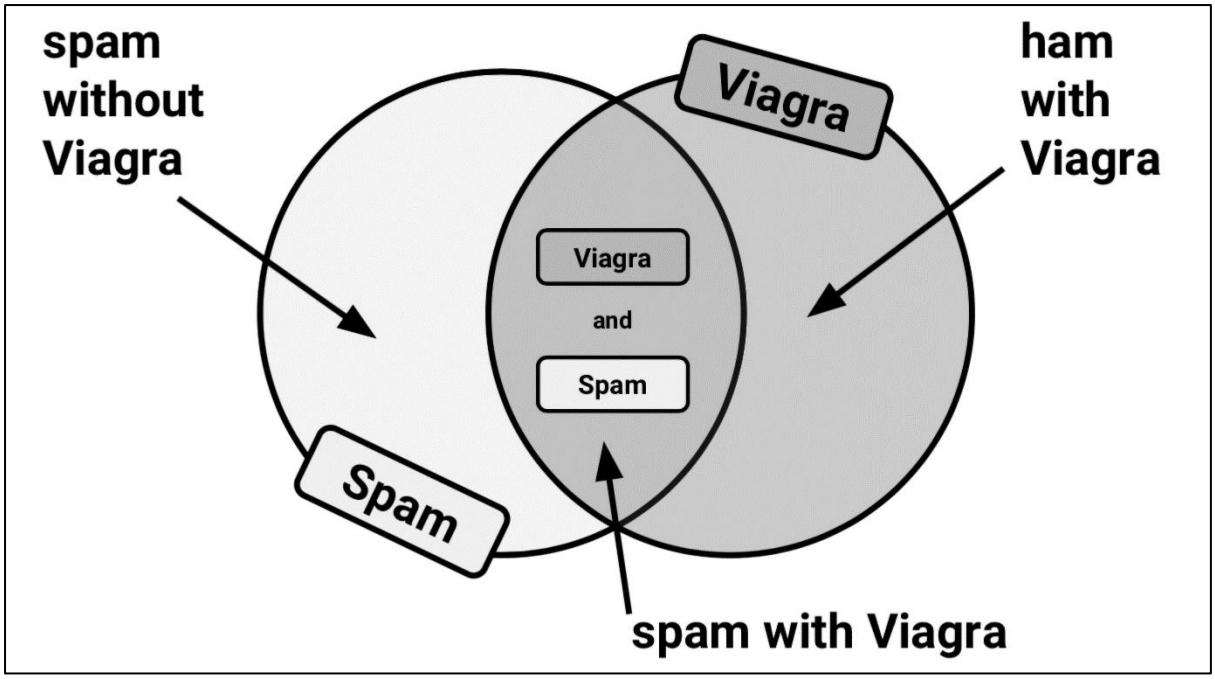
Example:

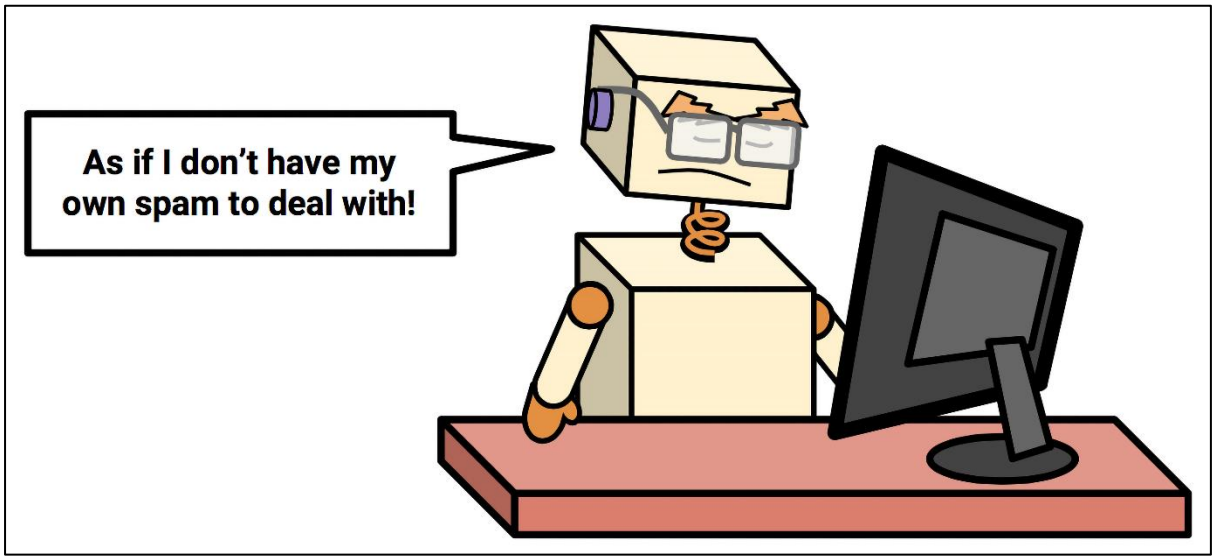
```
wbcd_pred <- knn(train = wbcd_train, test = wbcd_test,
                 cl = wbcd_train_labels, k = 3)
```

Chapter 4: Probabilistic Learning – Classification Using Naive Bayes









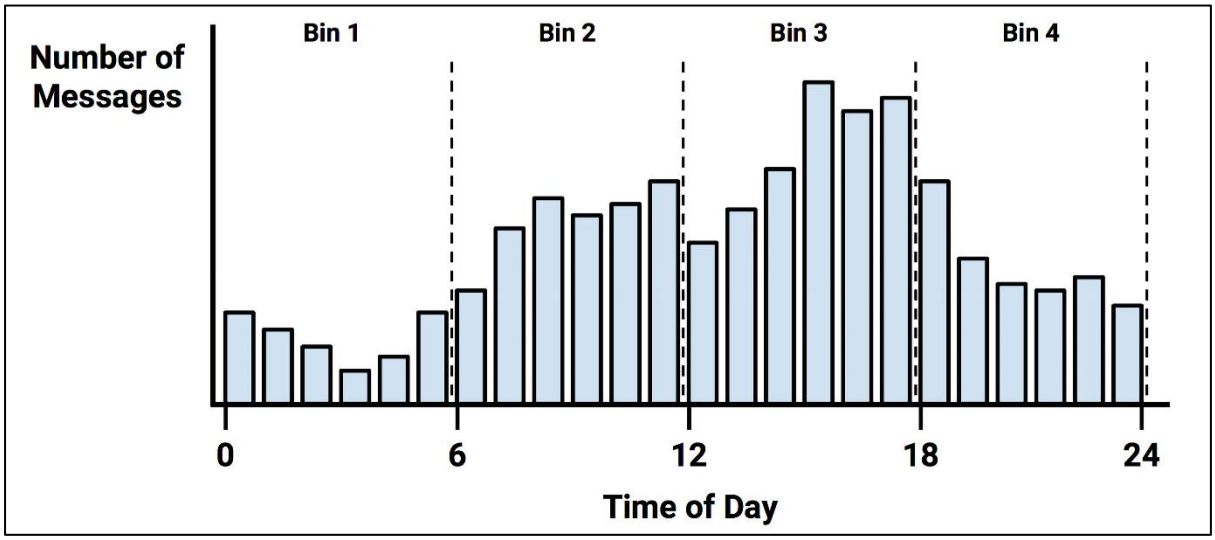
$$P(\text{spam}|\text{Viagra}) = \frac{P(\text{Viagra}|\text{spam})P(\text{spam})}{P(\text{Viagra})}$$

likelihood · prior probability · posterior probability · marginal likelihood

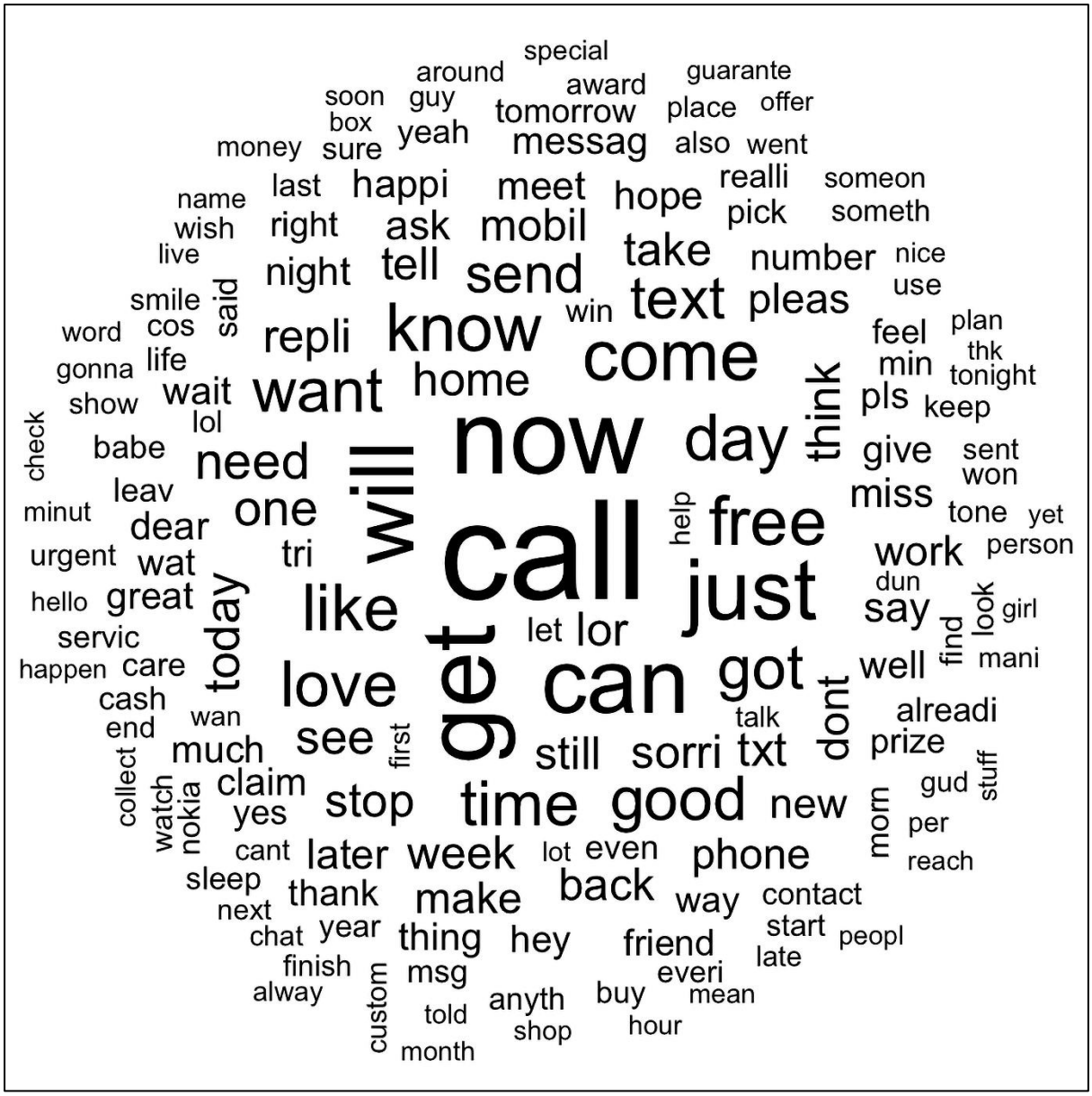
	Viagra		
Frequency	Yes	No	Total
spam	4	16	20
ham	1	79	80
Total	5	95	100

	Viagra		
Likelihood	Yes	No	Total
spam	4 / 20	16 / 20	20
ham	1 / 80	79 / 80	80
Total	5 / 100	95 / 100	100

	Viagra (W_1)		Money (W_2)		Groceries (W_3)		Unsubscribe (W_4)		
Likelihood	Yes	No	Yes	No	Yes	No	Yes	No	Total
spam	4 / 20	16 / 20	10 / 20	10 / 20	0 / 20	20 / 20	12 / 20	8 / 20	20
ham	1 / 80	79 / 80	14 / 80	66 / 80	8 / 80	71 / 80	23 / 80	57 / 80	80
Total	5 / 100	95 / 100	24 / 100	76 / 100	8 / 100	91 / 100	35 / 100	65 / 100	100



message #	balloon	balls	bam	bambling	band
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0
5	0	0	0	0	0


special
around
award
guarante
soon
guy
tomorrow
place
offer
box
yeah
messag
money
sure
also
went
realli
someon
last
happi
meet
hope
name
pick
someth
wish
right
ask
mobil
take
live
night
tell
send
number
nice
smile
said
win
text
pleas
use
word
cos
repli
know
come
feel
plan
thk
gonna
life
min
tonight
wait
want
home
pls
keep
show
lol
now
day
think
give
sent
check
babe
need
will
won
leav
miss
minut
one
call
free
help
tone
yet
dear
work
person
urgent
wat
tri
just
dun
hello
great
like
say
look
girl
today
let
lor
find
servic
get
can
got
well
mani
happen
care
love
dont
cash
talk
alreadi
end
wan
see
first
still
sorri
txt
prize
much
collect
claim
gud
stuff
watch
nokia
yes
stop
time
good
new
mom
per
cant
later
week
lot
even
phone
reach
sleep
thank
make
back
way
contact
next
chat
year
thing
hey
friend
start
peopl
late
finish
msg
everi
alway
custom
anyth
buy
mean
told
shop
hour
month


claim
your
mobile
£1000
win
latest
awarded
cash
call
chat
reply
won
line
mins
service
text
send
get
nokia
week
customer
new
tone
guaranteed
draw
phoneper
this
will
urgent
please
stop
contact
just
txt
150ppm
you
nowfree
prize
just
good
need
can
like
send
take
call
get
now
come
day
got
cant
its
how
still
night
back
see
sorry
but
tell
later
you
think
well
time
one
ill
much
today
lor
going
will
know
love
home
want
dont

Naive Bayes classification syntax

Using the `naive_bayes()` function in the `naivebayes` package

Building the classifier:

```
m <- naive_bayes(x, y, laplace = 0)
```

- `x` is a data frame or matrix containing training data
- `y` is a factor vector with the class for each row in the training data
- `laplace` is a number to control the Laplace estimator (by default, 0)

The function will return a Naive Bayes model object that can be used to make predictions.

Making predictions:

```
p <- predict(m, test, type = "class")
```

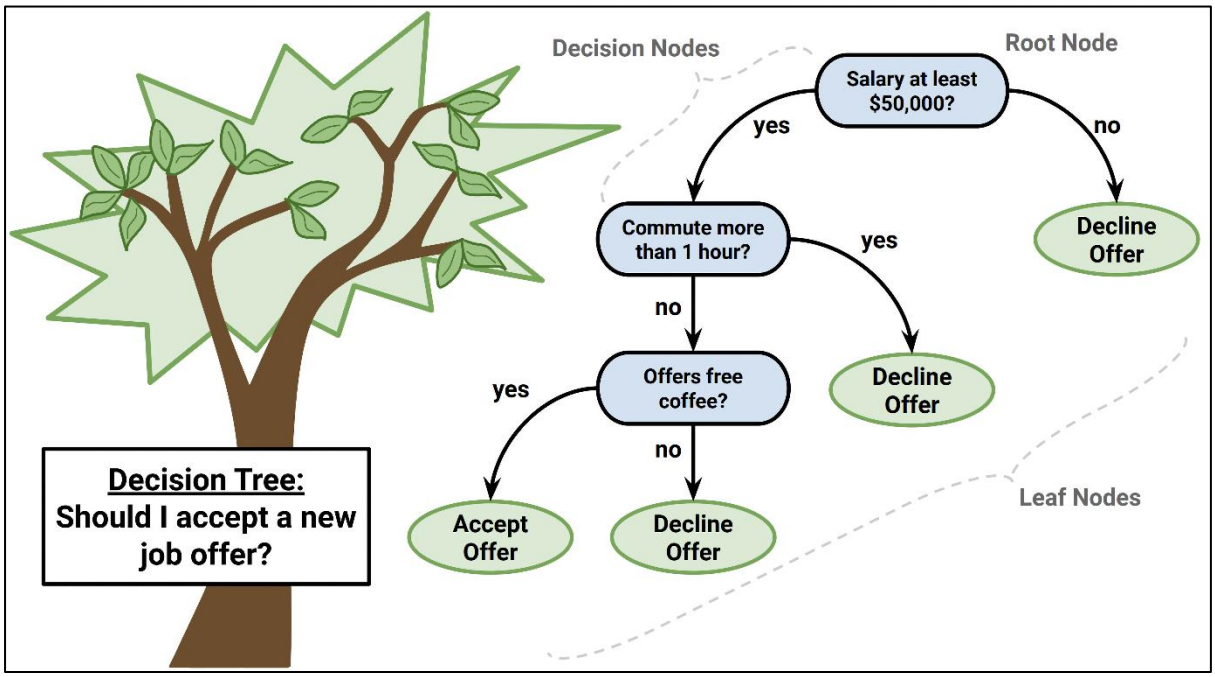
- `m` is a model trained by the `naive_bayes()` function
- `test` is a data frame or matrix containing test data with the same features as the training data used to build the classifier
- `type` is either `"class"` or `"prob"` and specifies whether the predictions should be the most likely class value or the raw predicted probabilities

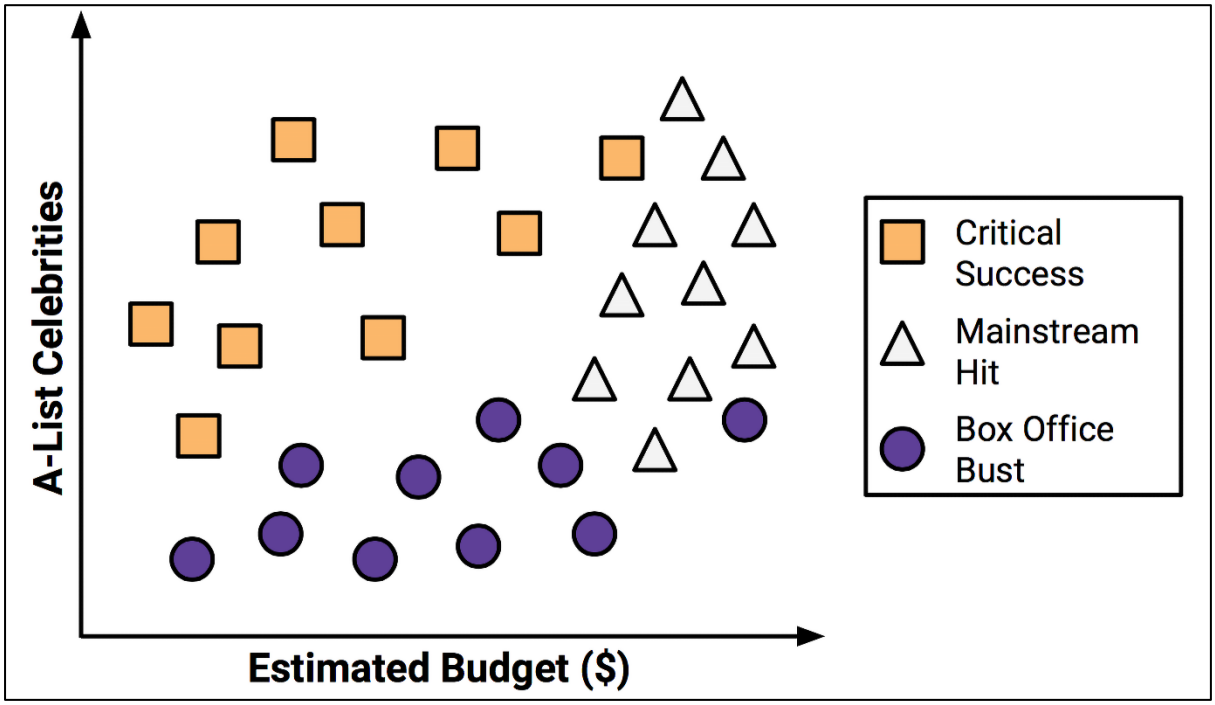
The function will return a vector of predicted class values or raw predicted probabilities depending upon the value of the `type` parameter.

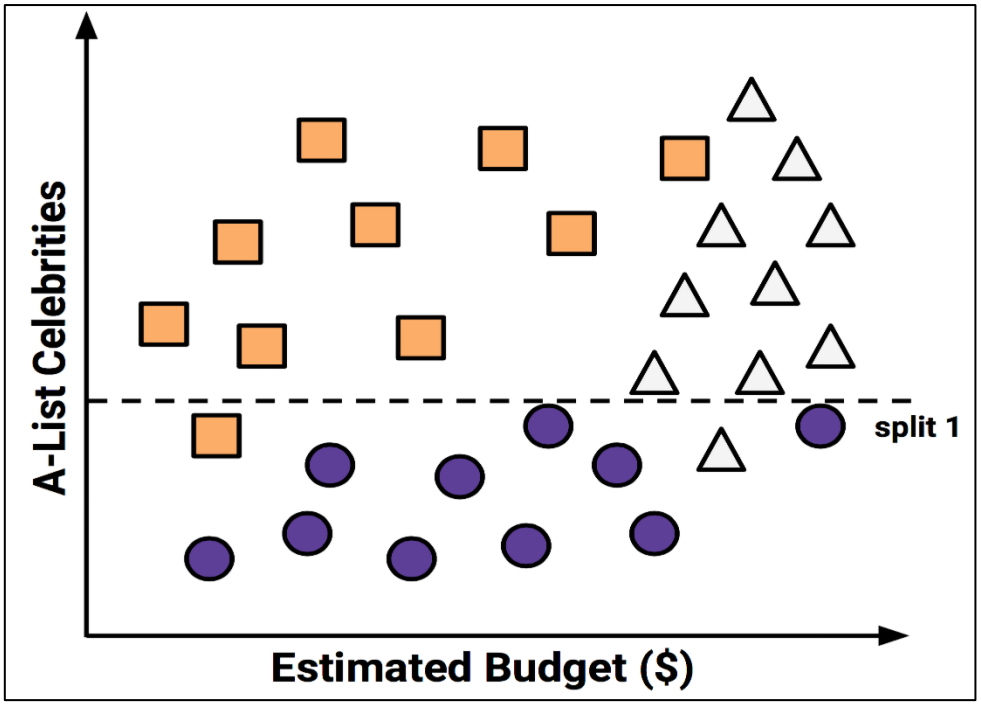
Example:

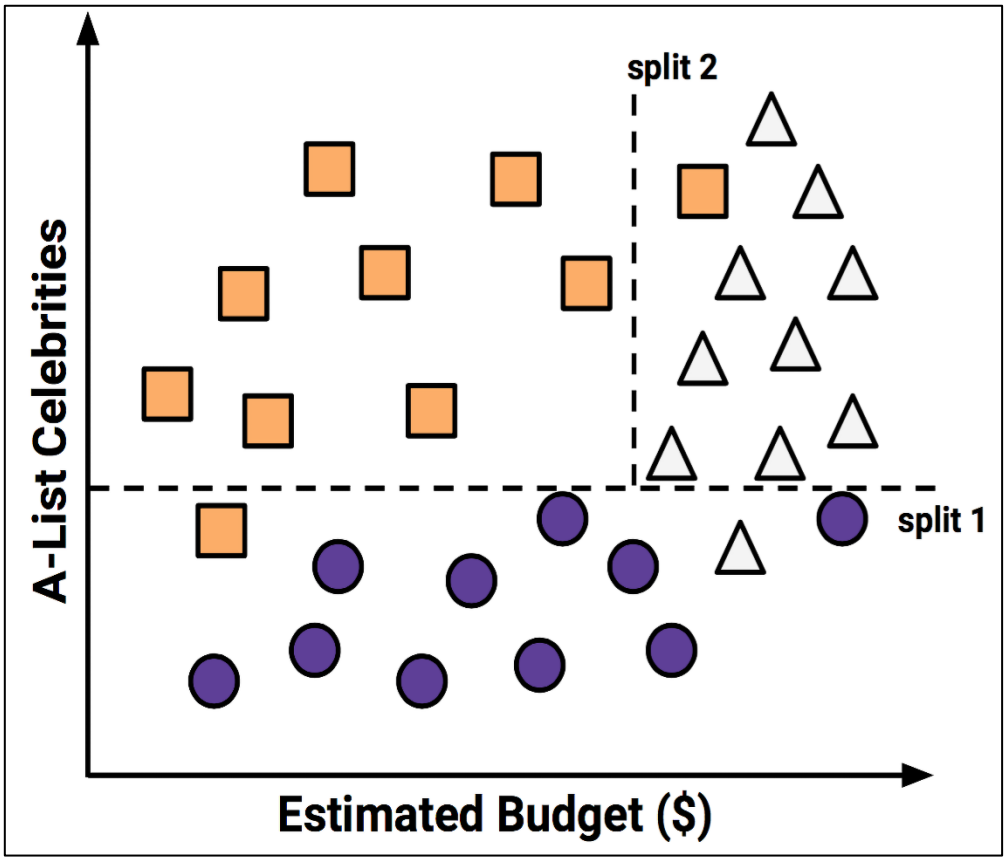
```
sms_classifier <- naive_bayes(sms_train, sms_type)
sms_predictions <- predict(sms_classifier, sms_test)
```

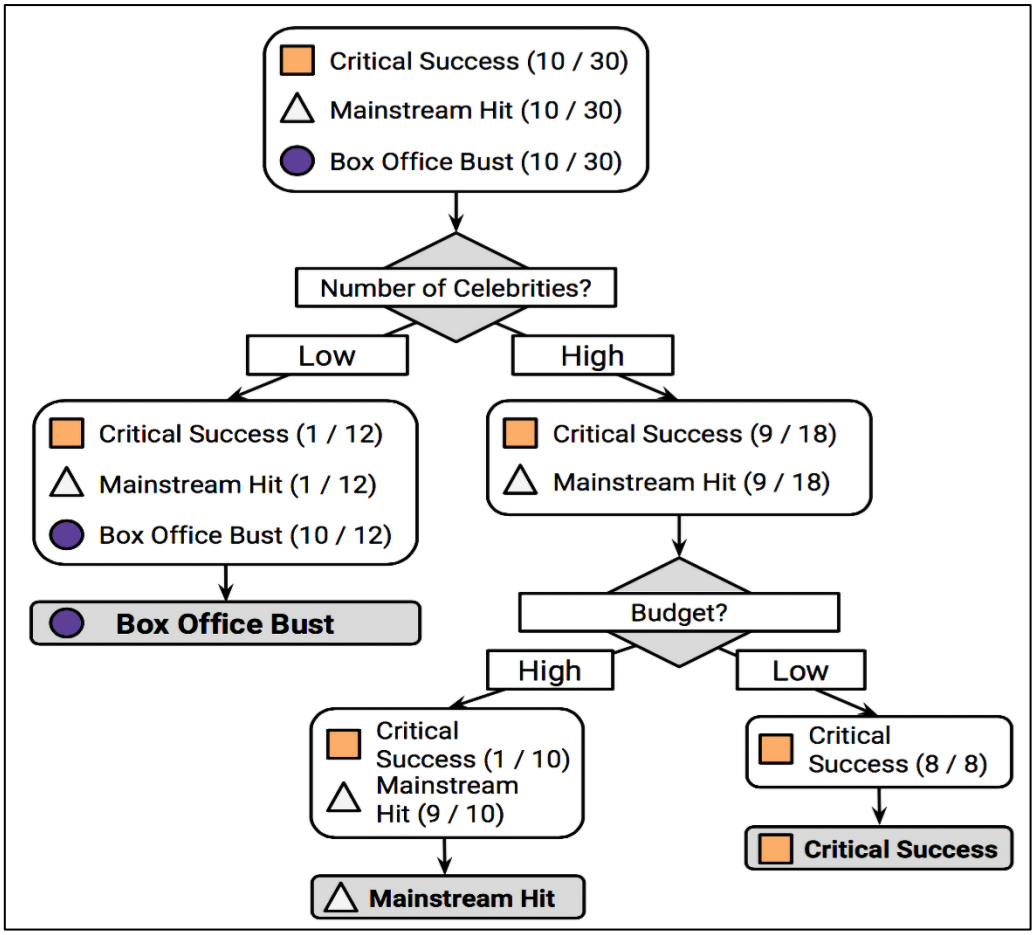
Chapter 5: Divide and Conquer - Classification Using Decision Trees and Rules

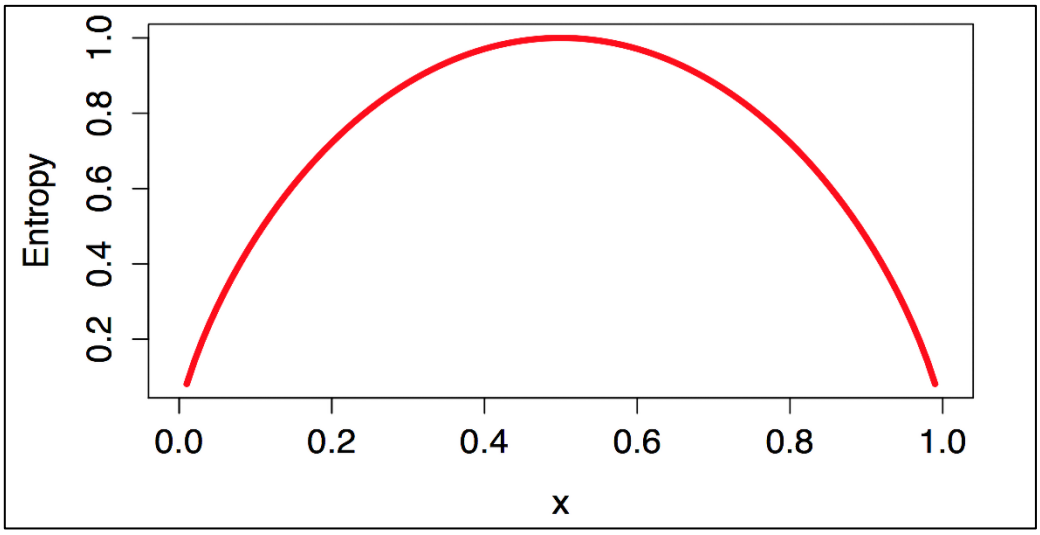





A-List Celebrities
split 1
Estimated Budget ($)


split 2
A-List Celebrities
split 1
Estimated Budget ($)


Critical Success (10 / 30)
Mainstream Hit (10 / 30)
Box Office Bust (10 / 30)
Number of Celebrities?
Low
High
Critical Success (1 / 12)
Mainstream Hit (1 / 12)
Box Office Bust (10 / 12)
Box Office Bust
Critical Success (9 / 18)
Mainstream Hit (9 / 18)
Budget?
High
Low
Critical Success (1 / 10)
Mainstream Hit (9 / 10)
Mainstream Hit
Critical Success (8 / 8)
Critical Success


Entropy
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0
x

C5.0 decision tree syntax

Using the `C5.0()` function in the `C50` package

Building the classifier:

```
m <- C5.0(class ~ predictors, data = mydata,
          trials = 1, costs = NULL)
```

- `class` is the column in the `mydata` data frame to be predicted
- `predictors` is an R formula specifying the features in the `mydata` data frame to use for prediction
- `trials` is an optional number to control the number of boosting iterations (set to 1 by default)
- `costs` is an optional matrix specifying costs associated with various types of errors

The function will return a C5.0 model object that can be used to make predictions.

Making predictions:

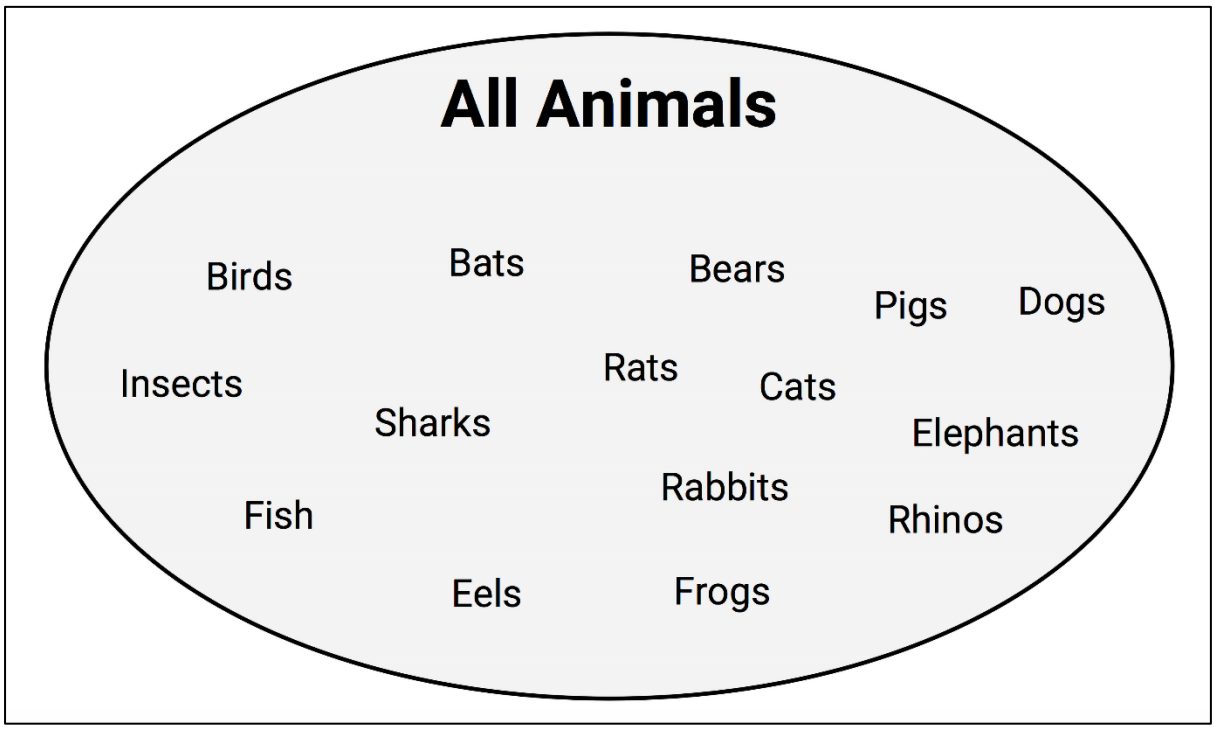
```
p <- predict(m, test, type = "class")
```

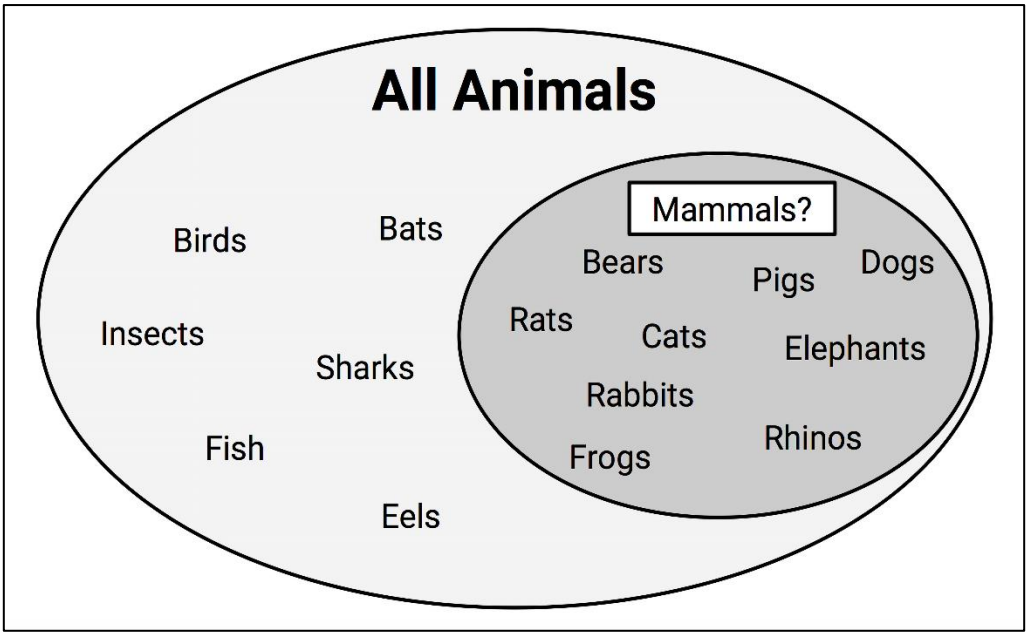
- `m` is a model trained by the `C5.0()` function
- `test` is a data frame containing test data with the same features as the training data used to build the classifier
- `type` is either `"class"` or `"prob"` and specifies whether the predictions should be the most probable class value or the raw predicted probabilities

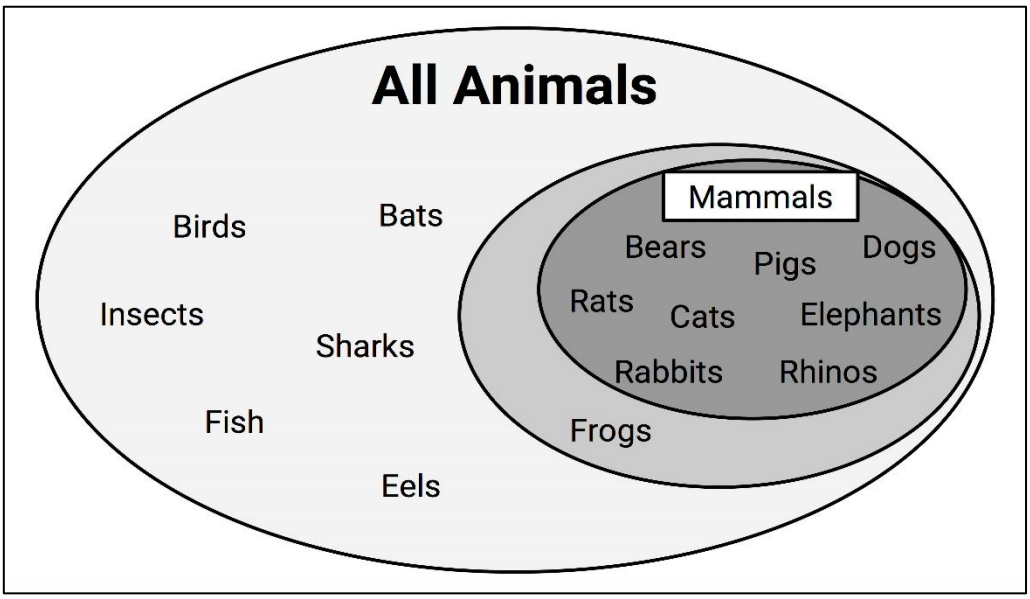
The function will return a vector of predicted class values or raw predicted probabilities depending upon the value of the `type` parameter.

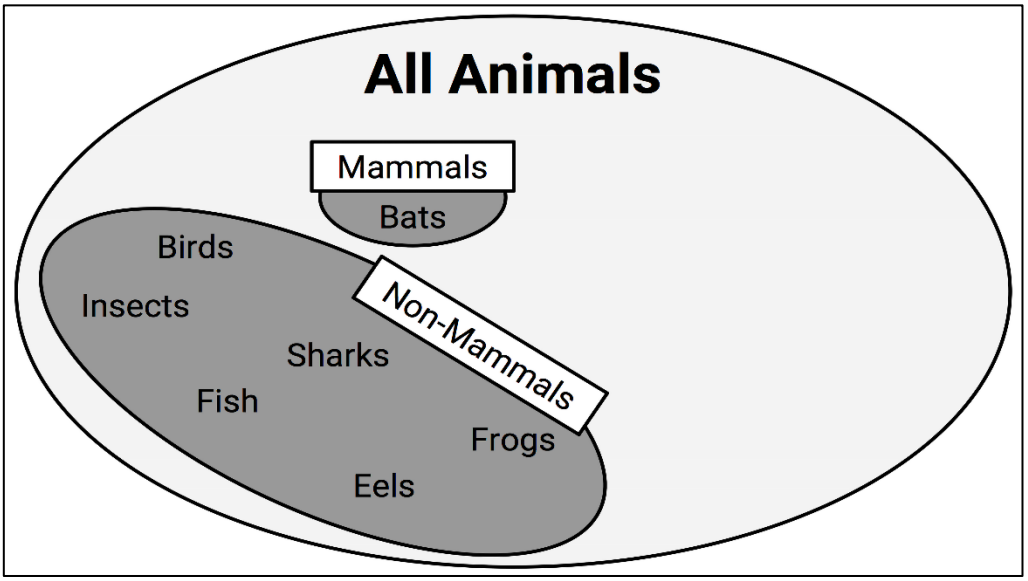
Example:

```
credit_model <- C5.0(default ~ ., data = credit_train)
credit_prediction <- predict(credit_model, credit_test)
```




All Animals
Mammals?
Birds
Bats
Bears
Pigs
Dogs
Rats
Cats
Elephants
Insects
Sharks
Rabbits
Rhinos
Fish
Frogs
Eels


All Animals
Mammals
Birds
Bats
Bears
Pigs
Dogs
Rats
Cats
Elephants
Insects
Sharks
Rabbits
Rhinos
Fish
Frogs
Eels


All Animals
Mammals
Bats
Birds
Insects
Non-Mammals
Sharks
Fish
Frogs
Eels

Animal	Travels By	Has Fur	Mammal
Bats	Air	Yes	Yes
Bears	Land	Yes	Yes
Birds	Air	No	No
Cats	Land	Yes	Yes
Dogs	Land	Yes	Yes
Eels	Sea	No	No
Elephants	Land	No	Yes
Fish	Sea	No	No
Frogs	Land	No	No
Insects	Air	No	No
Pigs	Land	No	Yes
Rabbits	Land	Yes	Yes
Rats	Land	Yes	Yes
Rhinos	Land	No	Yes
Sharks	Sea	No	No

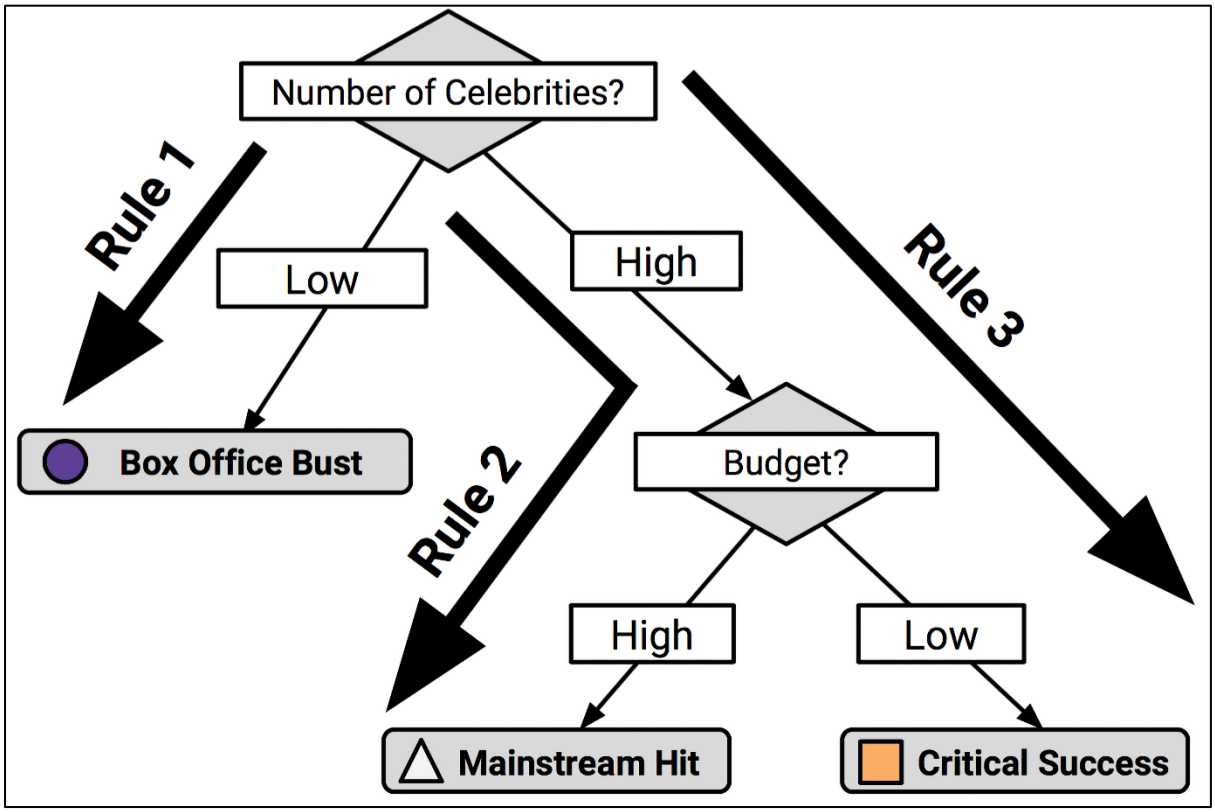
Full Dataset

Travels By	Predicted	Mammal	
Air	No	Yes	✖
Air	No	No	
Air	No	No	
Land	Yes	Yes	
Land	Yes	Yes	
Land	Yes	Yes	
Land	Yes	Yes	
Land	Yes	No	✖
Land	Yes	Yes	
Land	Yes	Yes	
Land	Yes	Yes	
Land	Yes	Yes	
Sea	No	No	
Sea	No	No	
Sea	No	No	

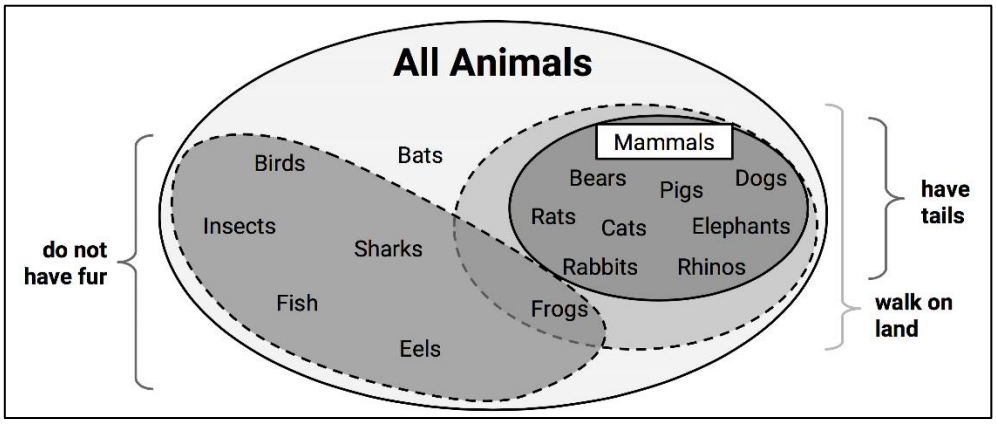
Rule for "Travels By"
Error Rate = 2 / 15

Has Fur	Predicted	Mammal	
No	No	No	
No	No	No	
No	No	Yes	✖
No	No	No	
No	No	No	
No	No	No	
No	No	Yes	✖
No	No	Yes	✖
No	No	No	
Yes	Yes	Yes	
Yes	Yes	Yes	
Yes	Yes	Yes	
Yes	Yes	Yes	
Yes	Yes	Yes	
Yes	Yes	Yes	

Rule for "Has Fur"
Error Rate = 3 / 15







1R classification rule syntax

Using the `OneR()` function in the `OneR` package

Building the classifier:

```
m <- OneR(class ~ predictors, data = mydata)
```

- `class` is the column in the `mydata` data frame to be predicted
- `predictors` is an R formula specifying the features in the `mydata` data frame to use for prediction
- `data` is the data frame in which `class` and `predictors` can be found

The function will return a OneR model object that can be used to make predictions.

Making predictions:

```
p <- predict(m, test)
```

- `m` is a model trained by the `OneR()` function
- `test` is a data frame containing test data with the same features as the training data used to build the classifier

The function will return a vector of predicted class values.

Example:

```
mushroom_classifier <- OneR(type ~ odor + cap_color,
                            data = mushroom_train)
mushroom_prediction <- predict(mushroom_classifier,
                               mushroom_test)
```

RIPPER classification rule syntax

Using the `JRip()` function in the `RWeka` package

Building the classifier:

```
m <- JRip(class ~ predictors, data = mydata)
```

- `class` is the column in the `mydata` data frame to be predicted
- `predictors` is an R formula specifying the features in the `mydata` data frame to use for prediction
- `data` is the data frame in which `class` and `predictors` can be found

The function will return a RIPPER model object that can be used to make predictions.

Making predictions:

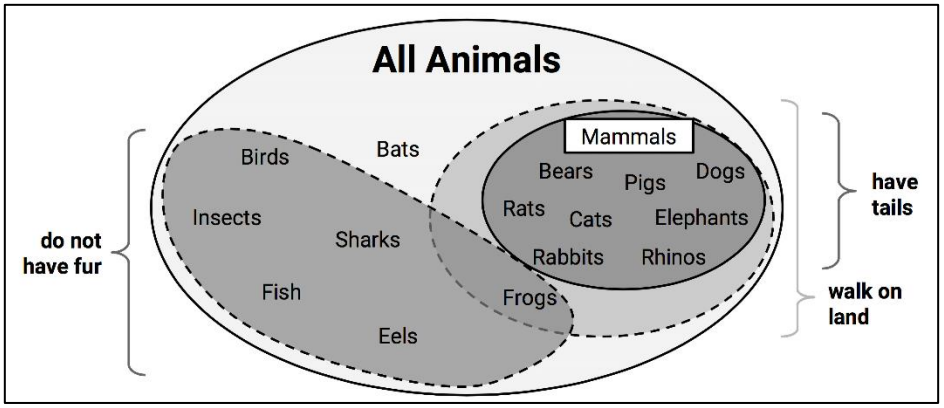
```
p <- predict(m, test)
```

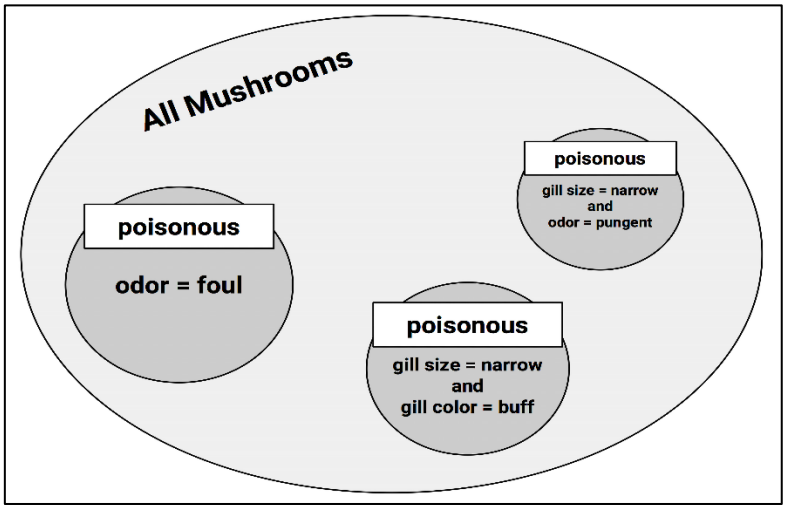
- `m` is a model trained by the `JRip()` function
- `test` is a data frame containing test data with the same features as the training data used to build the classifier

The function will return a vector of predicted class values.

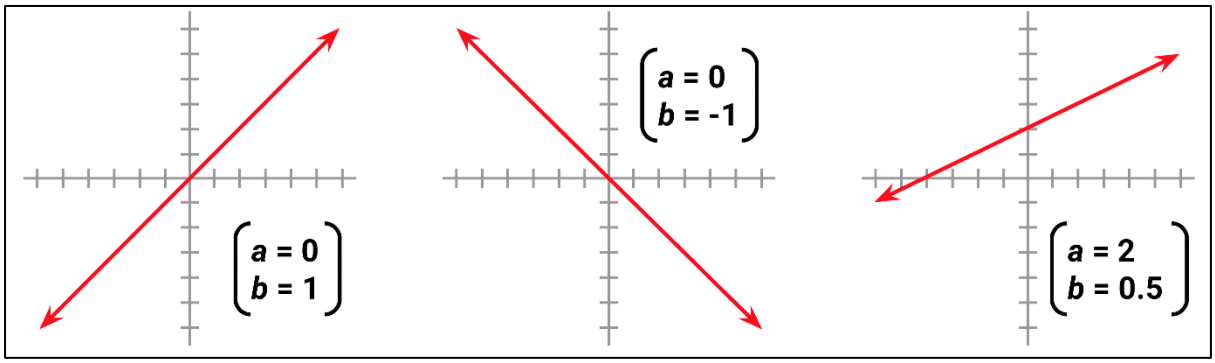
Example:

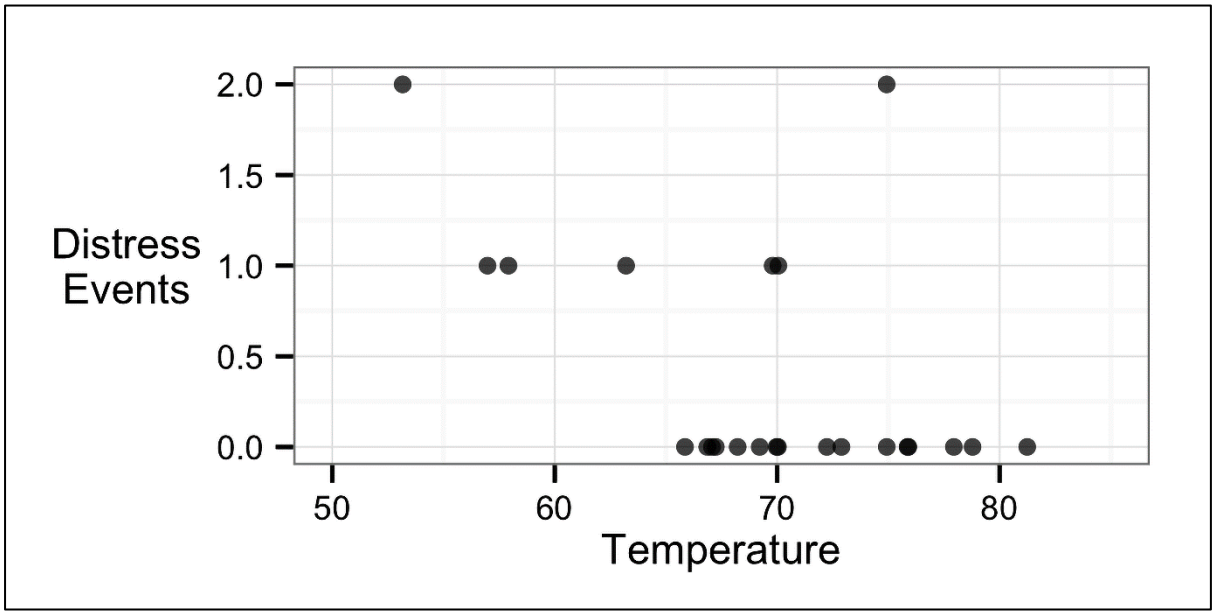
```
mushroom_classifier <- JRip(type ~ odor + cap_color,
                            data = mushroom_train)
mushroom_prediction <- predict(mushroom_classifier,
                               mushroom_test)
```

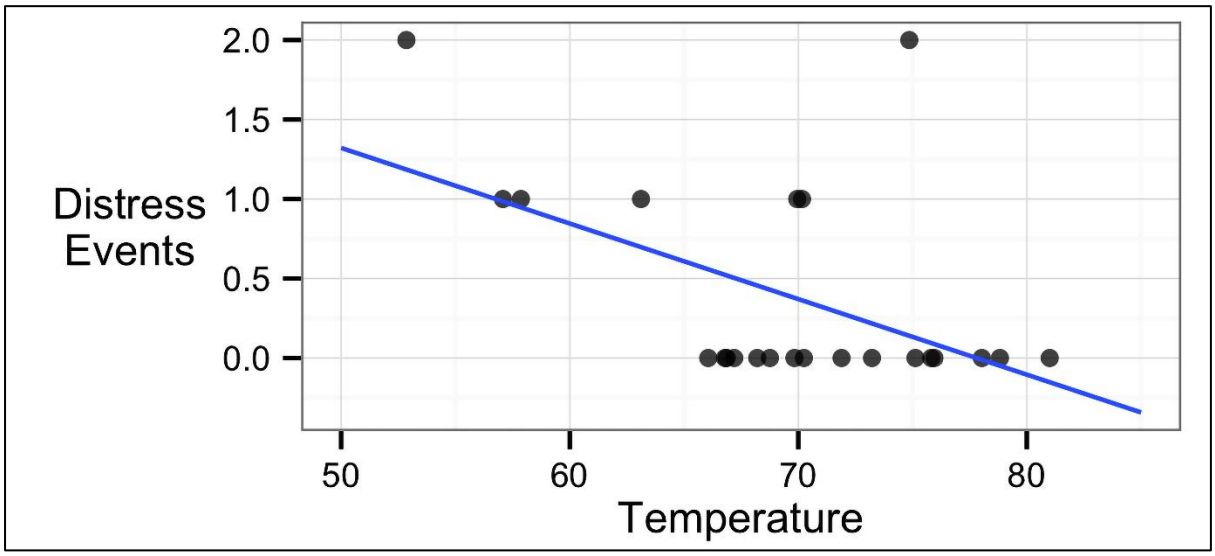


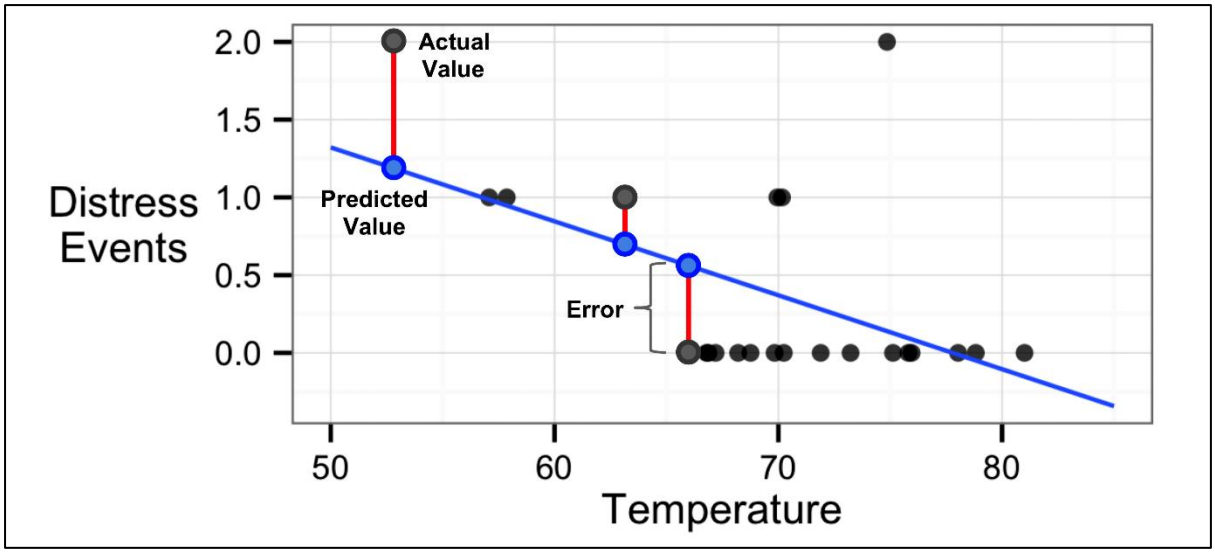


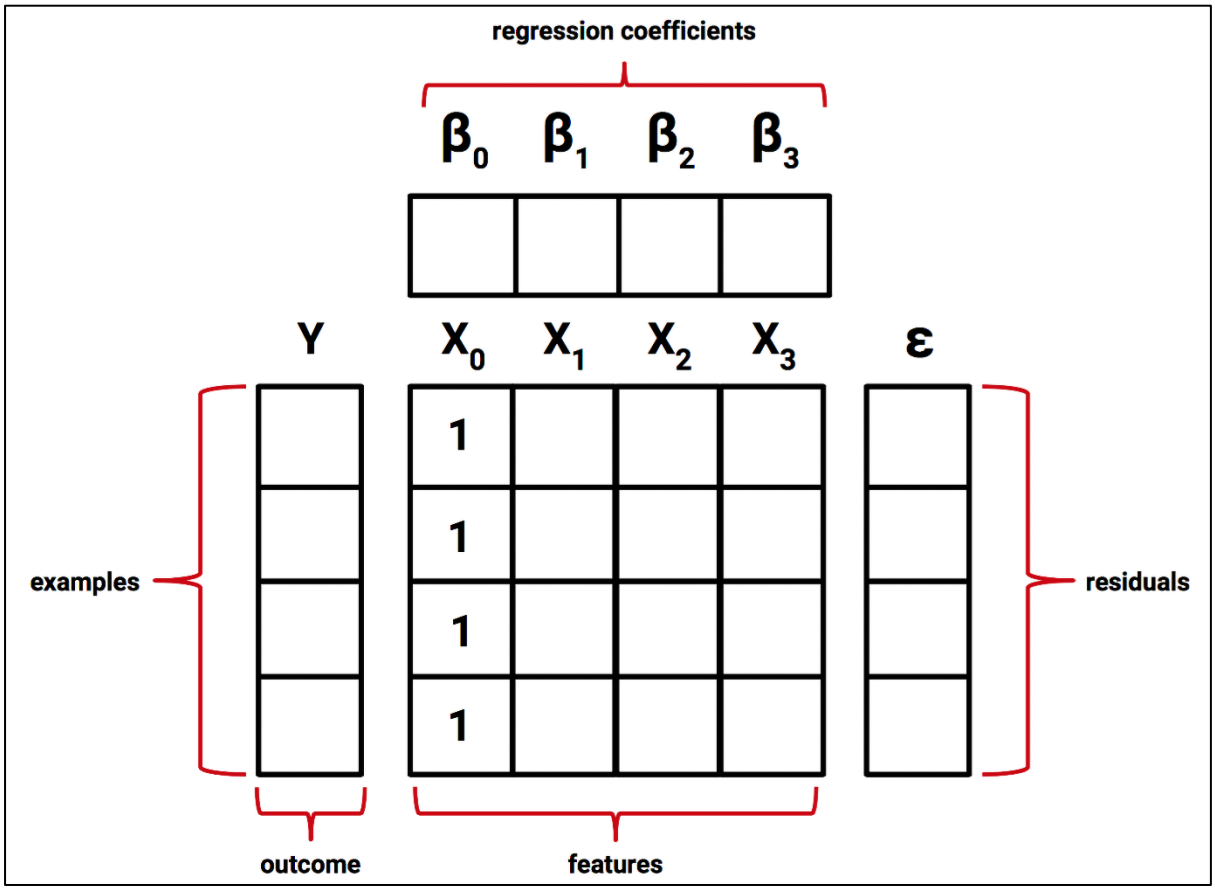
Chapter 6: Forecasting Numeric Data - Regression Methods

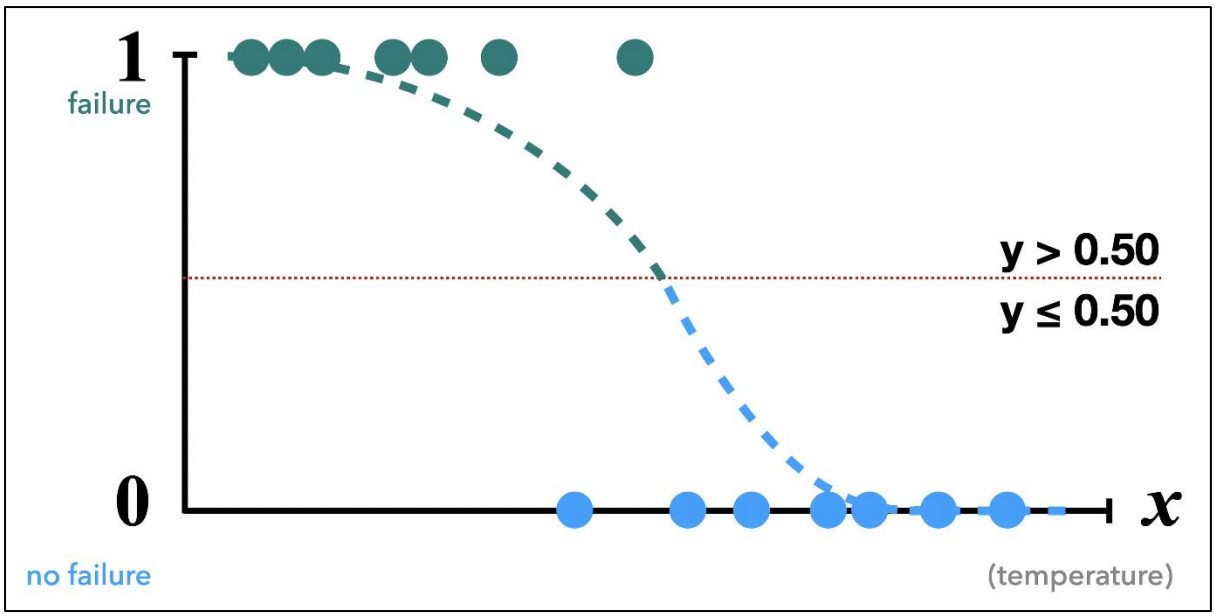


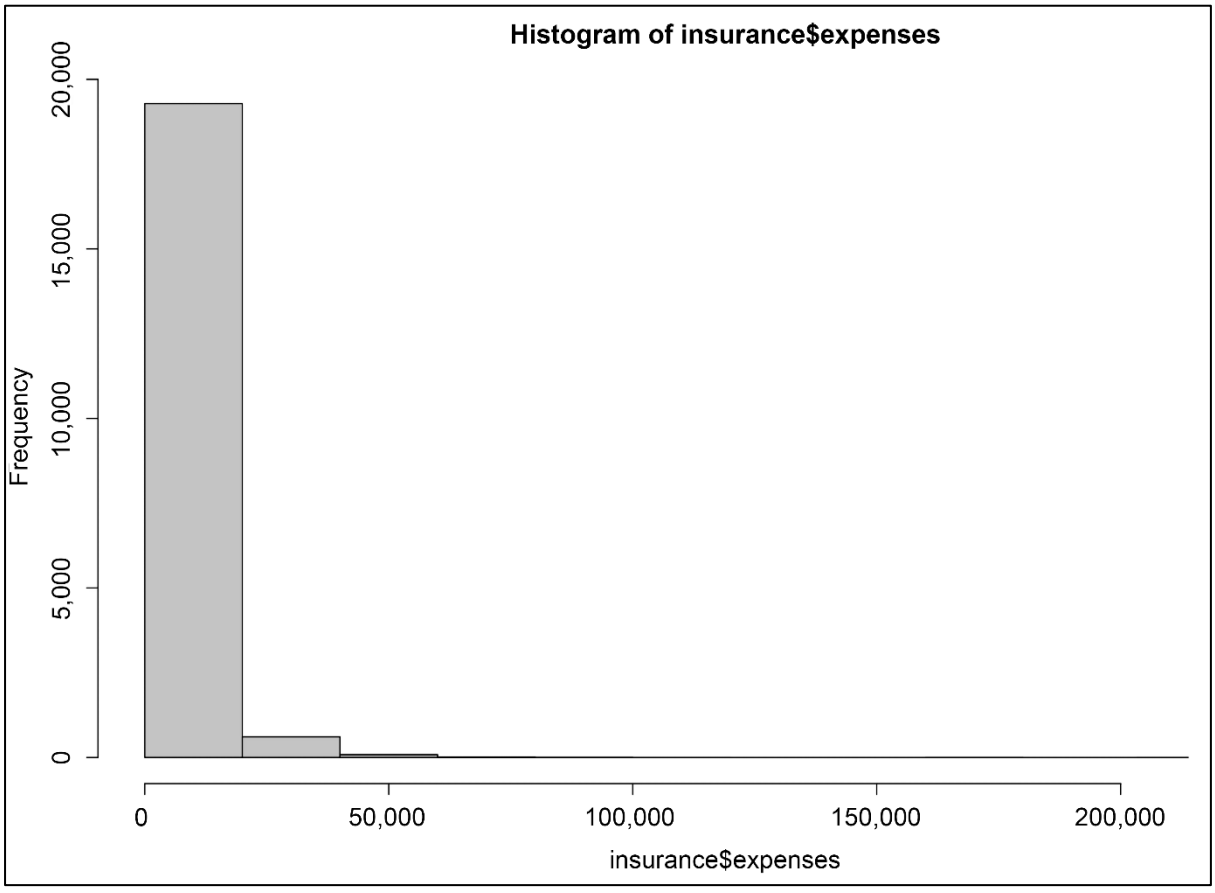


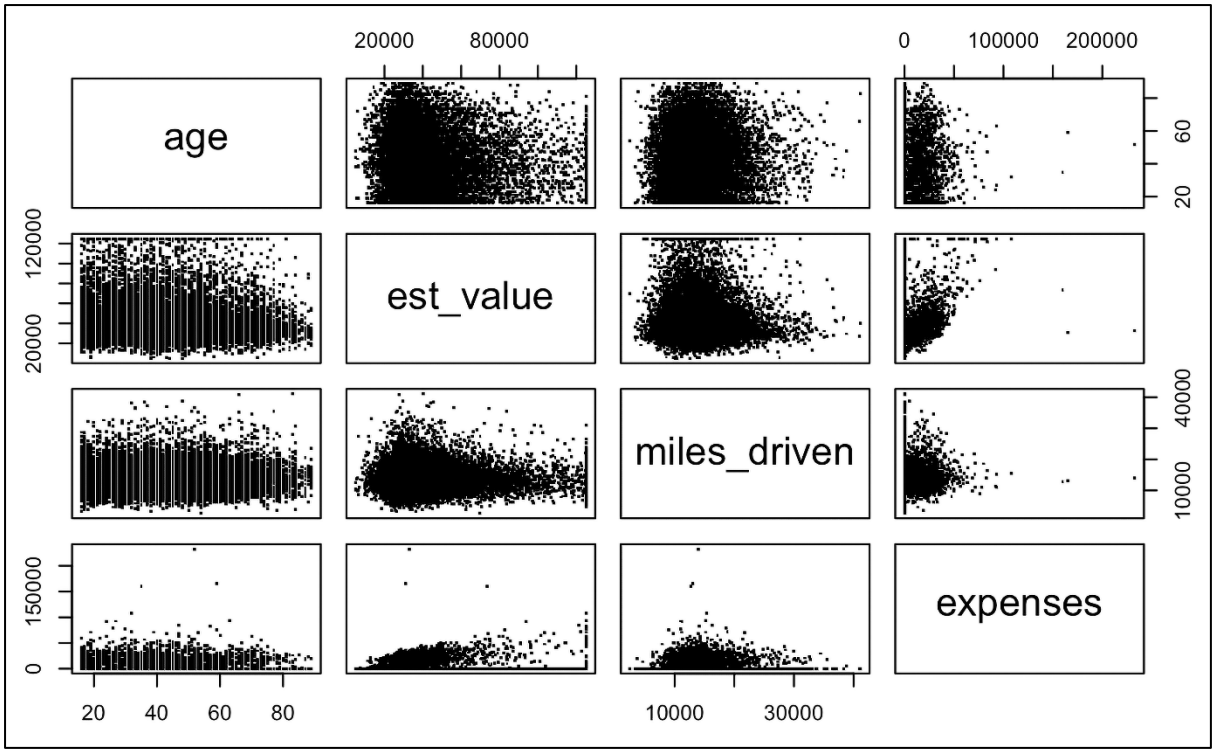


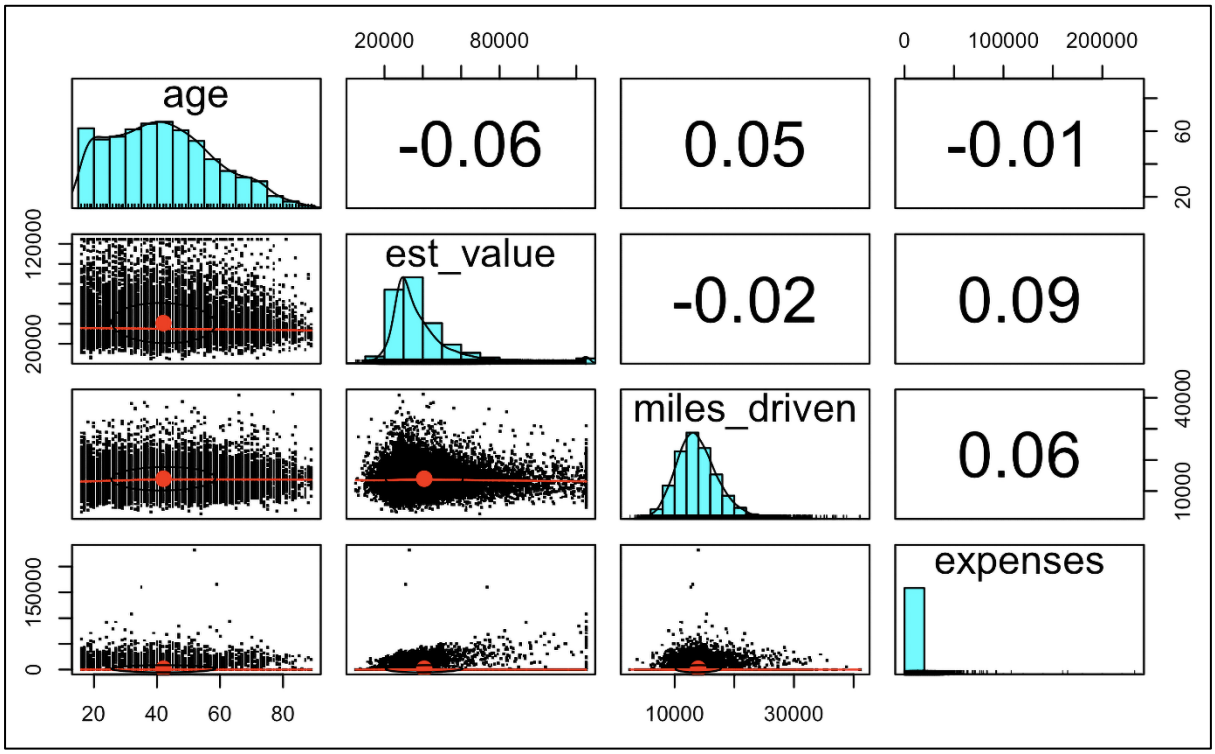

Actual Value
Predicted Value
Error
Distress Events
2.0
1.5
1.0
0.5
0.0
50
60
70
80
Temperature


regression coefficients
β0 β1 β2 β3
Y
X0 X1 X2 X3
ε
examples
residuals
1
1
1
1
outcome
features


1
failure
y > 0.50
y ≤ 0.50
0
x
no failure
(temperature)


Histogram of insurance$expenses
Frequency
0
5,000
10,000
15,000
20,000
0
50,000
100,000
150,000
200,000
insurance$expenses


age
est_value
miles_driven
expenses
20000
80000
0
100000
200000
20
60
120000
20000
40000
10000
150000
0
20
40
60
80
10000
30000


age
-0.06
0.05
-0.01
est_value
-0.02
0.09
miles_driven
0.06
expenses
20000
80000
0
100000
200000
20
60
120000
20000
40000
10000
150000
0
20
40
60
80
10000
30000

Multiple regression modeling syntax

using the `lm()` function in the `stats` package

Building the model:

```
m <- lm(dv ~ iv, data = mydata)
```

- `dv` is the dependent variable in the `mydata` data frame to be modeled
- `iv` is an R formula specifying the independent variables in the `mydata` data frame to use in the model
- `data` specifies the data frame in which the `dv` and `iv` variables can be found

The function will return a regression model object, which can be used to make predictions. Interactions between independent variables can be specified using the * operator.

Making predictions:

```
p <- predict(m, test)
```

- `m` is a model trained by the `lm()` function
- `test` is a data frame containing test data with the same features as the training data used to build the model

The function will return a vector of predicted values.

Example:

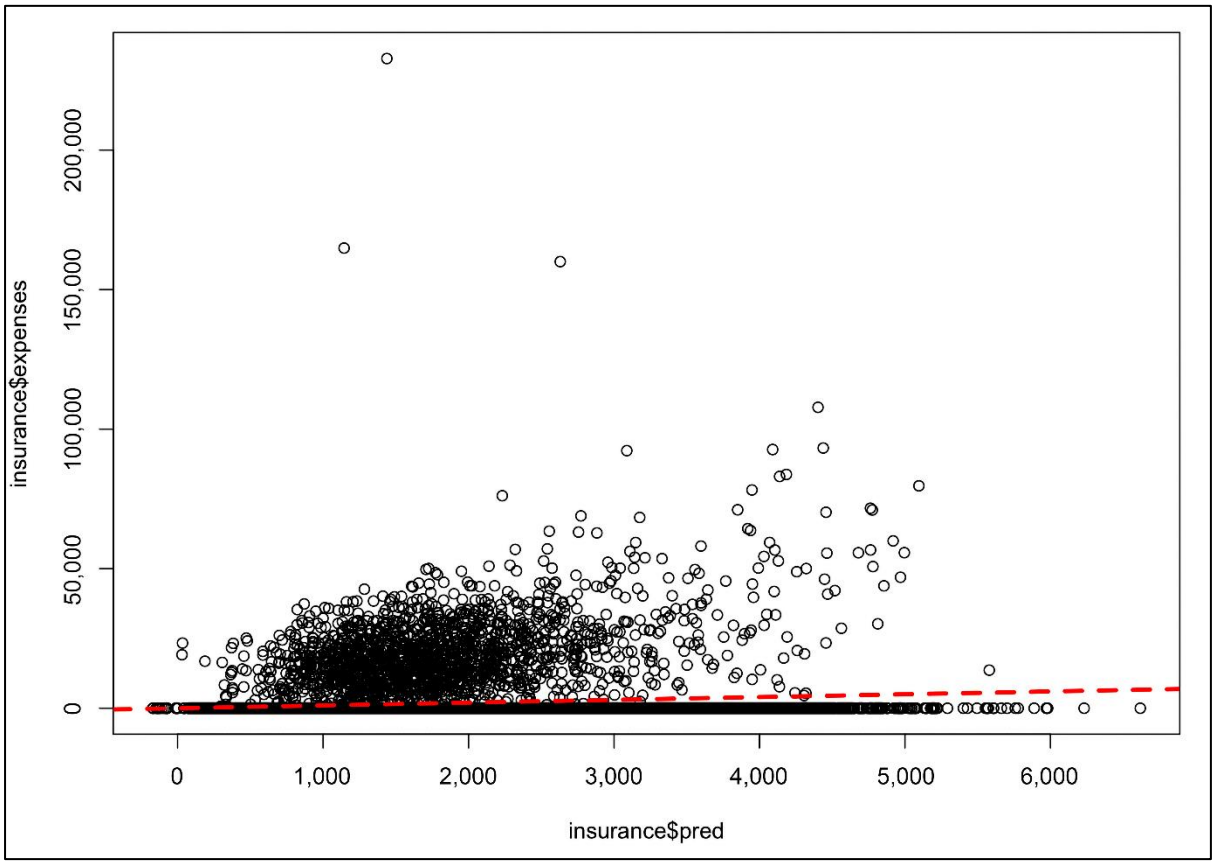
```
ins_model <- lm(charges ~ age + est_value + miles_driven,
                data = insurance)
ins_pred <- predict(ins_model, insurance_test)
```

```
Call:
lm(formula = expenses ~ ., data = insurance)

Residuals:
   Min     1Q Median     3Q    Max
 -6707  -1989  -1492  -1057 231252

Coefficients:
                         Estimate   Std. Error t value             Pr(>|t|)
(Intercept)         -1154.914856   351.370732  -3.287              0.00101 **
age                    -1.886027     3.144556  -0.600              0.54866
geo_areasuburban      191.078953   143.199245   1.334              0.18210
geo_areaurban         169.114255   157.850516   1.071              0.28402
vehicle_typeminivan   115.278619   276.579845   0.417              0.67683
vehicle_typesuv       -19.695001   117.990151  -0.167              0.86743
vehicle_typetruck      21.568360   153.630939   0.140              0.88835
est_value               0.031145     0.002497  12.475 < 0.0000000000000002 ***
miles_driven            0.118986     0.014327   8.305 < 0.0000000000000002 ***
college_grad_ind      -25.040302   115.578315  -0.217              0.82848
speeding_ticket_ind   155.824097   140.155213   1.112              0.26624
hard_braking_ind       11.845220   106.912005   0.111              0.91178
late_driving_ind      362.485502   224.655385   1.614              0.10665
clean_driving_ind    -239.047399   111.076229  -2.152              0.03140 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 6995 on 19986 degrees of freedom
Multiple R-squared:  0.01241,   Adjusted R-squared:  0.01176
F-statistic: 19.31 on 13 and 19986 DF,  p-value: < 0.00000000000000022
```



Logistic regression modeling syntax

using the `glm()` function in the `stats` package

Building the model:

```
m <- glm(dv ~ iv, data = mydata,
         family = binomial(link = "logit")
```

- `dv` is the dependent variable in the `mydata` data frame to be modeled
- `iv` is an R formula specifying the independent variables in the `mydata` data frame to use in the model
- `data` specifies the data frame in which the `dv` and `iv` variables can be found
- `family` sets the distribution and link function to be used for the GLM; the above syntax is for logistic regression, but note that linear regression is used by default if `family` is unspecified (see `?family` for help with other GLM variants)

The function will return a GLM model object, which can be used to make predictions. Interactions between independent variables can be specified using the * operator.

Making predictions:

```
p <- predict(m, test, type = "link")
```

- `m` is a model trained by the `glm()` function
- `test` is a data frame containing test data with the same features as the training data used to build the model
- `type` specifies the type of prediction to return, either `"link"` for predicted log-odds values (the default) or `"response"` for predicted probabilities (typically the more useful of the two types)

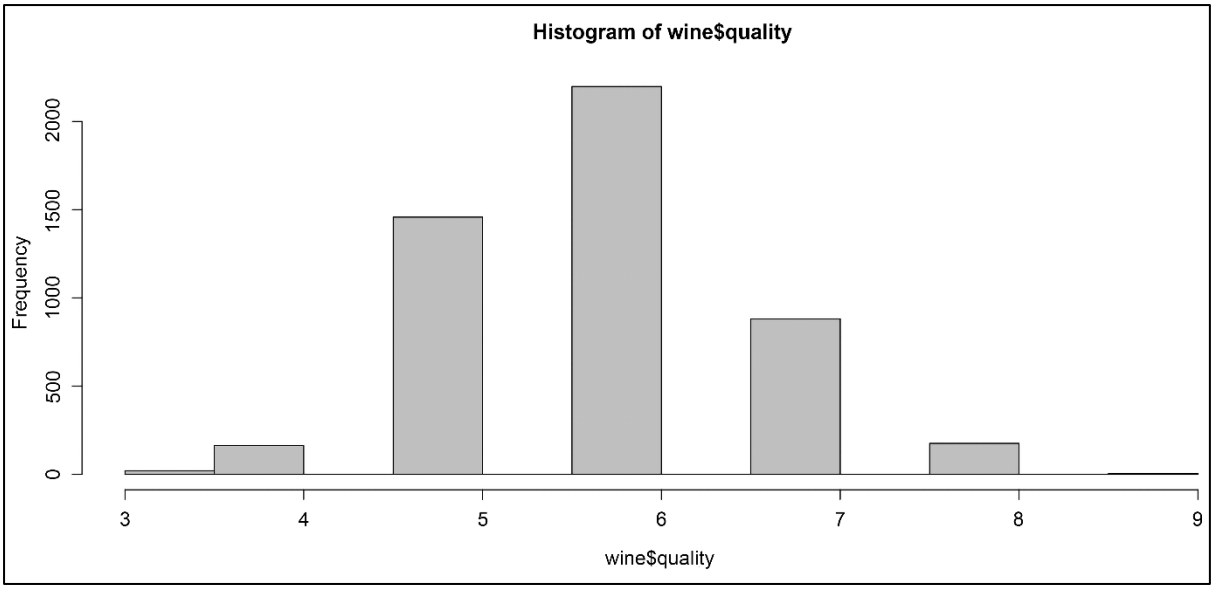
The function will return a vector of predicted values.

Example:

```
churn_model <- glm(churn ~ ., data = churn_data,
                   family = binomial(link = "logit")

churn_prob <- predict(churn_model, churn_test,
                      type = "response")
```

original data	1	1	1	2	2	3	4	5	5	6	6	7	7	7	7
split on feature A	1	1	1	2	2	3	4	5	5	6	6	7	7	7	7
split on feature B	1	1	1	2	2	3	4	5	5	6	6	7	7	7	7
						T_1						T_2			



Regression tree syntax

using the `rpart()` function in the `rpart` package

Building the model:

```
m <- rpart(dv ~ iv, data = mydata)
```

- `dv` is the dependent variable in the `mydata` data frame to be modeled
- `iv` is an R formula specifying the independent variables in the `mydata` data frame to use in the model
- `data` specifies the data frame in which the `dv` and `iv` variables can be found

The function will return a regression tree model object, which can be used to make predictions.

Making predictions:

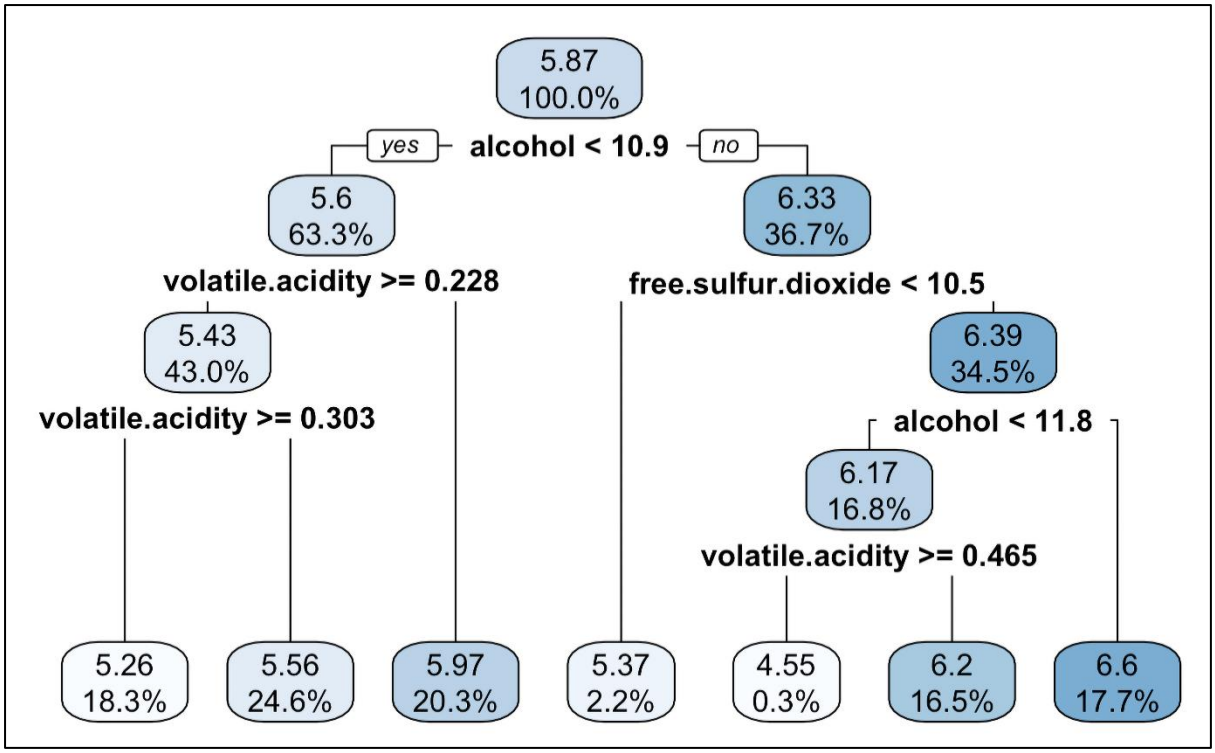
```
p <- predict(m, test, type = "vector")
```

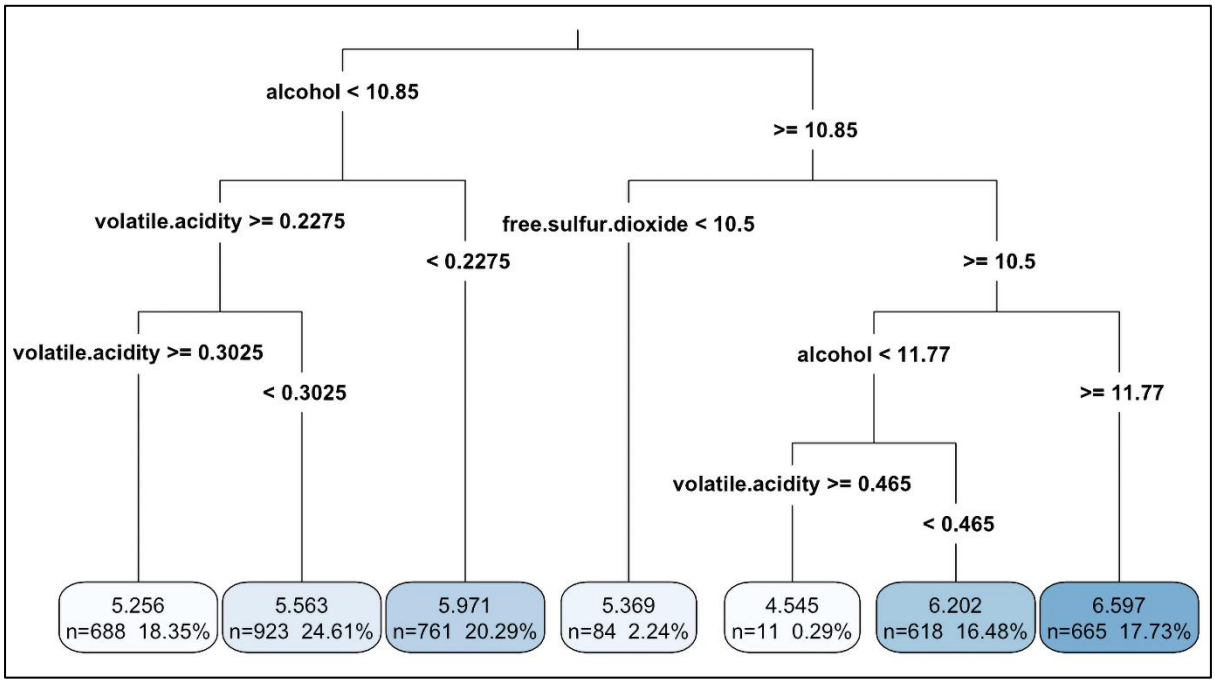
- `m` is a model trained by the `rpart()` function
- `test` is a data frame containing test data with the same features as the training data used to build the model
- `type` specifies the type of prediction to return, either `"vector"` (for predicted numeric values), `"class"` for predicted classes, or `"prob"` (for predicted class probabilities)

The function will return a vector of predictions depending on the `type` parameter.

Example:

```
wine_model <- rpart(quality ~ alcohol + sulfates,
                    data = wine_train)
wine_predictions <- predict(wine_model, wine_test)
```


5.87
100.0%
yes
alcohol < 10.9
no
5.6
63.3%
6.33
36.7%
volatile.acidity >= 0.228
free.sulfur.dioxide < 10.5
5.43
43.0%
6.39
34.5%
volatile.acidity >= 0.303
alcohol < 11.8
6.17
16.8%
volatile.acidity >= 0.465
5.26
18.3%
5.56
24.6%
5.97
20.3%
5.37
2.2%
4.55
0.3%
6.2
16.5%
6.6
17.7%


alcohol < 10.85
>= 10.85
volatile.acidity >= 0.2275
free.sulfur.dioxide < 10.5
< 0.2275
>= 10.5
volatile.acidity >= 0.3025
alcohol < 11.77
< 0.3025
>= 11.77
volatile.acidity >= 0.465
< 0.465
5.256
n=688 18.35%
5.563
n=923 24.61%
5.971
n=761 20.29%
5.369
n=84 2.24%
4.545
n=11 0.29%
6.202
n=618 16.48%
6.597
n=665 17.73%

Model tree syntax

using the `cubist()` function in the `Cubist` package

Building the model:

```
m <- cubist(train, class)
```

- `train` is a data frame or matrix containing training data
- `class` is a factor vector with the class for each row in the training data

The function will return a cubist model tree object, which can be used to make predictions.

Making predictions:

```
p <- predict(m, test)
```

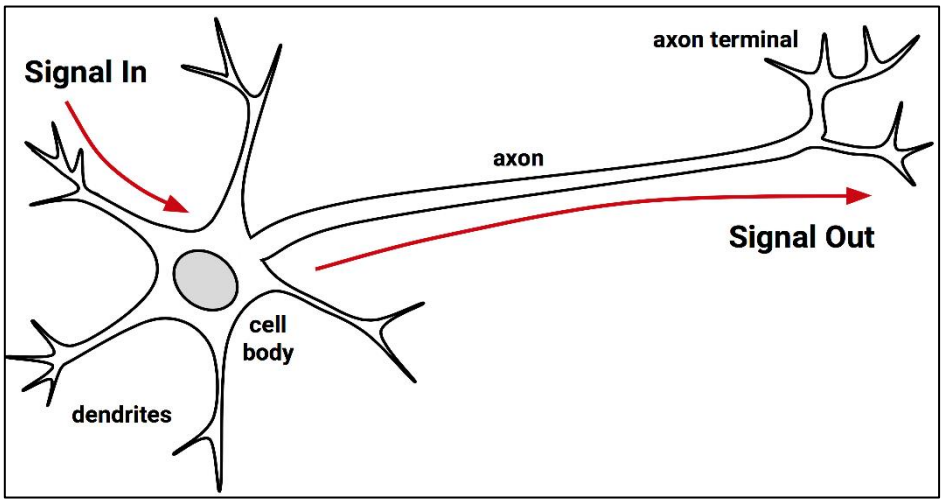
- `m` is a model trained by the `cubist()` function
- `test` is a data frame containing test data with the same features as the training data used to build the model

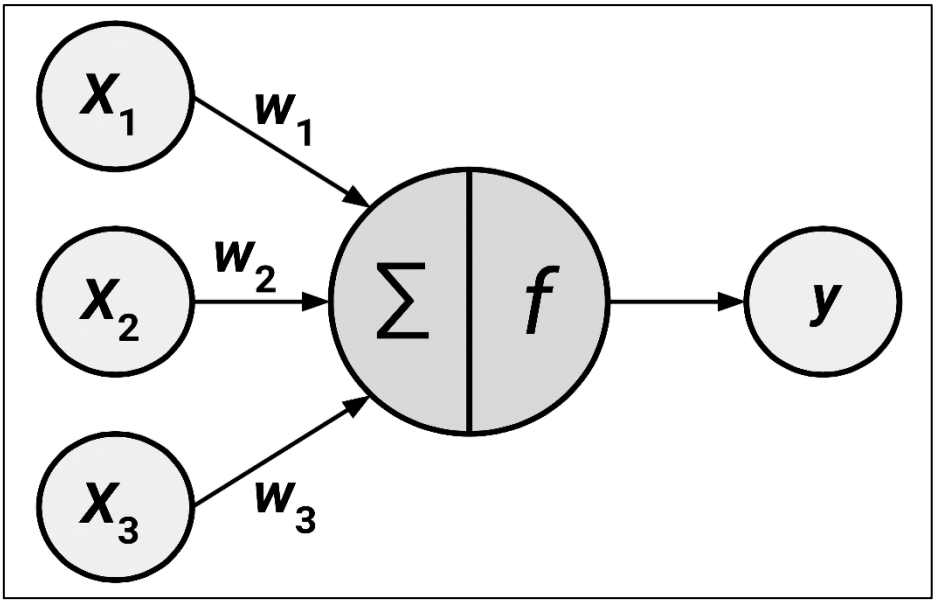
The function will return a vector of predicted numeric values.

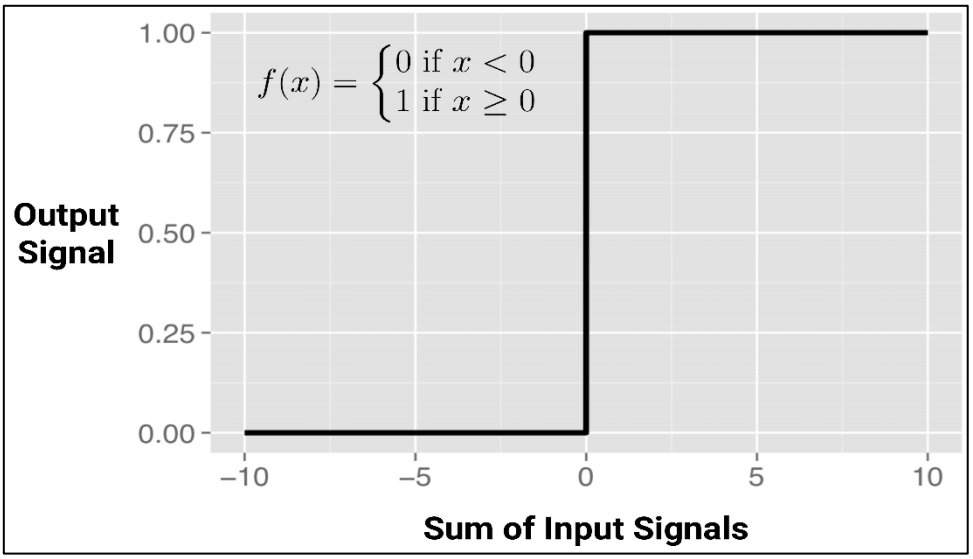
Example:

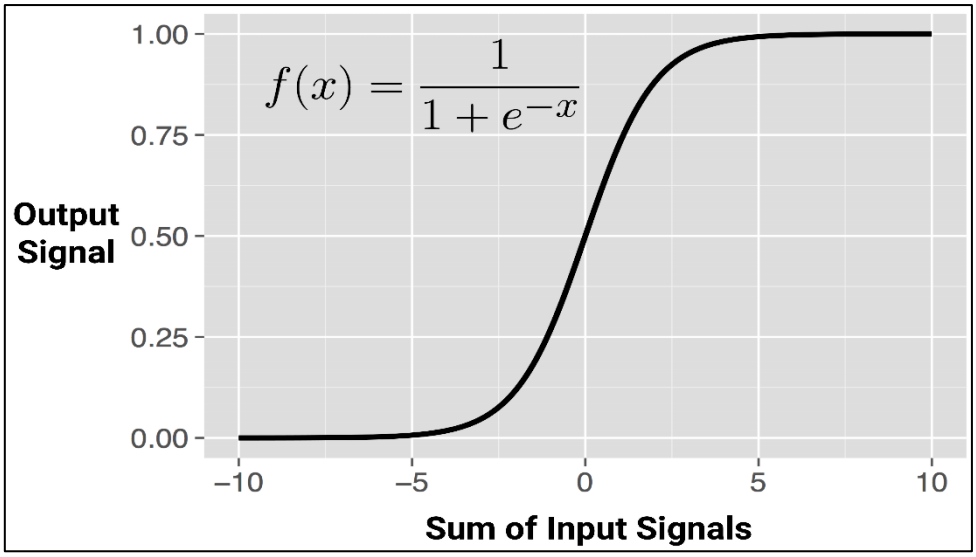
```
wine_model <- cubist(wine_train, wine_quality)
wine_predictions <- predict(wine_model, wine_test)
```

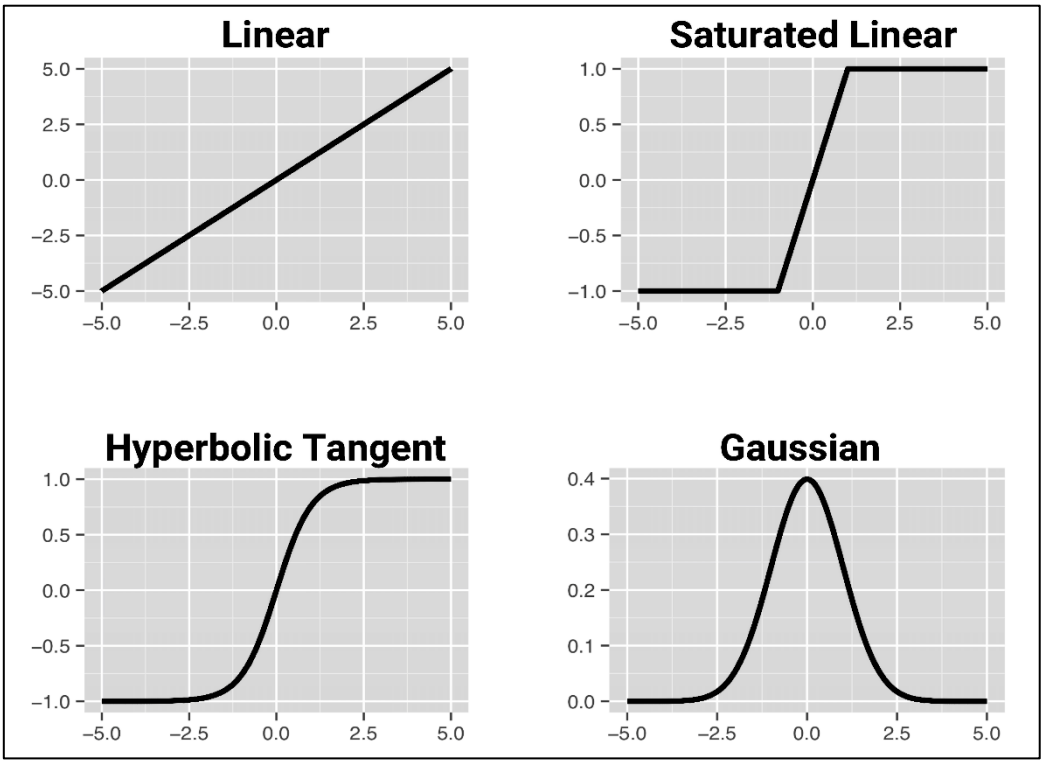
Chapter 7: Black-Box Methods – Neural Networks and Support Vector Machines

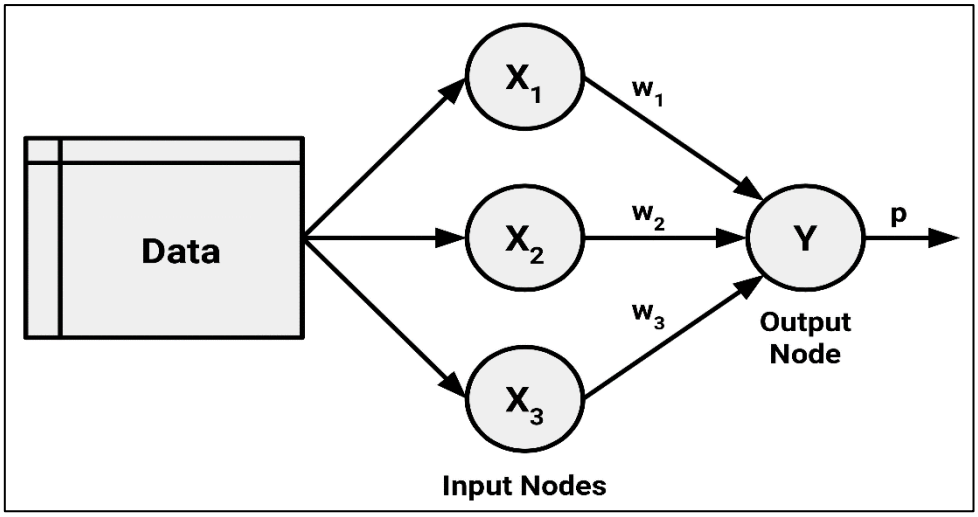


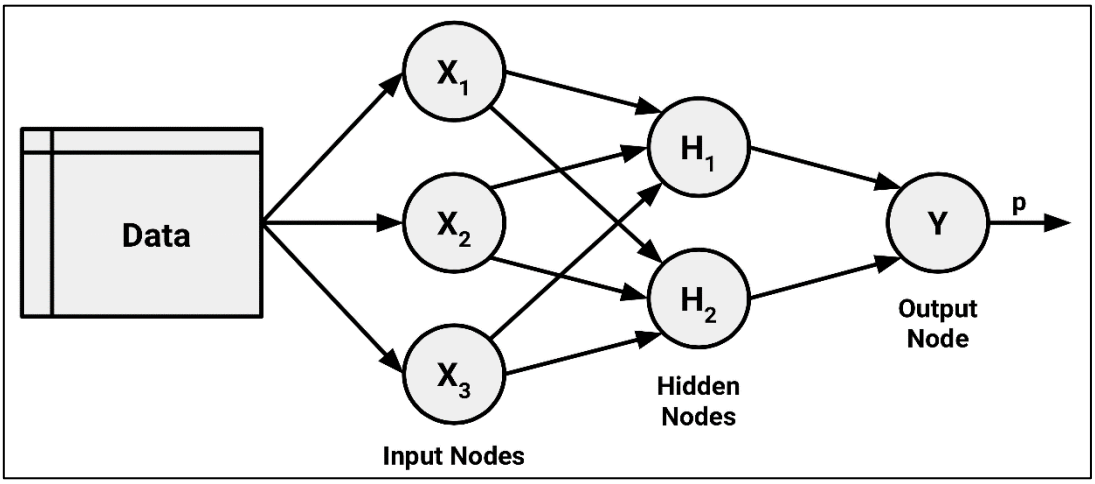


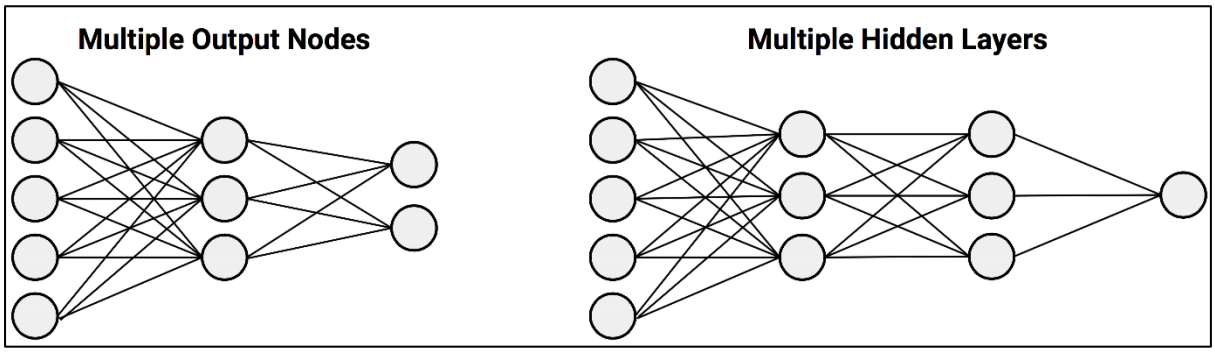


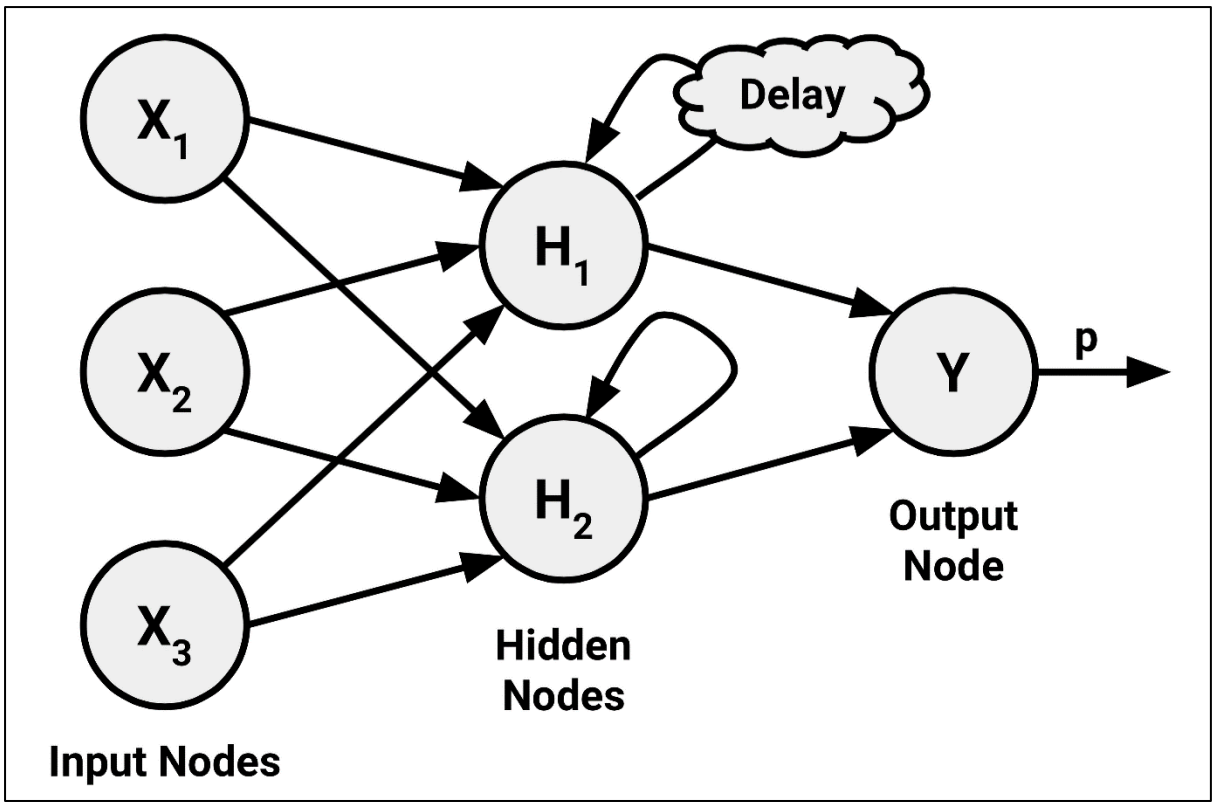

$f(x) = \frac{1}{1 + e^{-x}}$
Output Signal
1.00
0.75
0.50
0.25
0.00
−10
−5
0
5
10
Sum of Input Signals

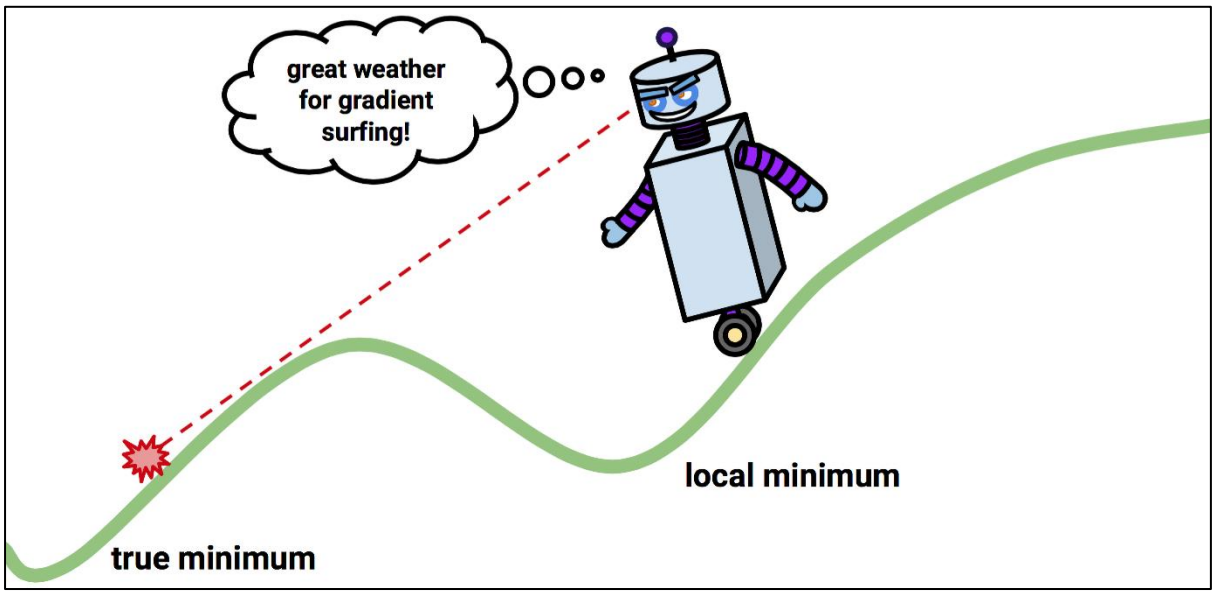

Linear
Saturated Linear
Hyperbolic Tangent
Gaussian


Data
X1
X2
X3
w1
w2
w3
Y
p
Output
Node
Input Nodes


Data
X1
X2
X3
H1
H2
Y
p
Output
Node
Hidden
Nodes
Input Nodes


Multiple Output Nodes
Multiple Hidden Layers


X1
X2
X3
Delay
H1
H2
Y
p
Output
Node
Hidden
Nodes
Input Nodes


great weather
for gradient
surfing!
local minimum
true minimum

Neural network syntax

using the `neuralnet()` function in the `neuralnet` package

Building the model:

```
m <- neuralnet(target ~ predictors, data = mydata,
               hidden = 1, act.fct = "logistic")
```

- `target` is the outcome in the `mydata` data frame to be modeled
- `predictors` is an R formula specifying the features in the `mydata` data frame to use for prediction
- `data` specifies the data frame where the `target` and `predictors` are found
- `hidden` specifies the number of neurons in the hidden layer (by default, 1)
  note: use an integer vector to specify multiple hidden layers, e.g., `c(2, 2)`
- `act.fct` specifies the activation function, either `"logistic"` or `"tanh"`
  note: a *differentiable* custom activation function can also be supplied

The function will return a neural network object that can be used to make predictions.

Making predictions:

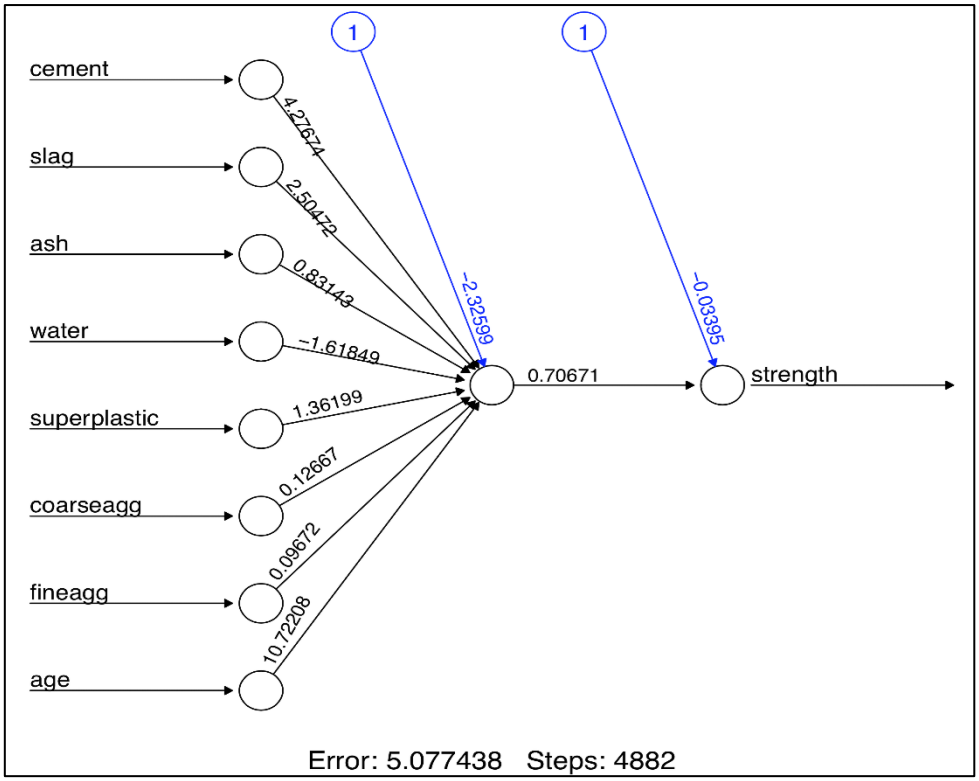
```
p <- compute(m, test)
```

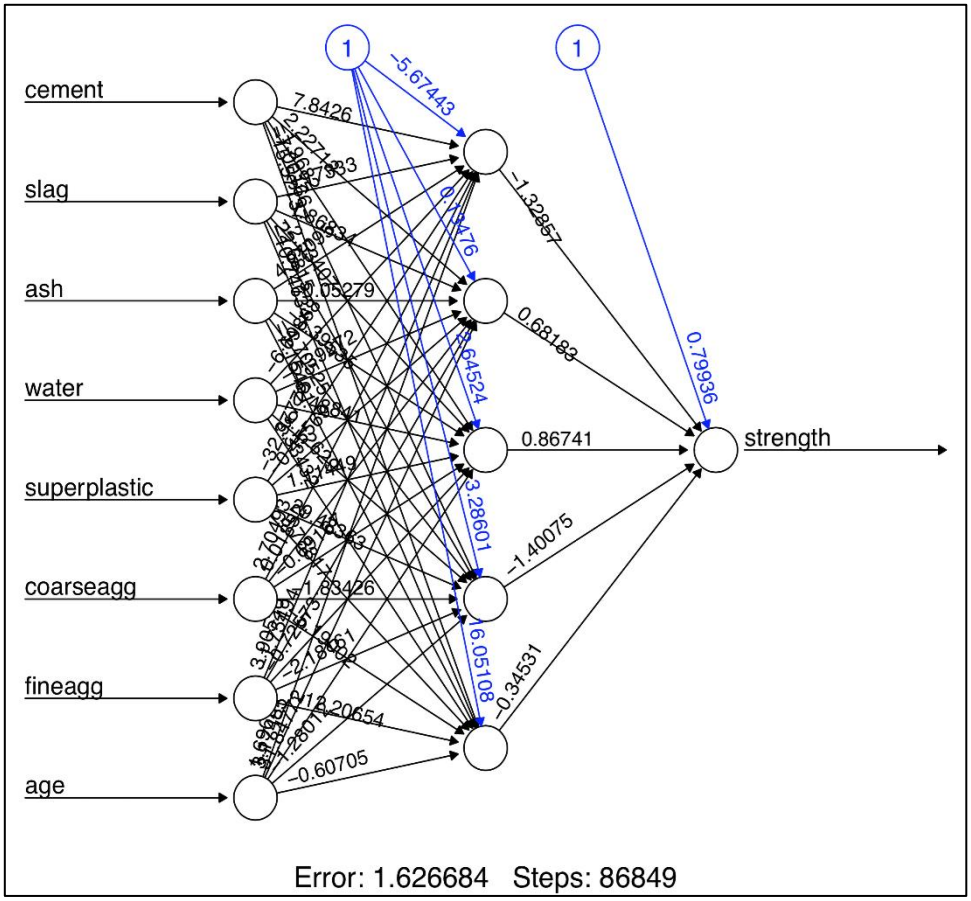
- `m` is a model trained by the `neuralnet()` function
- `test` is a data frame containing test data with the same features as the training data used to build the classifier

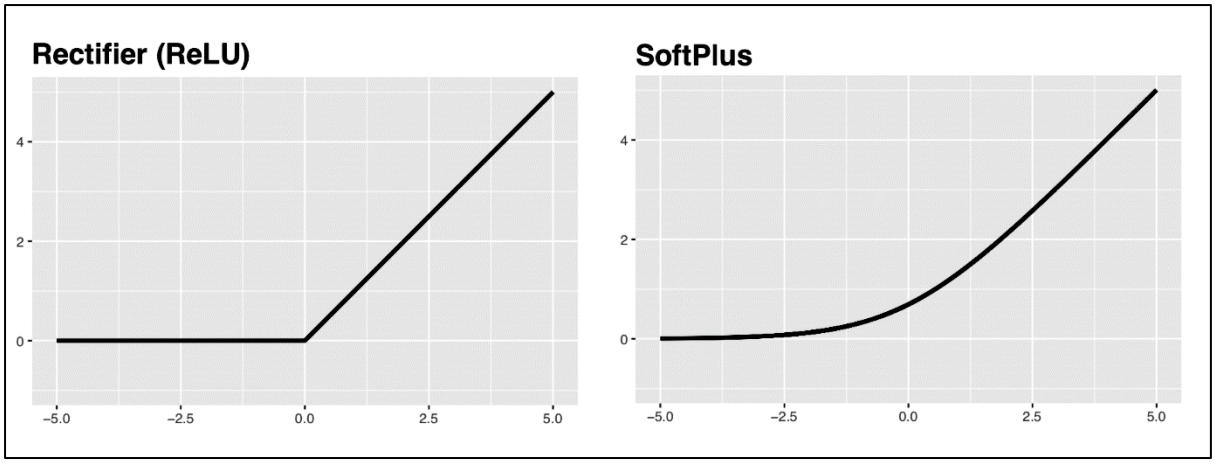
The function will return a list with two components: `$neurons`, which stores the neurons for each layer in the network, and `$net.result`, which stores the model's predicted values.

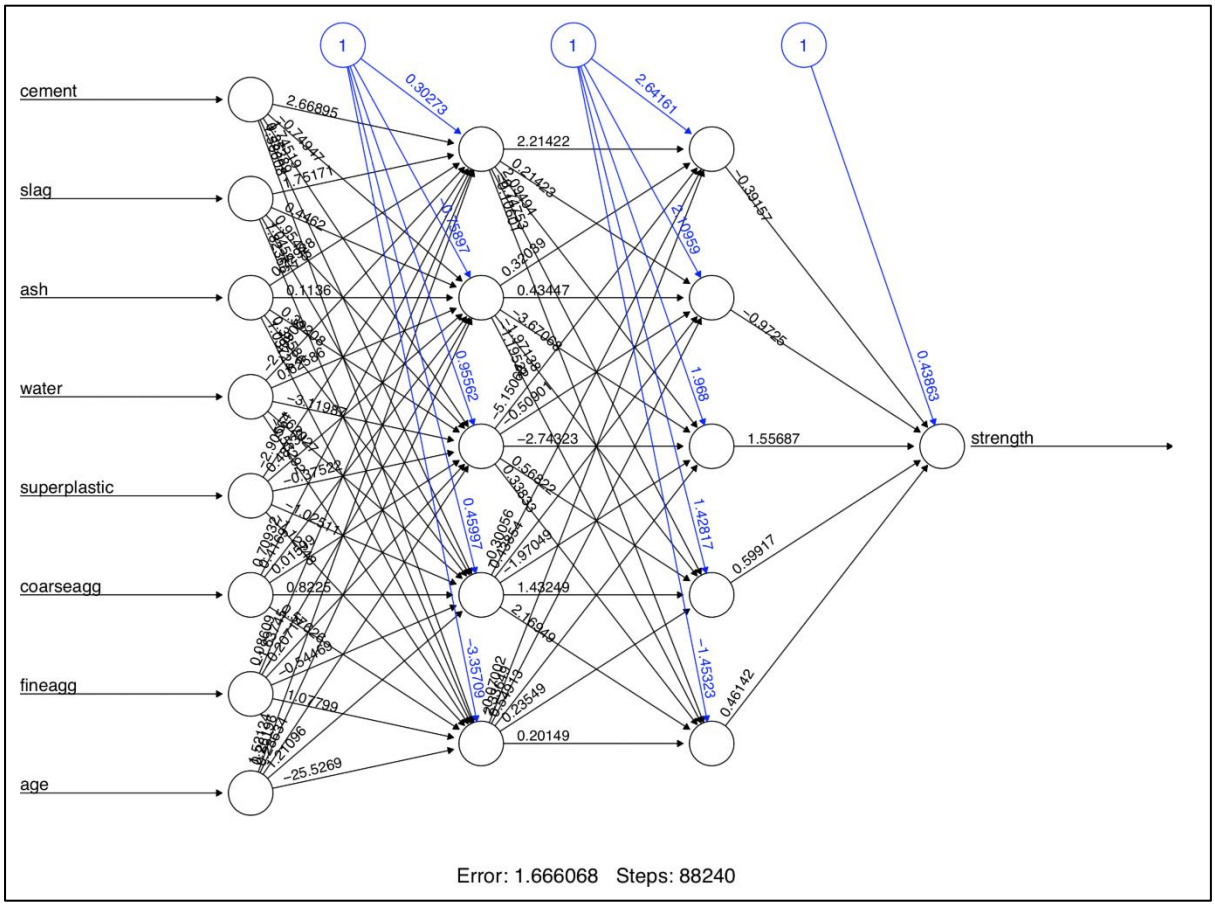
Example:

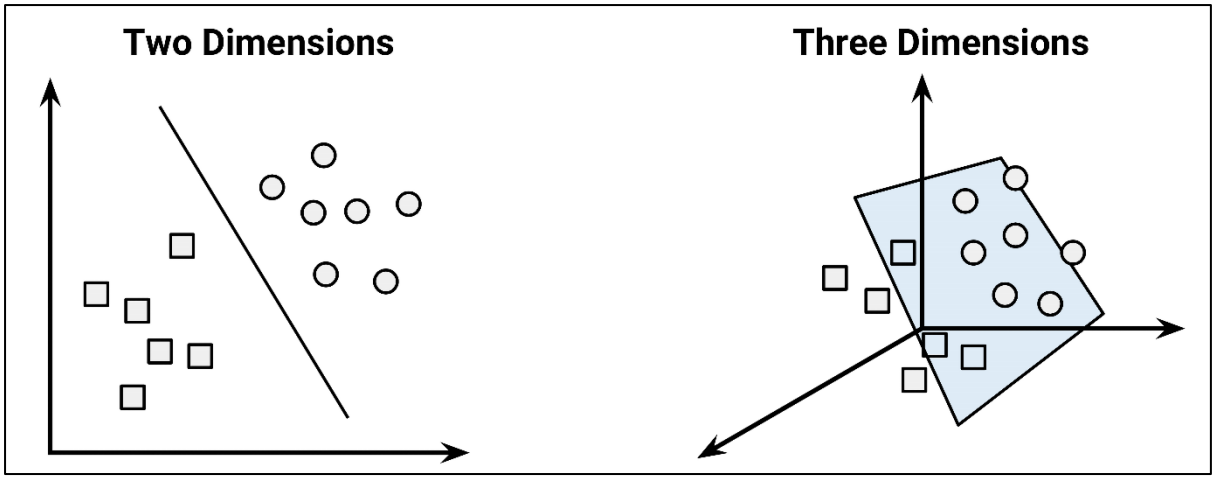
```
concrete_model <- neuralnet(strength ~ cement + slag + ash,
  data = concrete, hidden = c(5, 5), act.fct = "tanh")
model_results <- compute(concrete_model, concrete_data)
strength_predictions <- model_results$net.result
```

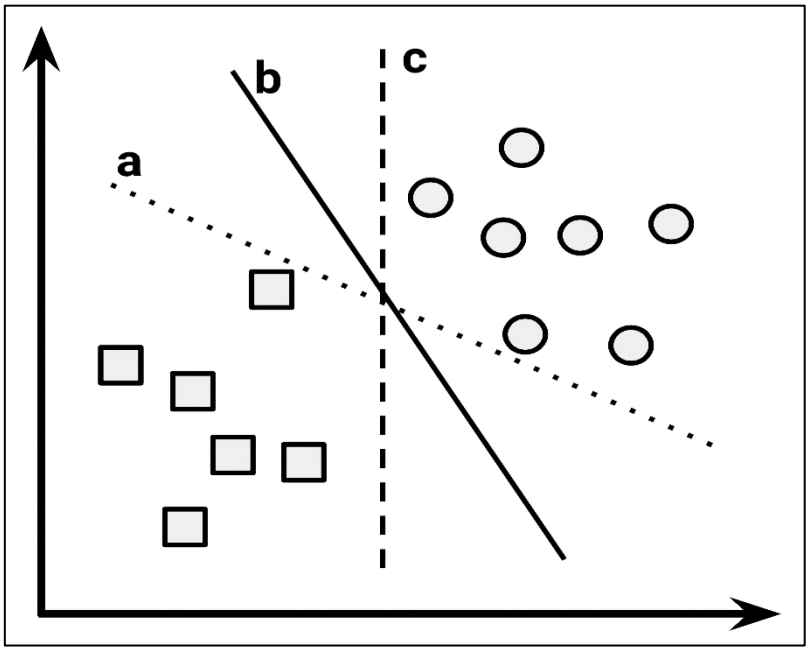

1
1
cement
slag
ash
water
superplastic
coarseagg
fineagg
age
4.27674
2.50472
0.83143
−1.61849
1.36199
0.12667
0.09672
10.72208
−2.32599
−0.03395
0.70671
strength
Error: 5.077438 Steps: 4882

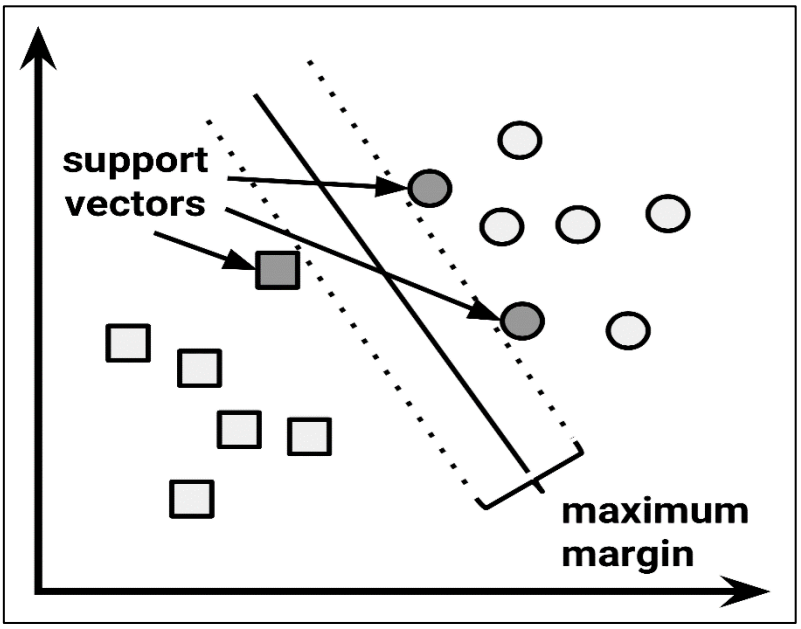

1
1
cement
slag
ash
water
superplastic
coarseagg
fineagg
age
−5.67443
−1.32857
0.68183
0.86741
−1.40075
−0.34531
0.79936
strength
Error: 1.626684 Steps: 86849

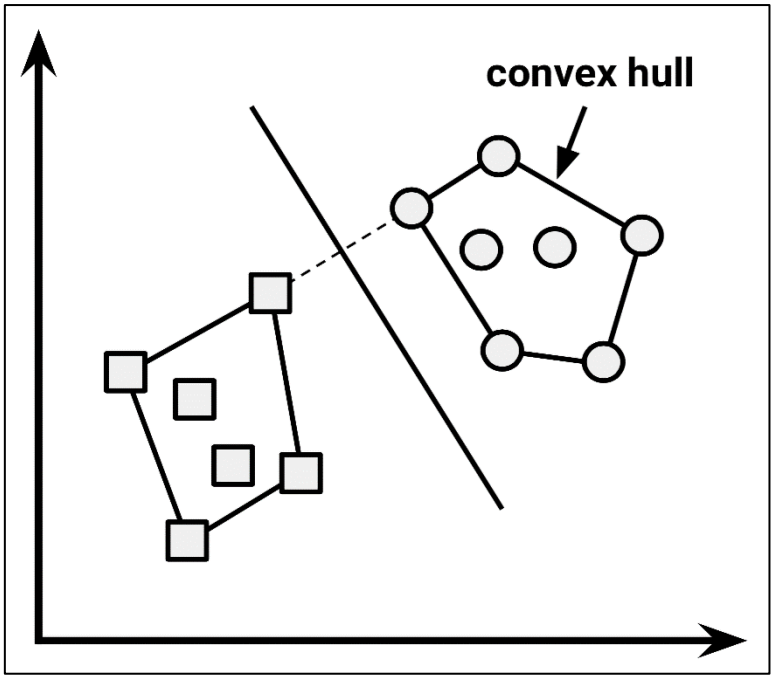

Rectifier (ReLU)
SoftPlus
4
2
0
−5.0
−2.5
0.0
2.5
5.0

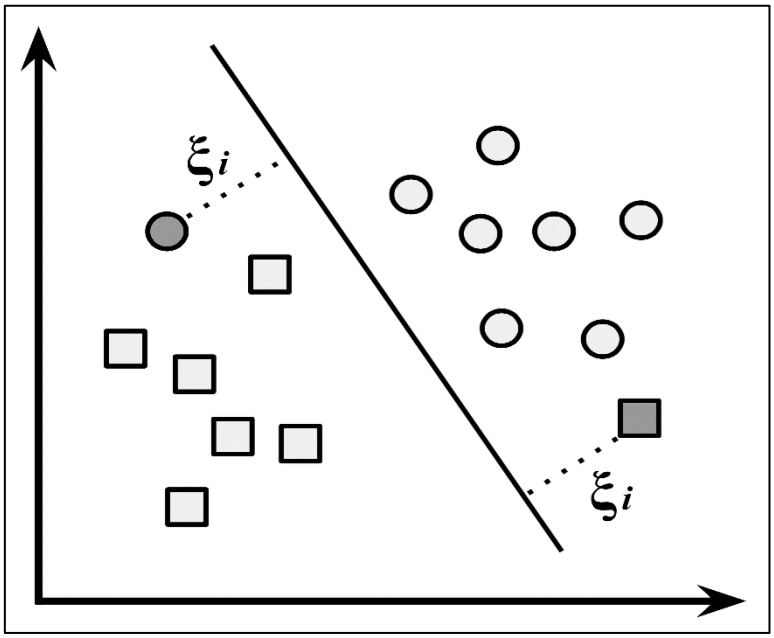

cement
slag
ash
water
superplastic
coarseagg
fineagg
age
1
2.66895
0.30273
2.64161
2.21422
−0.39157
0.43447
−0.9725
−2.74323
1.55687
strength
0.43863
1.968
1.42817
0.59917
1.43249
−1.45323
0.46142
0.20149
Error: 1.666068 Steps: 88240

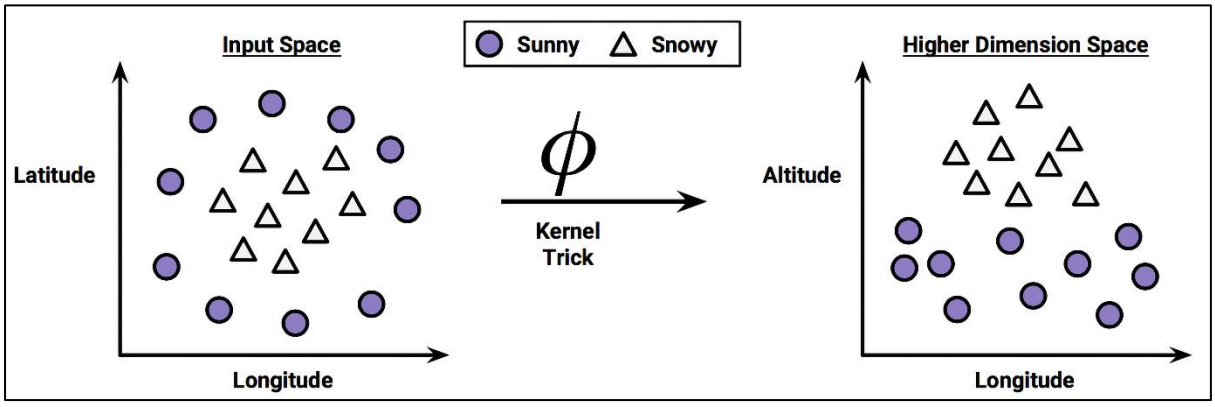

Two Dimensions
Three Dimensions

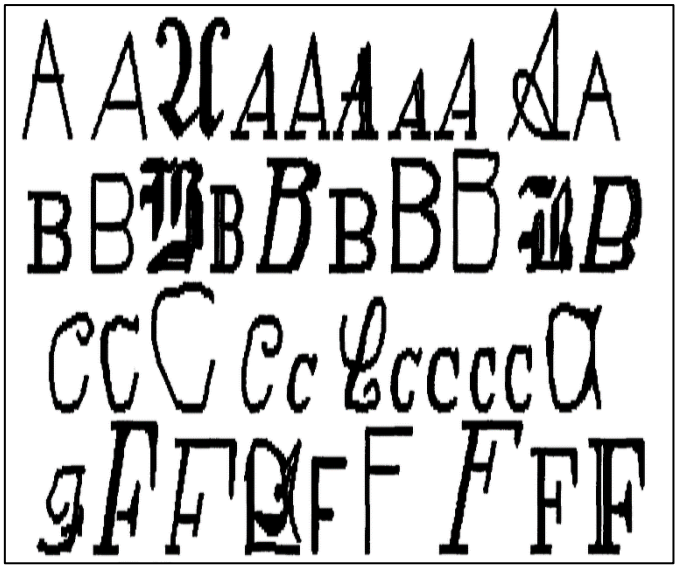

a
b
c


support
vectors
maximum
margin


convex hull


ξ_i
ξ_i


Input Space
Sunny
Snowy
Higher Dimension Space
Latitude
Longitude
ϕ
Kernel
Trick
Altitude
Longitude



Support vector machine syntax

using the `ksvm()` function in the `kernlab` package

Building the model:

```
m <- ksvm(target ~ predictors, data = mydata,
          kernel = "rbfdot", C = 1)
```

- `target` is the outcome in the `mydata` data frame to be modeled
- `predictors` is an R formula specifying the features in the `mydata` data frame to use for prediction
- `data` specifies the data frame in which the `target` and `predictors` variables can be found
- `kernel` specifies a nonlinear mapping such as `"rbfdot"` (radial basis), `"polydot"` (polynomial), `"tanhdot"` (hyperbolic tangent sigmoid), or `"vanilladot"` (linear)
- `C` is a number that specifies the cost of violating the constraints, i.e., how big of a penalty there is for the "soft margin." Larger values will result in narrower margins

The function will return a SVM object that can be used to make predictions.

Making predictions:

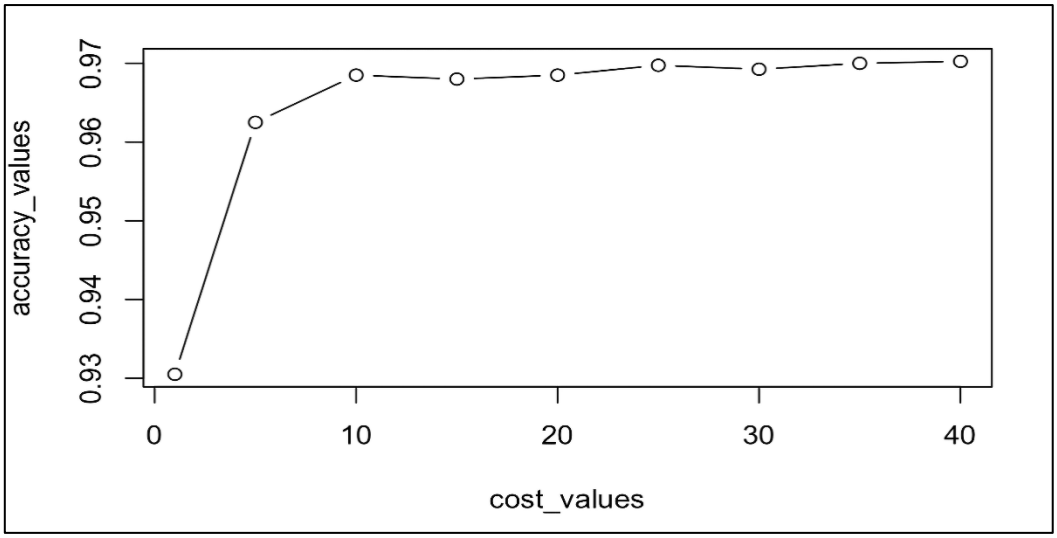
```
p <- predict(m, test, type = "response")
```

- `m` is a model trained by the `ksvm()` function
- `test` is a data frame containing test data with the same features as the training data used to build the classifier
- `type` specifies whether the predictions should be `"response"` (the predicted class) or `"probabilities"` (the predicted probability, one column per class level).

The function will return a vector (or matrix) of predicted classes (or probabilities) depending on the value of the type parameter.

Example:

```
letter_classifier <- ksvm(letter ~ ., data =
  letters_train, kernel = "vanilladot")
letter_prediction <- predict(letter_classifier,
  letters_test)
```

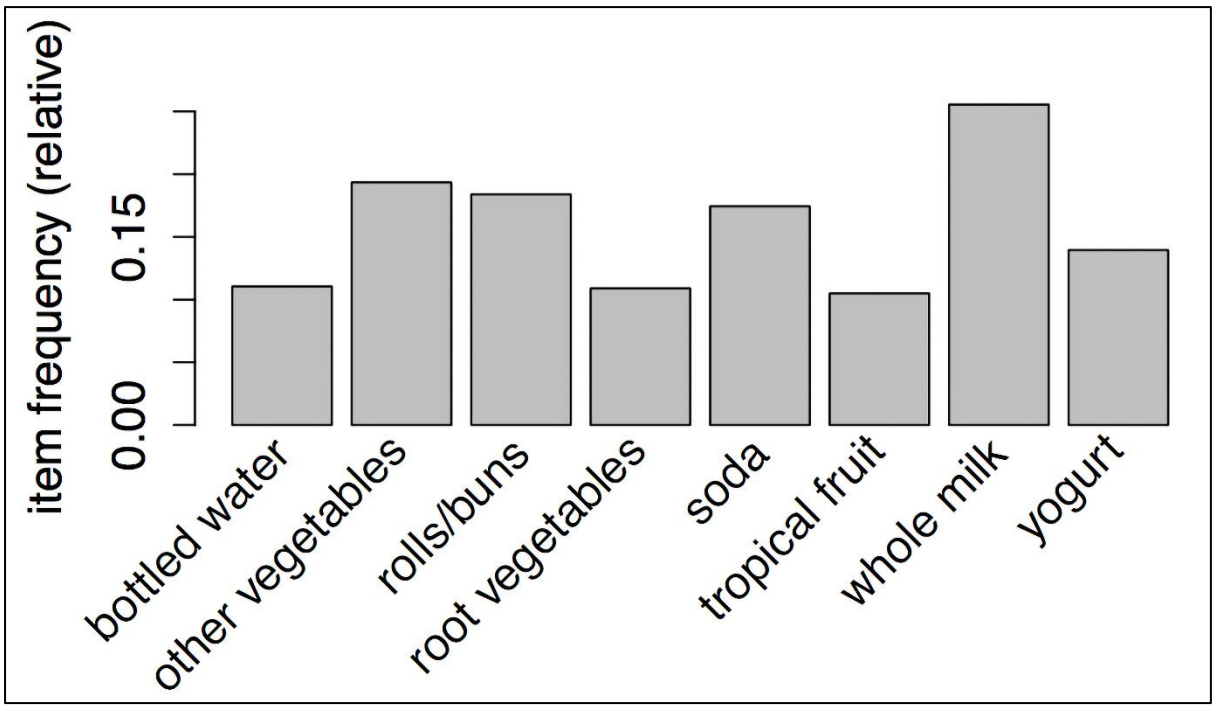

accuracy_values
0.93
0.94
0.95
0.96
0.97
0
10
20
30
40
cost_values

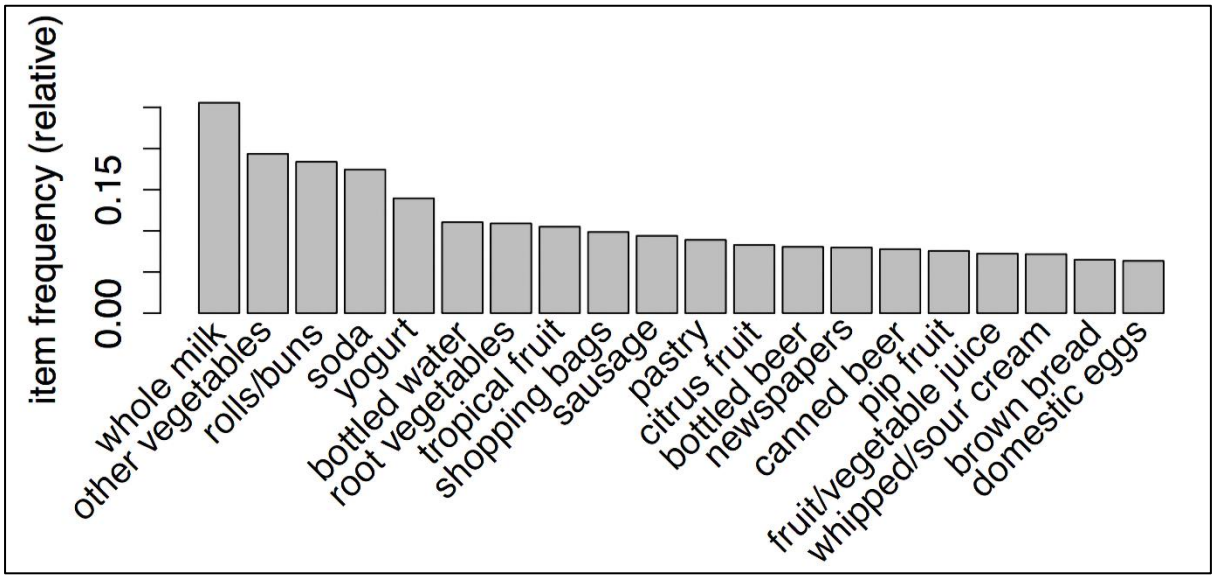
Chapter 8: Finding Patterns - Market Basket Analysis Using Association Rules

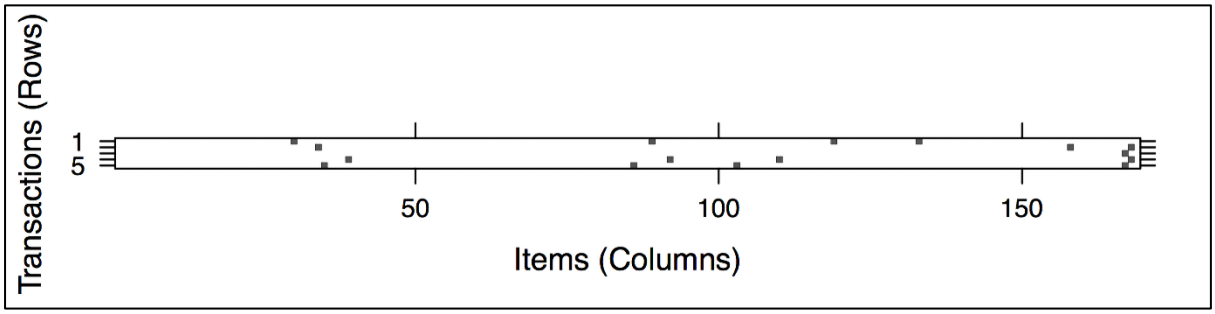
transaction ID	items purchased
1	{flowers, get-well card, soda}
2	{plush toy bear, flowers, balloons, candy bar}
3	{get-well card, candy bar, flowers}
4	{plush toy bear, balloons, soda}
5	{flowers, get-well card, soda}

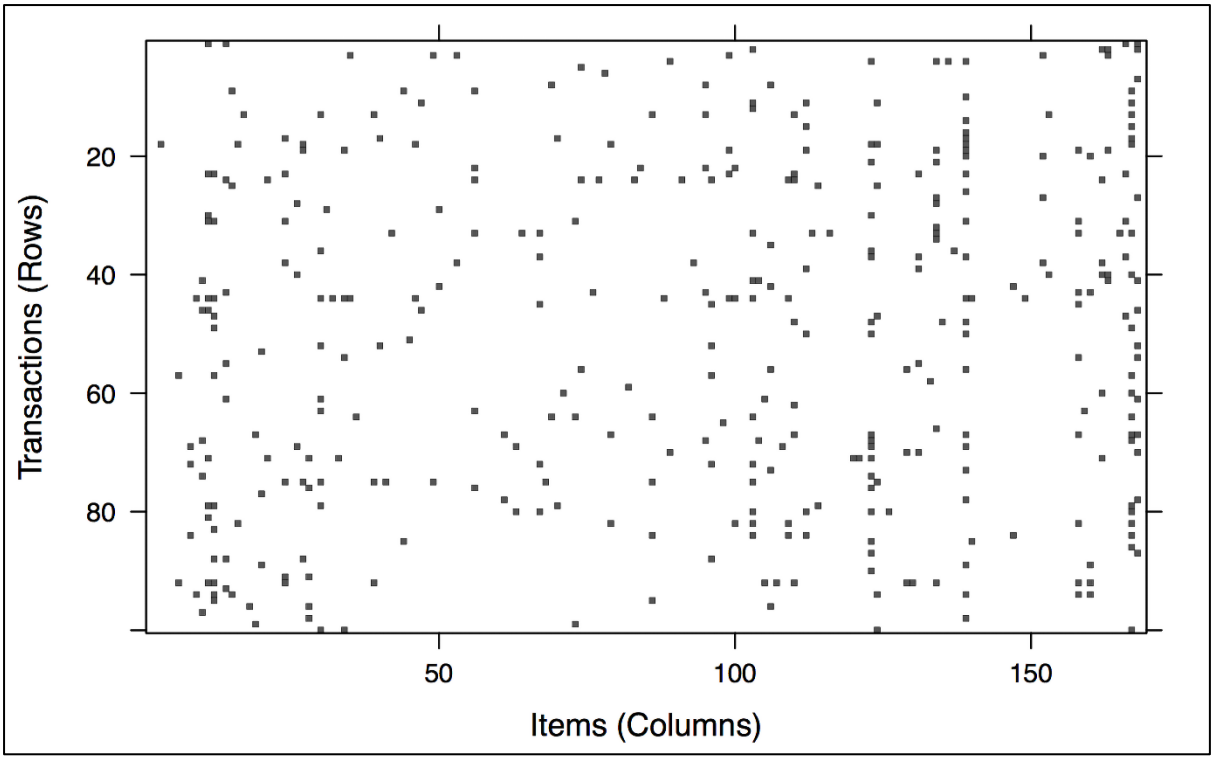
iteration	must evaluate	frequent itemsets	infrequent itemsets
1	{A}, {B}, {C}, {D}	{A}, {B}, {C}	{D}
2	{A, B}, {A, C}, {B, C} ~~{A, D}, {B, D}, {C, D}~~	{A, B}, {B, C}	{A, C}
3	~~{A, B, C}, {A, B, D}~~ ~~{A, C, D}, {B, C, D}~~		
4	~~{A, B, C, D}~~		

	V1	V2	V3	V4
1	citrus fruit	semi-finished bread	margarine	ready soups
2	tropical fruit	yogurt	coffee	
3	whole milk			
4	pip fruit	yogurt	cream cheese	meat spreads
5	other vegetables	whole milk	condensed milk	long life bakery product


item frequency (relative)
0.00
0.15
bottled water
other vegetables
rolls/buns
root vegetables
soda
tropical fruit
whole milk
yogurt


item frequency (relative)
0.00
0.15
whole milk
other vegetables
rolls/buns
soda
yogurt
bottled water
root vegetables
tropical fruit
shopping bags
sausage
pastry
citrus fruit
bottled beer
newspapers
canned beer
pip fruit
fruit/vegetable juice
whipped/sour cream
brown bread
domestic eggs


Transactions (Rows)
1
5
50
100
150
Items (Columns)



Association rule syntax

using the `apriori()` function in the `arules` package

Finding association rules:

```
myrules <- apriori(data = mydata, parameter =
  list(support = 0.1, confidence = 0.8, minlen = 1))
```

- `data` is a sparse item matrix holding transactional data
- `support` specifies the minimum required rule support
- `confidence` specifies the minimum required rule confidence
- `minlen` specifies the minimum required rule items

The function will return a rules object storing all rules that meet the minimum criteria.

Examining association rules:

```
inspect(myrules)
```

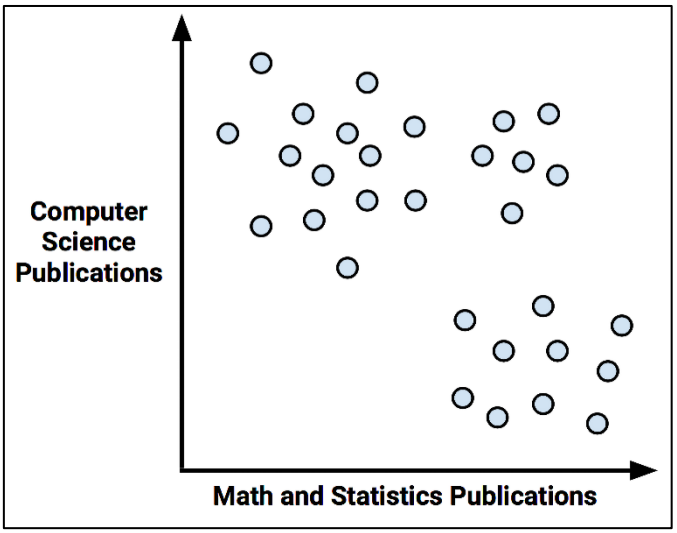
- `myrules` is a set of association rules from the `apriori()` function

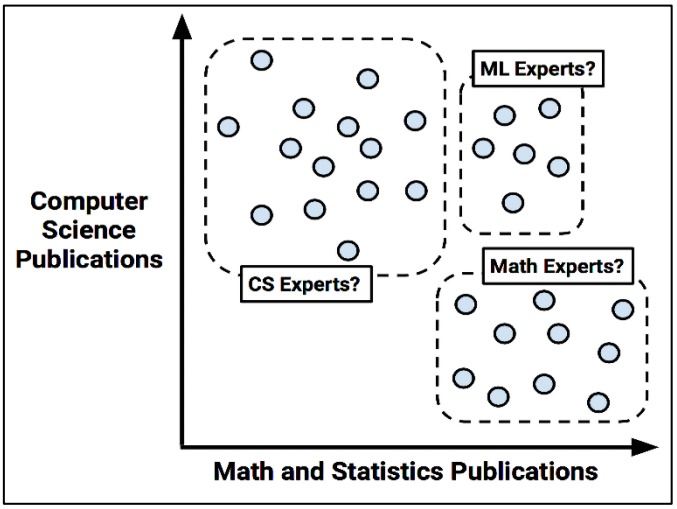
This will output the association rules to the screen. Vector operators can be used on `myrules` to choose a specific rule or rules to view.

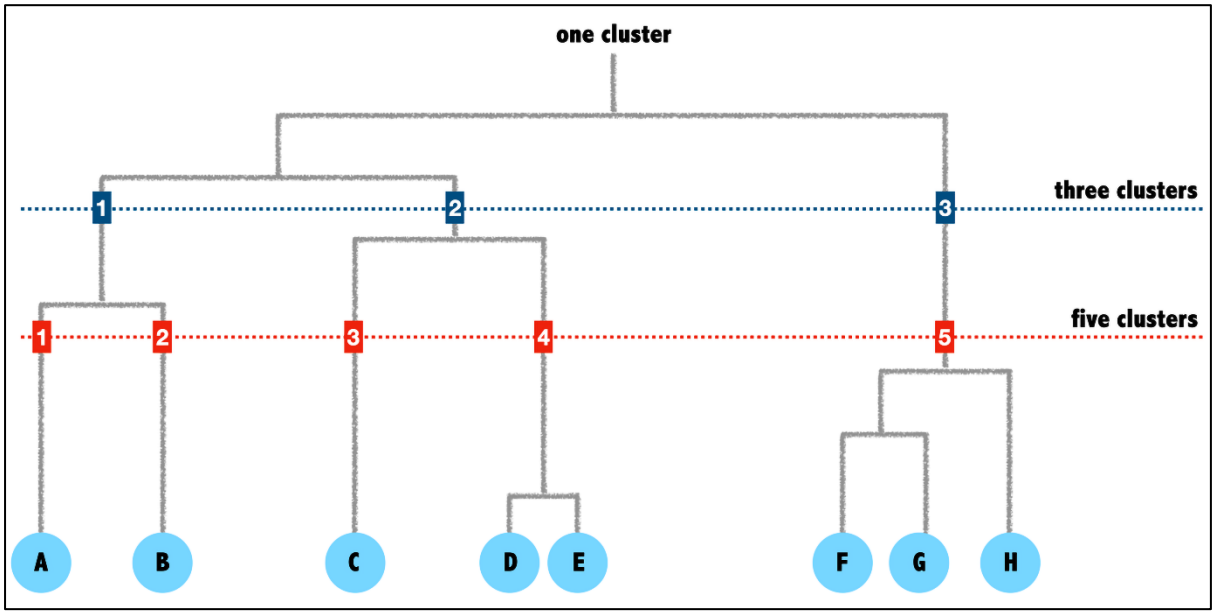
Example:

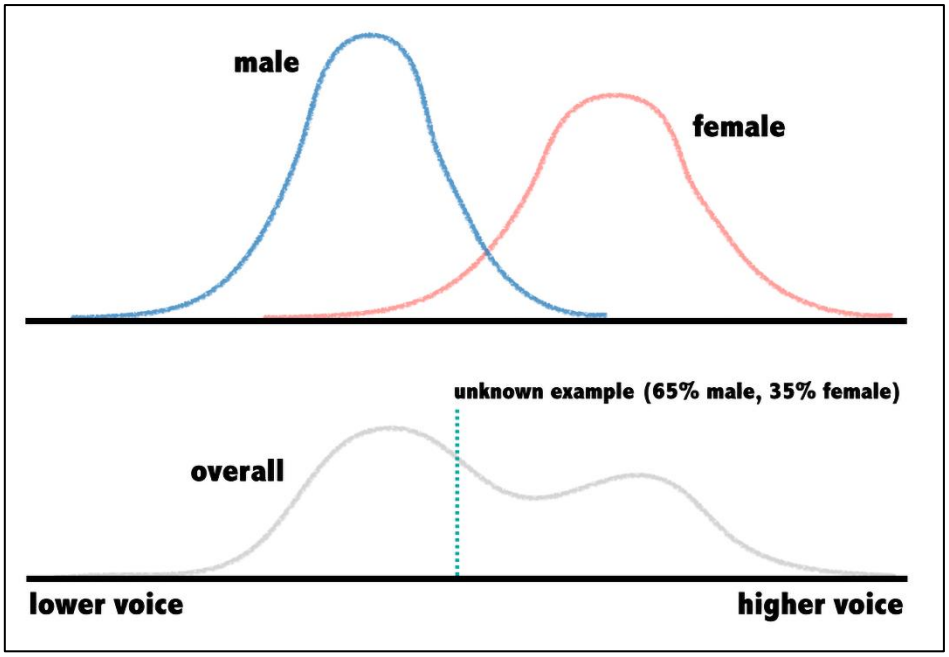
```
groceryrules <- apriori(groceries, parameter =
 list(support = 0.01, confidence = 0.25, minlen = 2))
inspect(groceryrules[1:3])
```

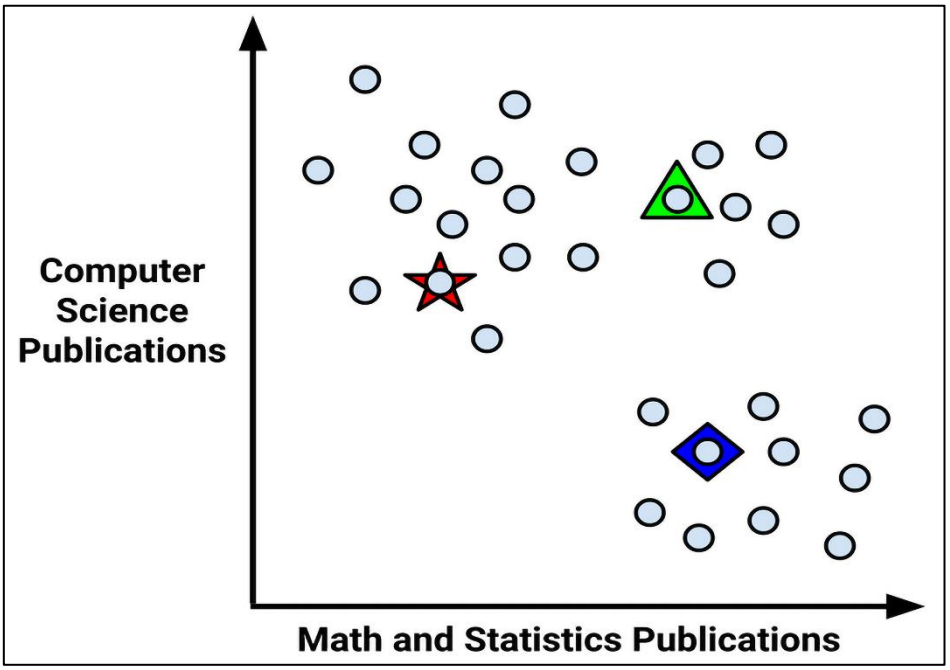
Chapter 9: Finding Groups of Data - Clustering with k-means

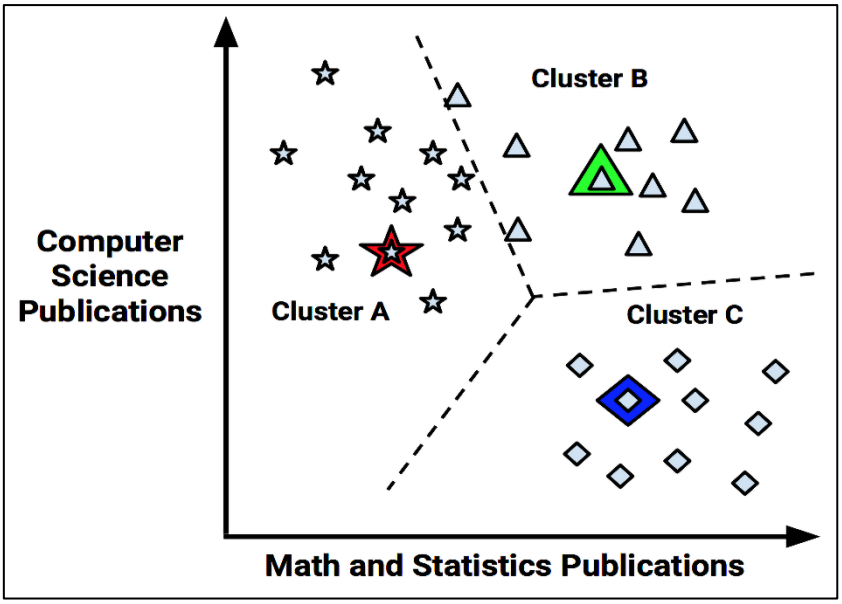


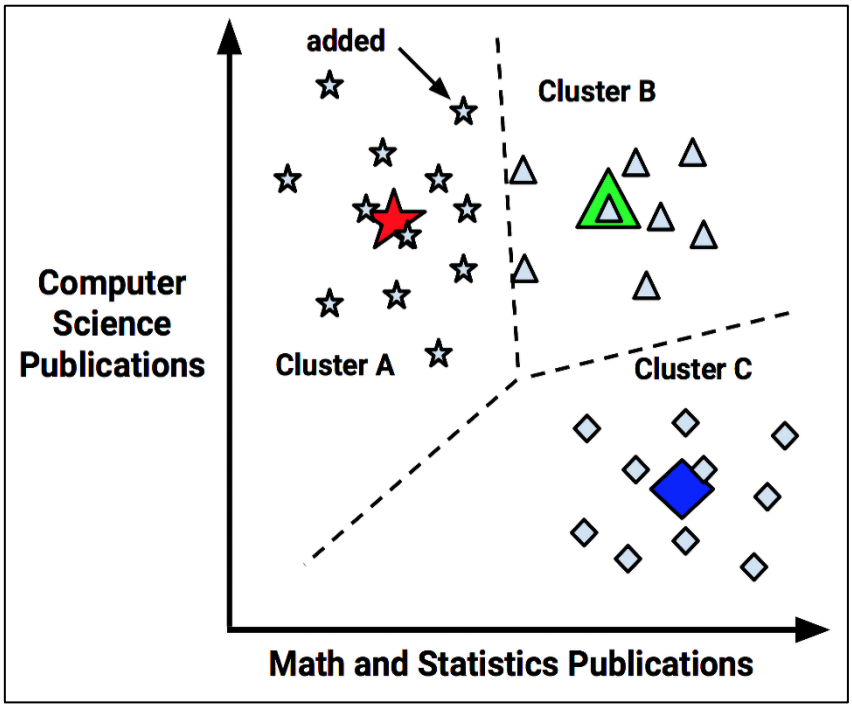


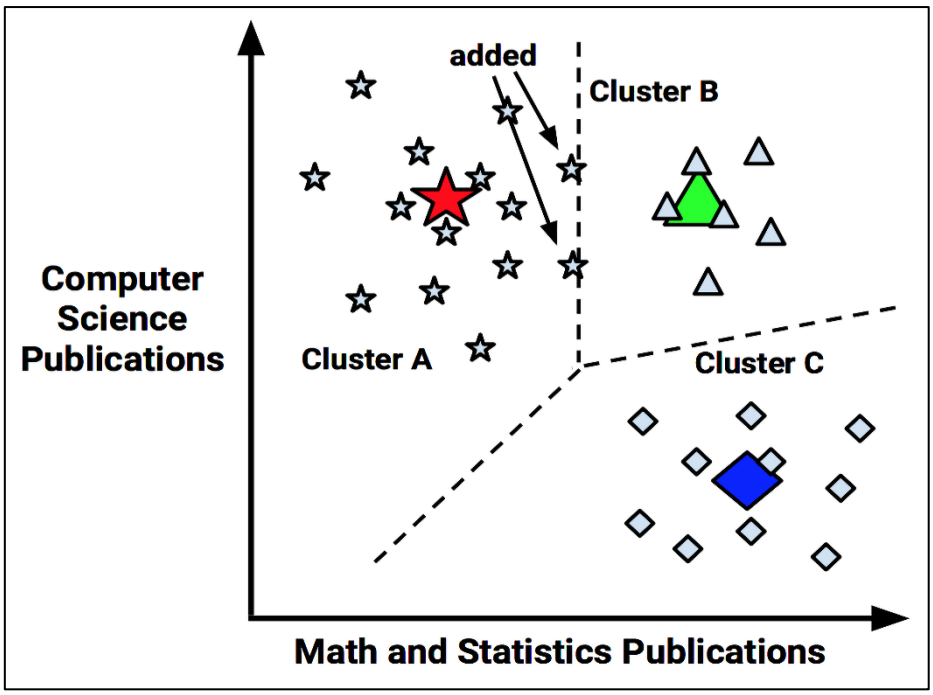


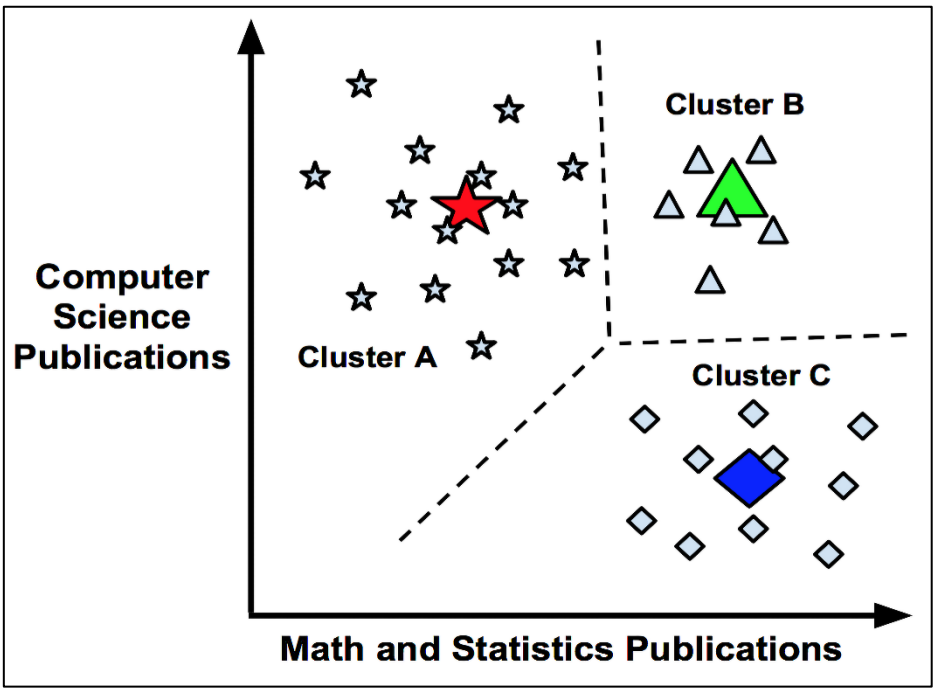

male
female
unknown example (65% male, 35% female)
overall
lower voice
higher voice

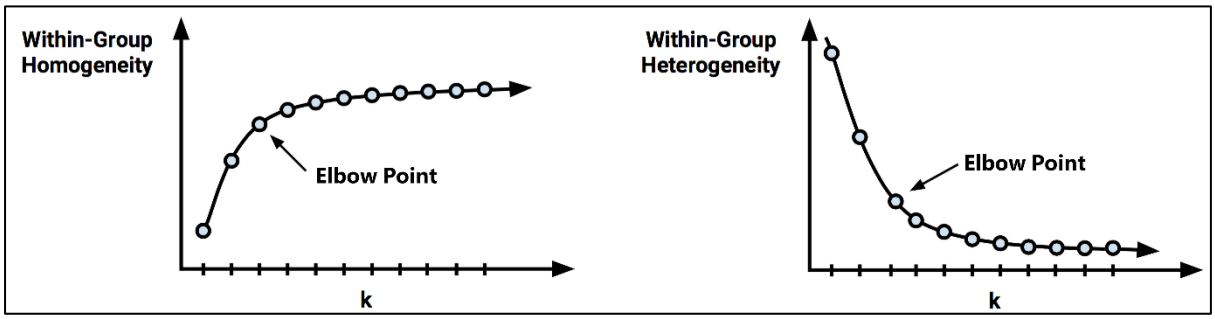

Computer Science Publications
Math and Statistics Publications


Cluster B
Computer Science Publications
Cluster A
Cluster C
Math and Statistics Publications


added
Cluster B
Computer Science Publications
Cluster A
Cluster C
Math and Statistics Publications


added
Cluster B
Computer Science Publications
Cluster A
Cluster C
Math and Statistics Publications


Cluster B
Computer Science Publications
Cluster A
Cluster C
Math and Statistics Publications


Within-Group Homogeneity
Elbow Point
k
Within-Group Heterogeneity
Elbow Point
k

Clustering syntax

using the `kmeans()` function in the `stats` package

Finding clusters:

```
myclusters <- kmeans(mydata, k)
```

- `mydata` is a matrix or data frame with the examples to be clustered
- `k` specifies the desired number of clusters

The function will return a cluster object, which stores information about the clusters.

Examining clusters:

- `myclusters$cluster` is a vector of cluster assignments from the `kmeans()` function
- `myclusters$centers` is a matrix indicating the mean values for each feature and cluster combination
- `myclusters$size` lists the number of examples assigned to each cluster

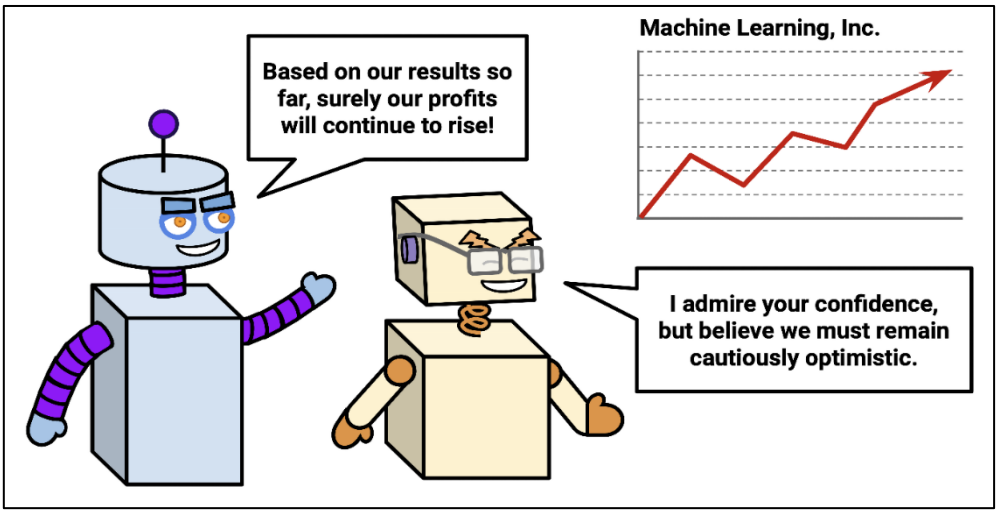
Example:

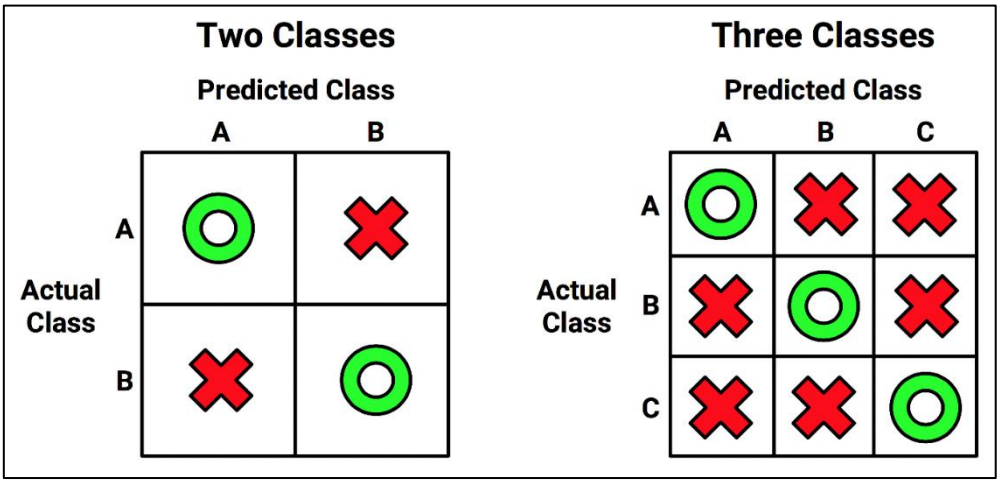
```
teen_clusters <- kmeans(teens, 5)
teens$cluster_id <- teen_clusters$cluster
```

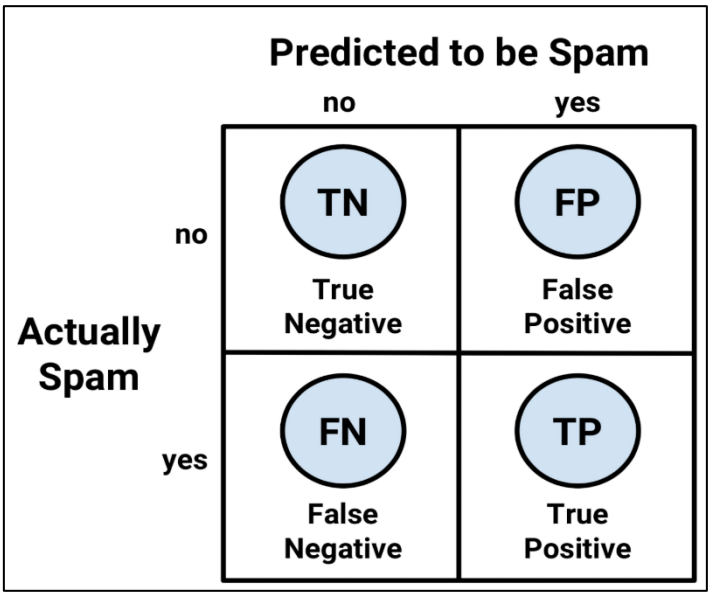
```
> teen_clusters$centers
    basketball    football      soccer   softball  volleyball    swimming
1  0.362160730  0.37985213  0.13734997  0.1272107  0.09247518  0.26180286
2 -0.094426312  0.06691768 -0.09956009 -0.0379725 -0.07286202  0.04578401
3  0.003980104  0.09524062  0.05342109 -0.0496864 -0.01459648  0.32944934
4  1.372334818  1.19570343  0.55621097  1.1304527  1.07177211  0.08513210
5 -0.186822093 -0.18729427 -0.08331351 -0.1368072 -0.13344819 -0.08650052
  cheerleading    baseball      tennis      sports         cute          sex
1    0.2159945  0.25312305  0.11991682  0.77040675  0.475265034  2.043945661
2   -0.1070370 -0.11182941  0.04027335 -0.10638613 -0.027044898 -0.042725567
3    0.5142451 -0.04933628  0.06703386 -0.05435093  0.796948359 -0.003156716
4    0.0400367  1.09279737  0.13887184  1.08316097 -0.005291962 -0.033193640
5   -0.1092056 -0.13616893 -0.03683671 -0.15903307 -0.171452198 -0.092301138
         sexy          hot      kissed       dance        band   marching
1  0.547956598  0.314845390  3.02610259  0.455501275  0.39009330 -0.0105463
2 -0.027913348 -0.035027022 -0.04581067  0.050772118  4.09723438  5.2196105
3  0.266741598  0.623263396 -0.01284964  0.650572336 -0.03301257 -0.1131486
4  0.003036966  0.009046774 -0.08755418 -0.001993853 -0.07317758 -0.1039509
5 -0.076149916 -0.132614350 -0.13080557 -0.145524147 -0.11740538 -0.1104553
```

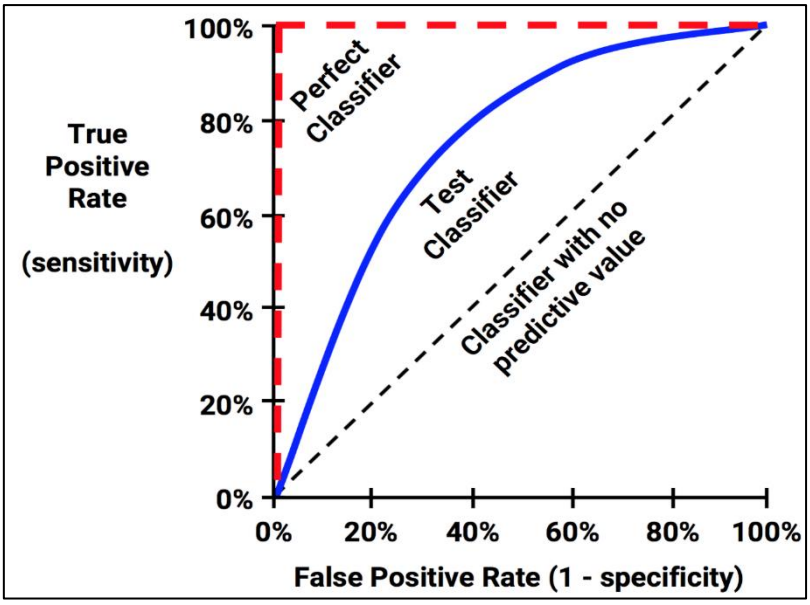
Cluster 1 (N = 1,038)	Cluster 2 (N = 601)	Cluster 3 (N = 4,066)	Cluster 4 (N = 2,696)	Cluster 5 (N = 21,599)
sex sexy kissed music rock god hair clothes die death drunk drugs	band marching music	cheerleading cute hot dance church Jesus Bible dress mall shopping Hollister Abercrombie	basketball football soccer softball volleyball baseball tennis sports	???
Criminals	Brains	Princesses	Athletes	Basket Cases

Chapter 10: Evaluating Model Performance

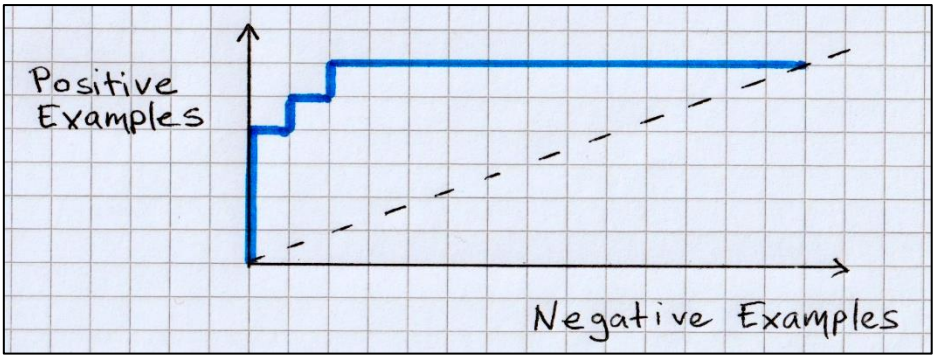


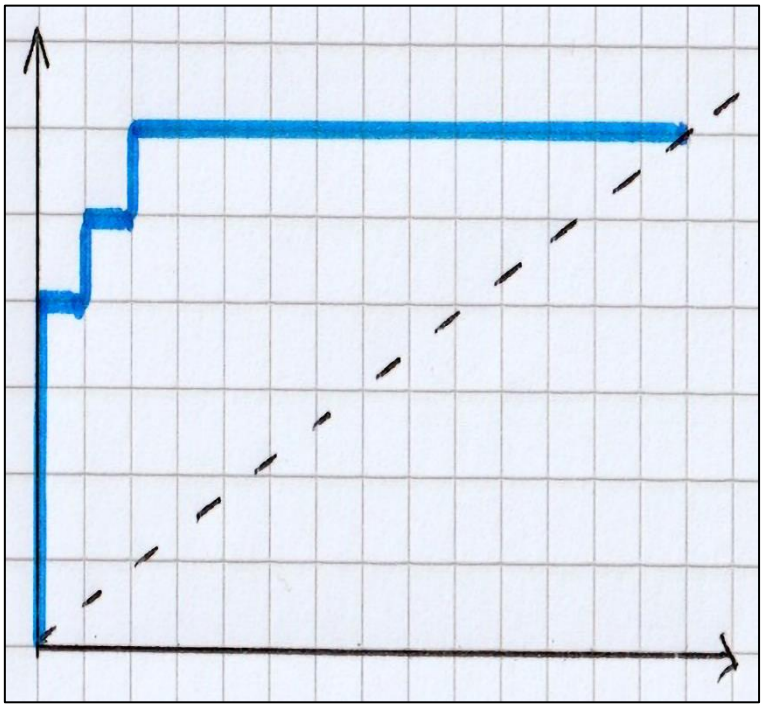




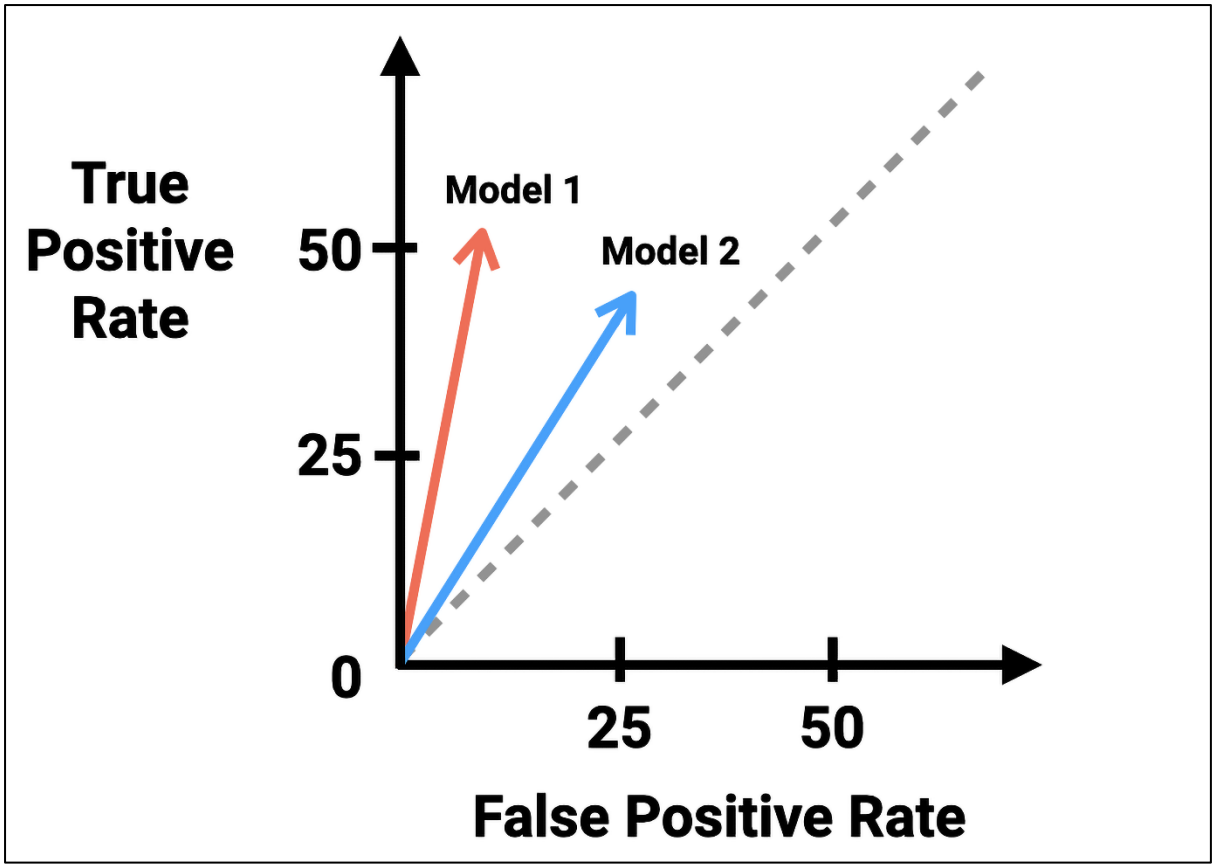


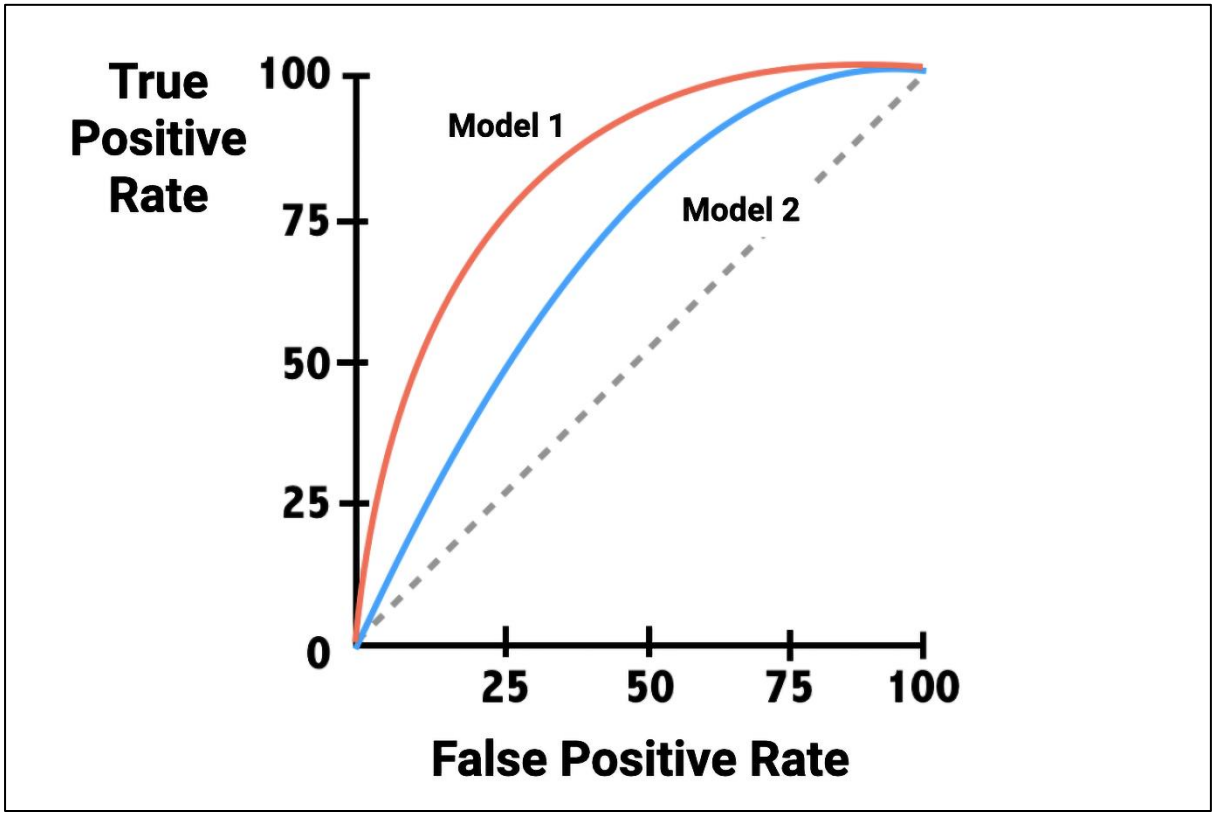
Estimated Spam Probability	Actual Message Type
0.95	spam
0.90	spam
0.75	spam
0.70	spam
0.60	ham
0.55	spam
0.51	ham
0.49	spam
0.38	ham
0.35	ham
0.30	ham
0.25	ham
0.21	ham
0.20	ham
0.19	ham
0.19	ham
0.18	ham
0.15	ham
0.11	ham
0.10	ham

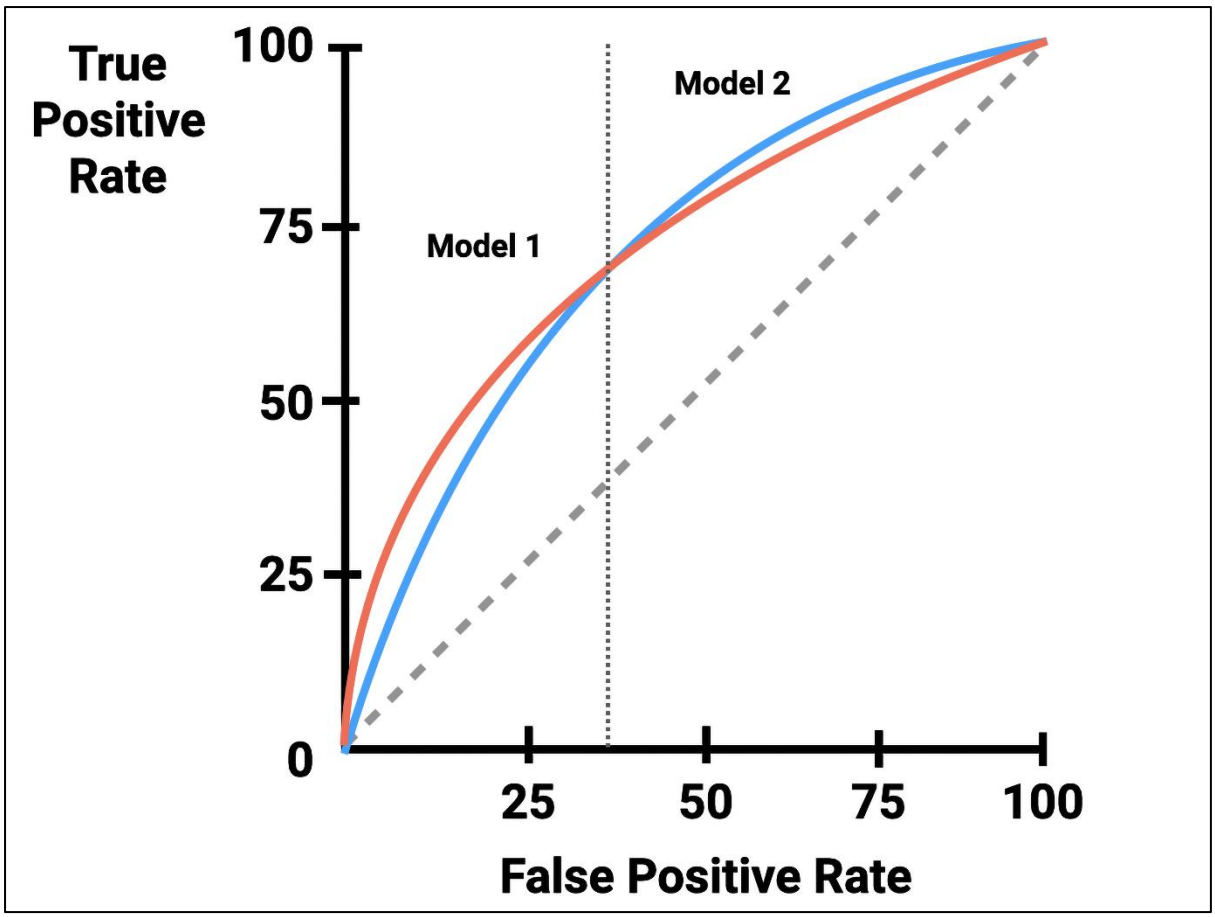


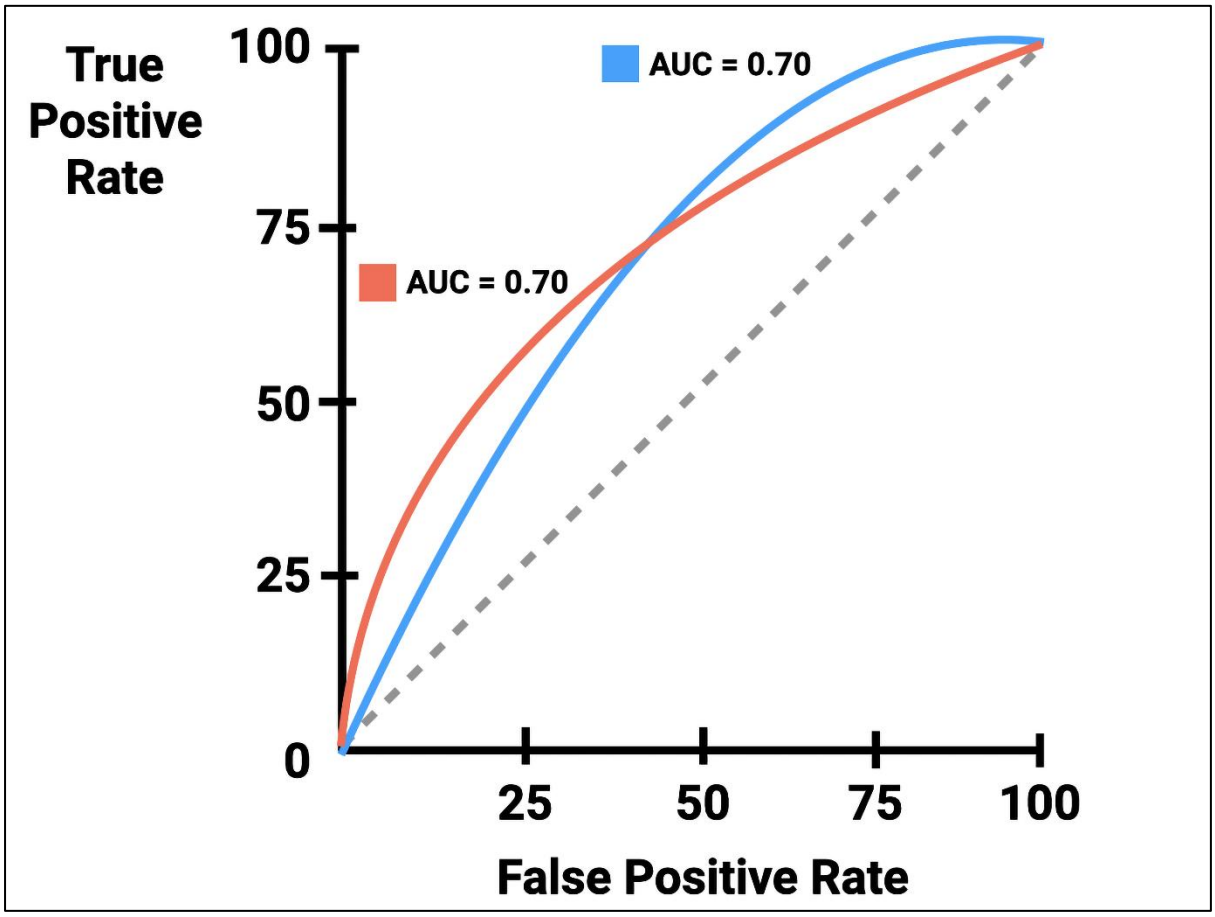


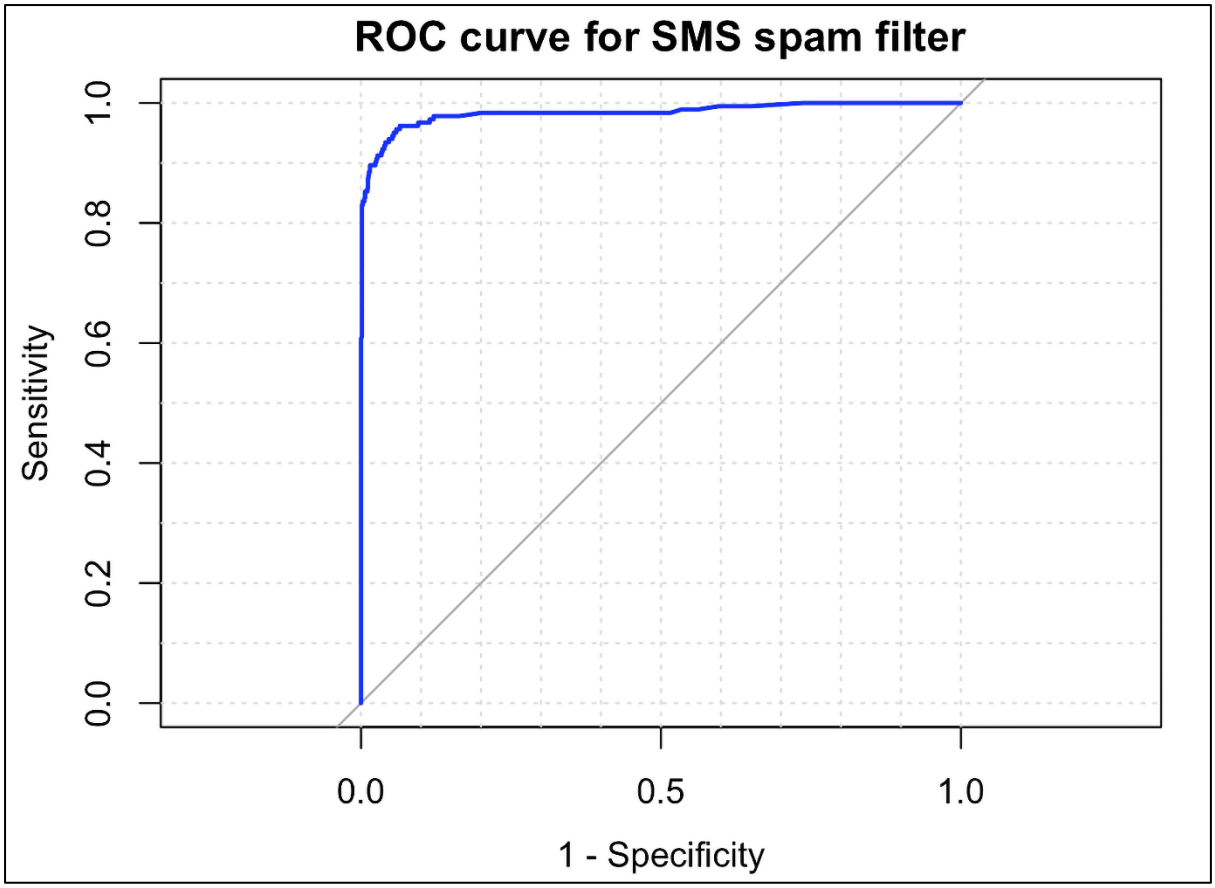
Example Spam Model						
Predicted Prob. of Spam	Actual Type	True Positives	True Negatives	TP Rate (Sensitivity)	TN Rate (Specificity)	FP Rate (1 - Specificity)
0.95	spam	1	14	16.7%	100.0%	0.0%
0.90	spam	2	14	33.3%	100.0%	0.0%
0.75	spam	3	14	50.0%	100.0%	0.0%
0.70	spam	4	14	66.7%	100.0%	0.0%
0.60	ham	4	13	66.7%	92.9%	7.1%
0.55	spam	5	13	83.3%	92.9%	7.1%
0.51	ham	5	12	83.3%	85.7%	14.3%
0.49	spam	6	12	100.0%	85.7%	14.3%
0.38	ham	6	11	100.0%	78.6%	21.4%
0.35	ham	6	10	100.0%	71.4%	28.6%
0.30	ham	6	9	100.0%	64.3%	35.7%
0.25	ham	6	8	100.0%	57.1%	42.9%
0.21	ham	6	7	100.0%	50.0%	50.0%
0.20	ham	6	6	100.0%	42.9%	57.1%
0.19	ham	6	5	100.0%	35.7%	64.3%
0.19	ham	6	4	100.0%	28.6%	71.4%
0.18	ham	6	3	100.0%	21.4%	78.6%
0.15	ham	6	2	100.0%	14.3%	85.7%
0.11	ham	6	1	100.0%	7.1%	92.9%
0.10	ham	6	0	100.0%	0.0%	100.0%

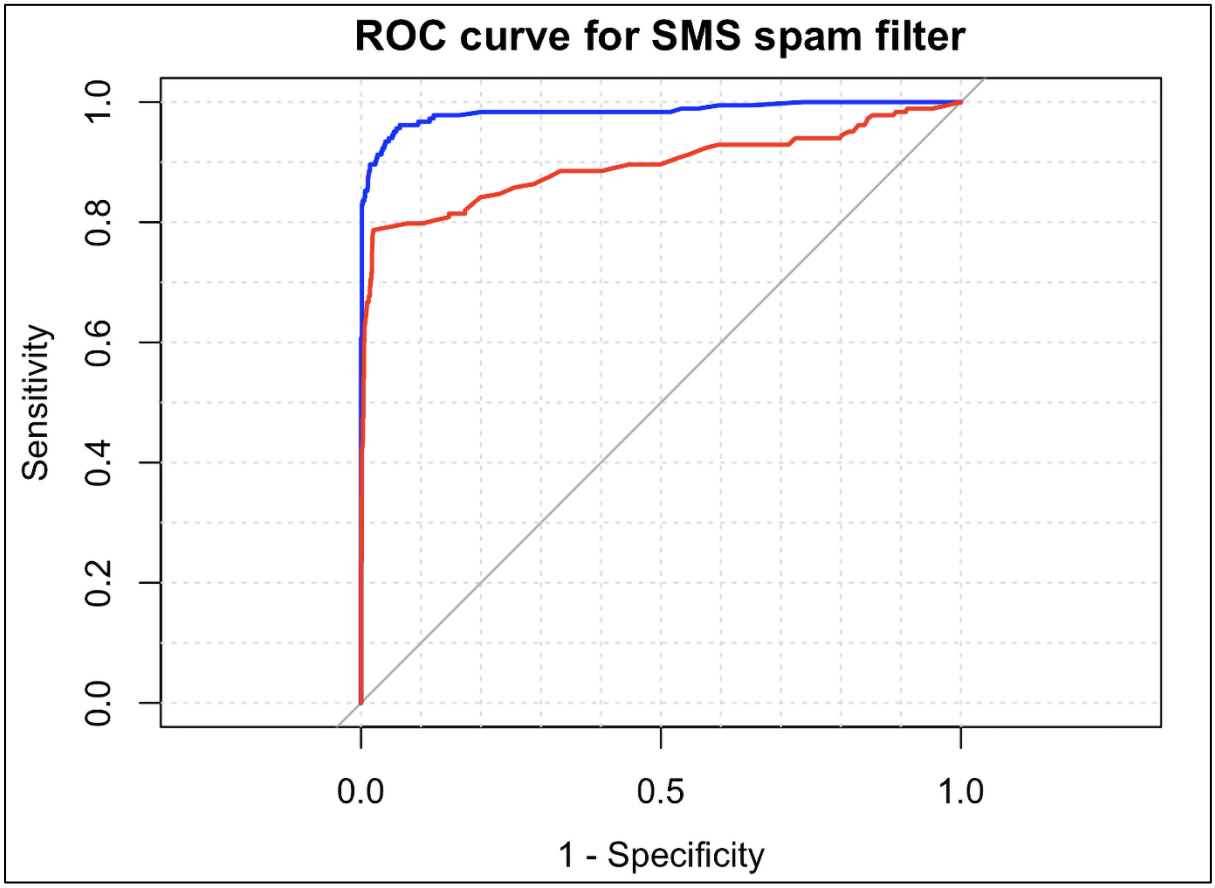

True
Positive
Rate
50
25
0
Model 1
Model 2
25
50
False Positive Rate

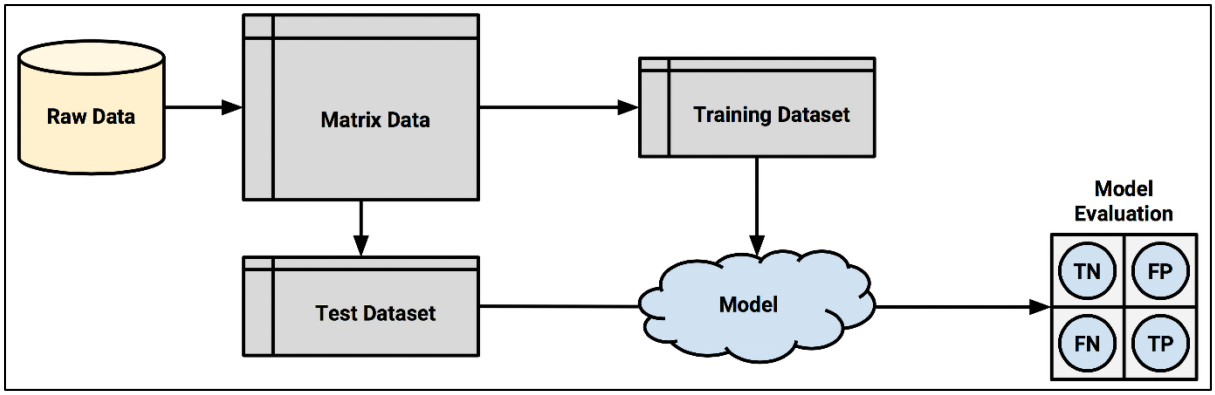

True
Positive
Rate
100
75
50
25
0
Model 1
Model 2
25
50
75
100
False Positive Rate

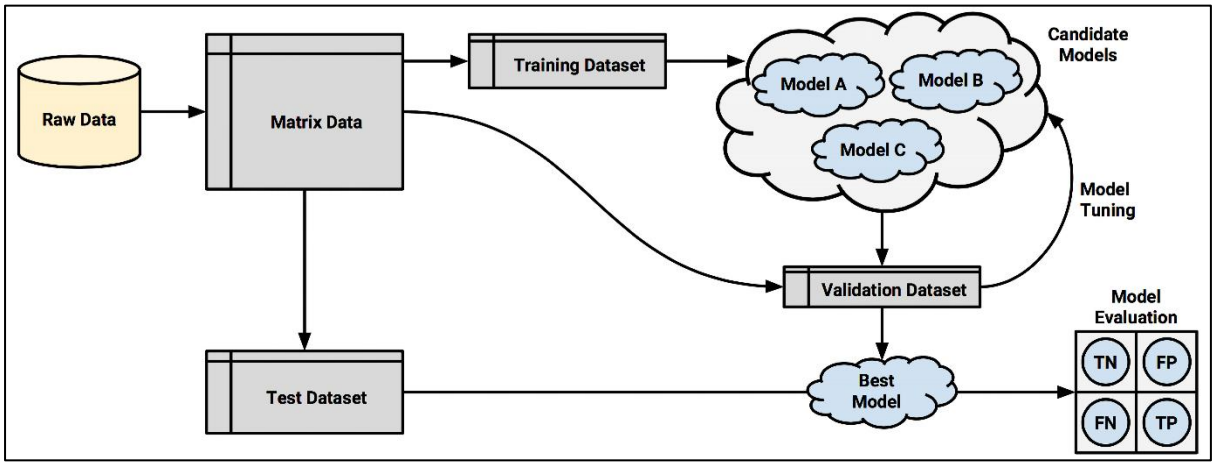

True Positive Rate
100
75
50
25
0
Model 2
Model 1
25
50
75
100
False Positive Rate


True Positive Rate
100
75
50
25
0
AUC = 0.70
AUC = 0.70
25
50
75
100
False Positive Rate

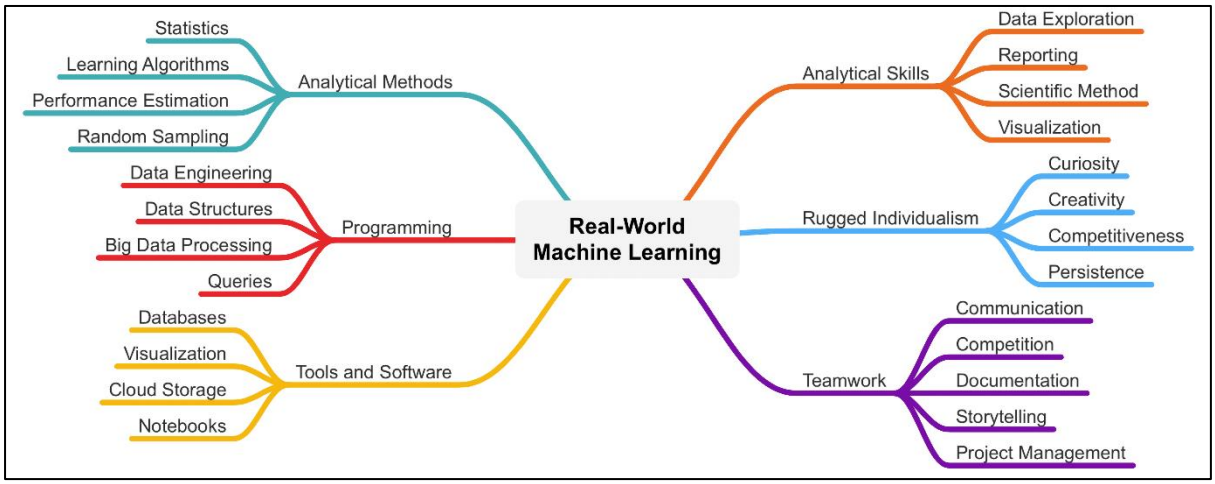

ROC curve for SMS spam filter
Sensitivity
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.5
1.0
1 - Specificity

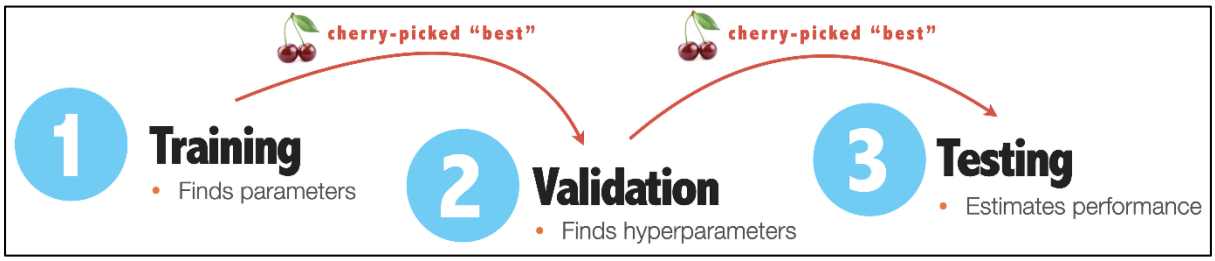

ROC curve for SMS spam filter
Sensitivity
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.5
1.0
1 - Specificity

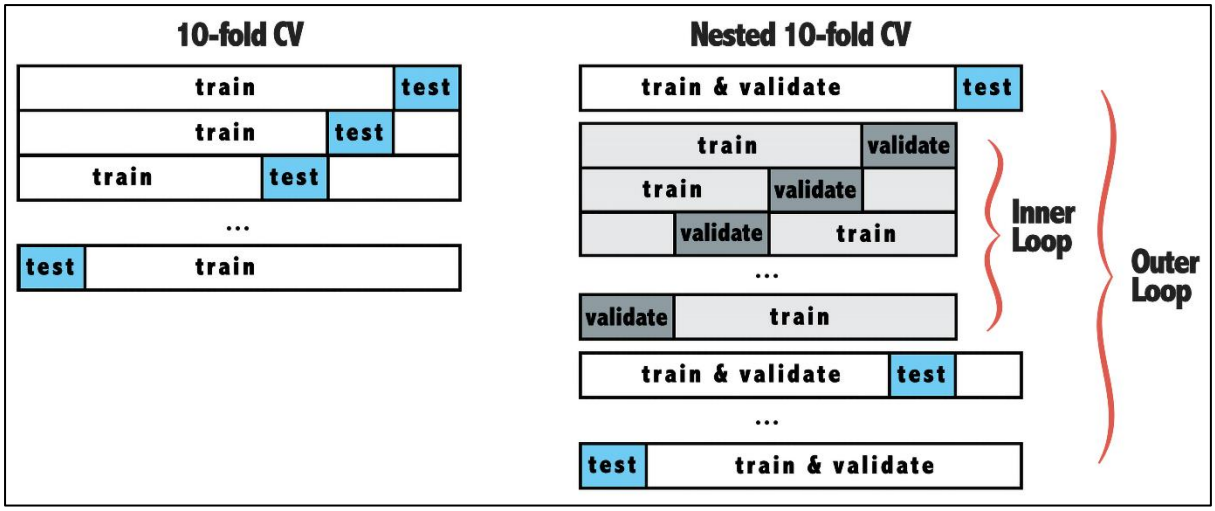

Raw Data
Matrix Data
Training Dataset
Test Dataset
Model
Model Evaluation
TN
FP
FN
TP


Raw Data
Matrix Data
Training Dataset
Model A
Model B
Model C
Candidate Models
Model Tuning
Validation Dataset
Test Dataset
Best Model
Model Evaluation
TN
FP
FN
TP

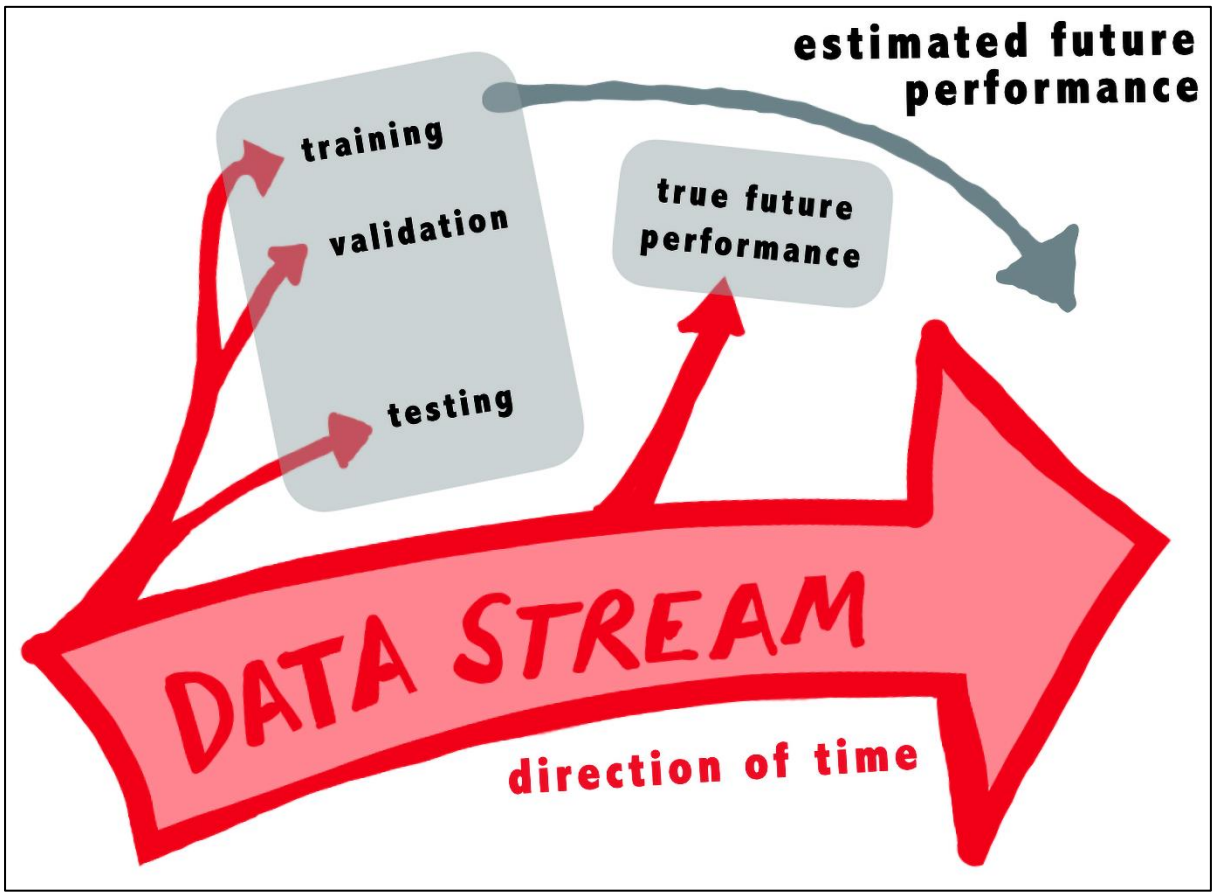
Chapter 11: Being Successful with Machine Learning

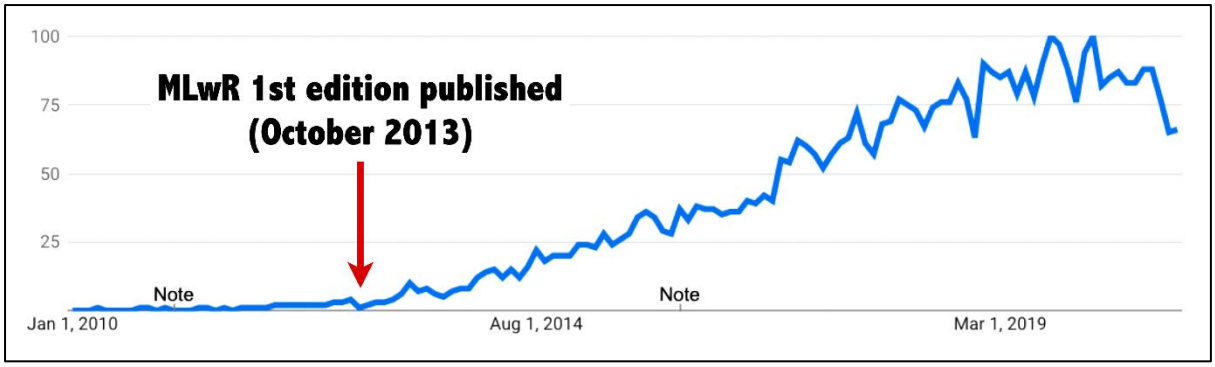


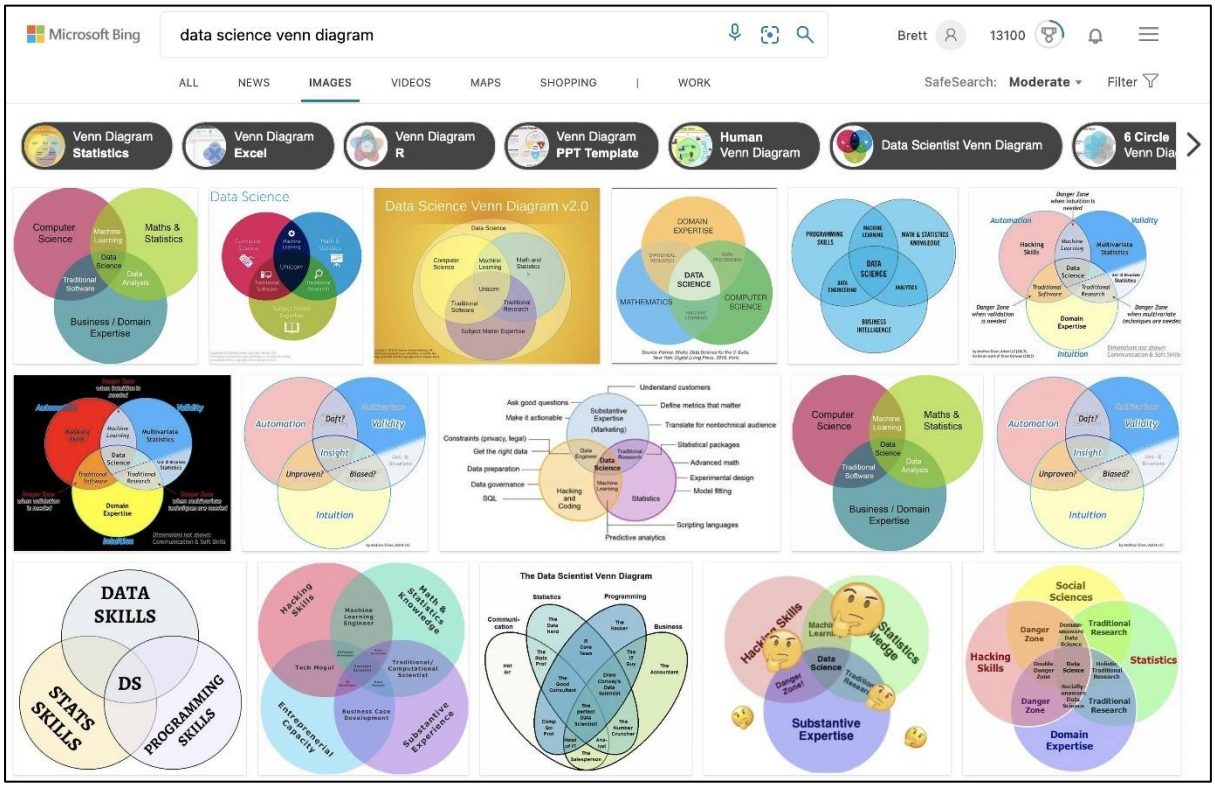


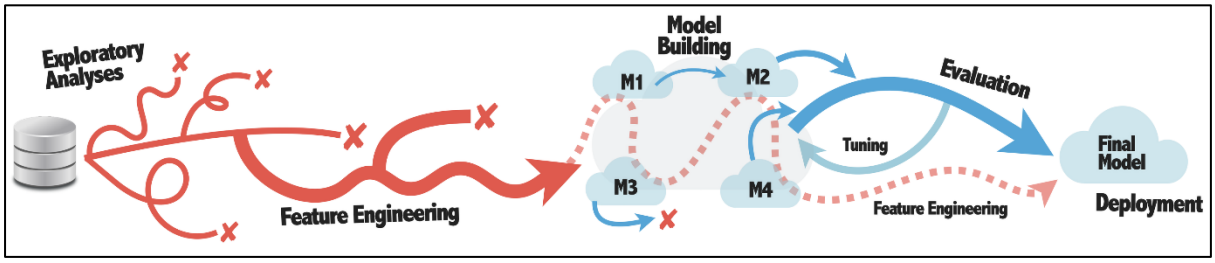


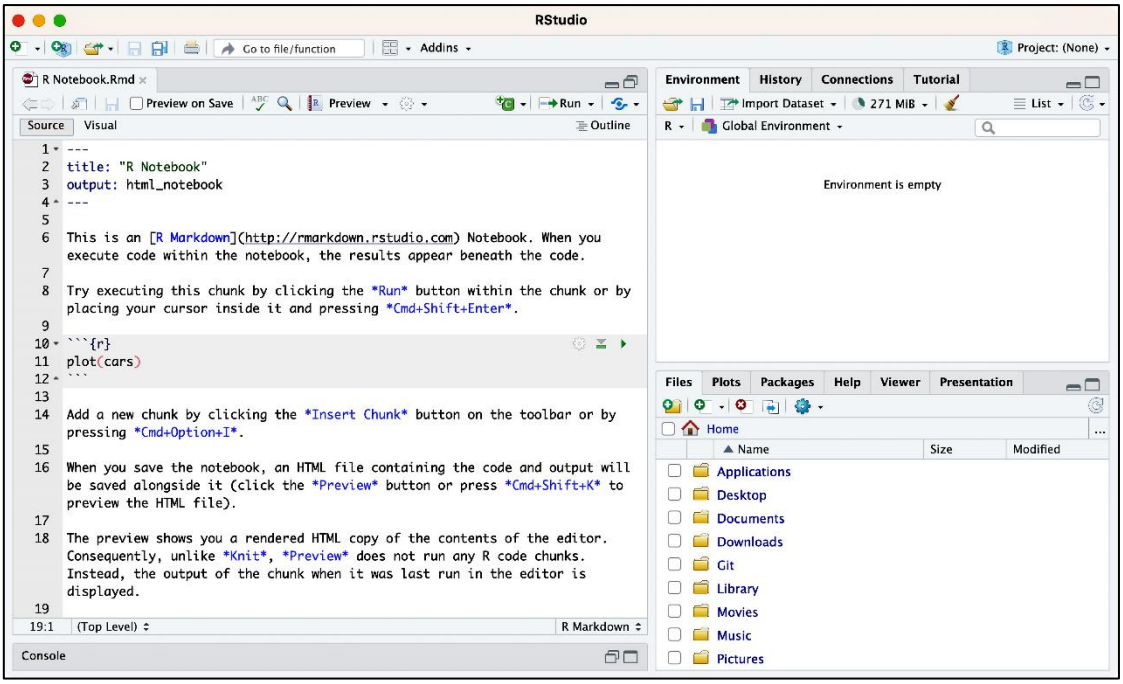

EXPERIENCE CITI® DIAMOND PREFERRED®
The Citi ThankYou® Preferred Card.
Not only a great intro rate but now twice as rewarding.
2X
1X
THE ONLY CARD THAT LETS YOU EARN
CASH BACK TWICE
ON ALL PURCHASES
WITH:
1% CASH BACK
1% CASH BACK
MAKE 2015 THE BEST YEAR YET.
EARN CASH BACK TWICE
ON EVERY PURCHASE WITH:
1% UNLIMITED CASH BACK
1% ADDITIONAL CASH BACK
THE CITI® DIAMOND PREFERRED® CARD

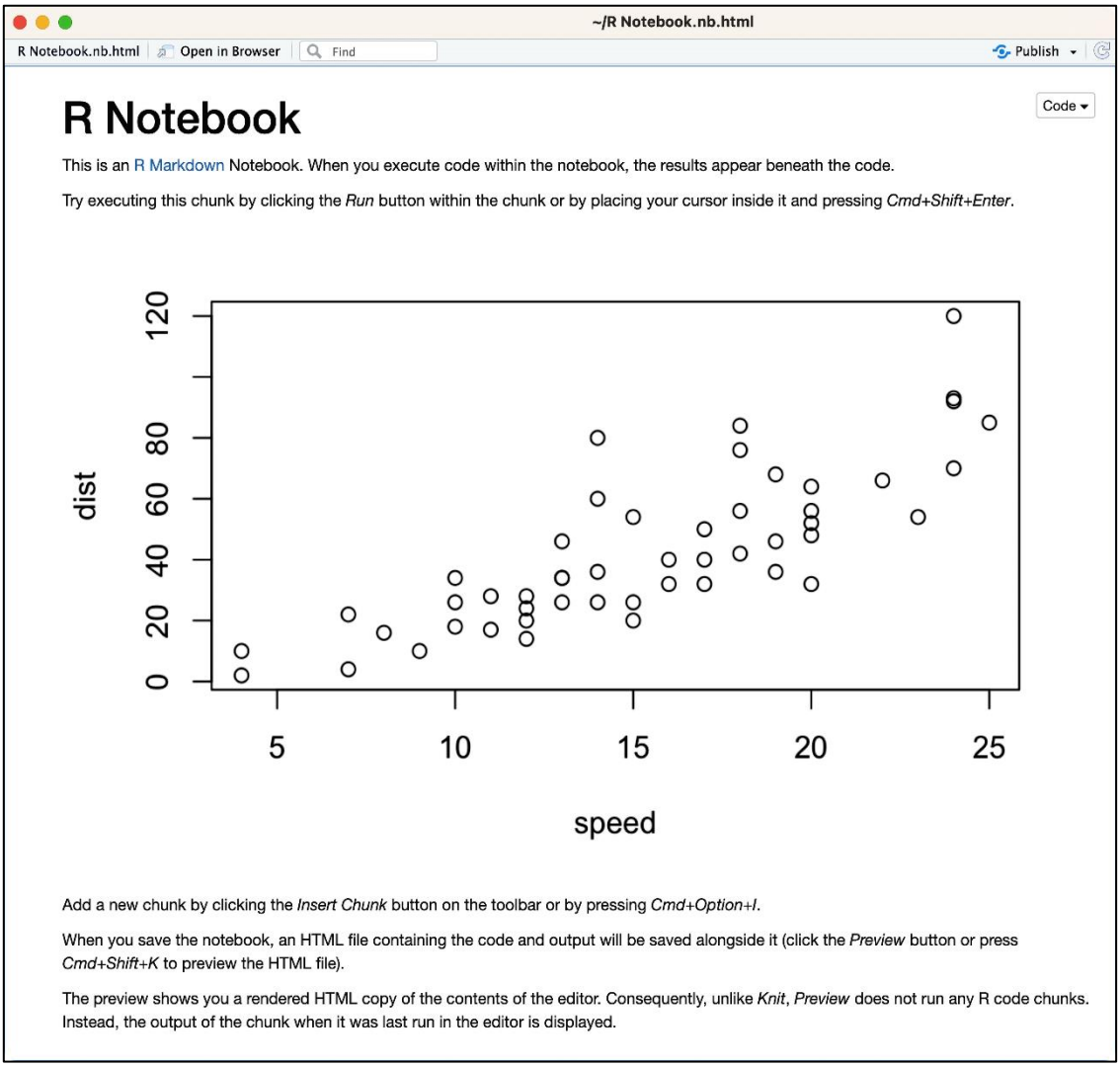

estimated future performance
training
validation
testing
true future performance
DATA STREAM
direction of time


MLwR 1st edition published
(October 2013)
100
75
50
25
Note
Note
Jan 1, 2010
Aug 1, 2014
Mar 1, 2019

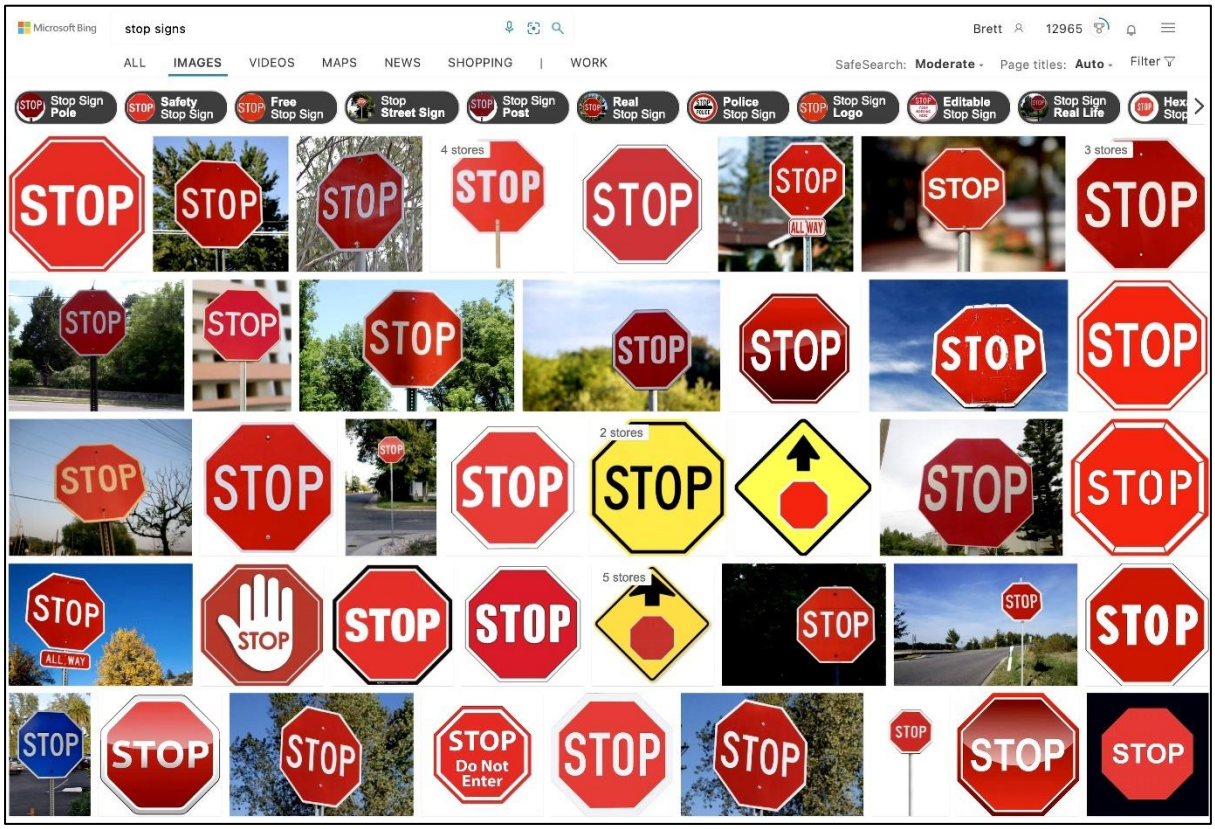

Microsoft Bing
data science venn diagram
Brett
13100
ALL
NEWS
IMAGES
VIDEOS
MAPS
SHOPPING
WORK
SafeSearch: Moderate
Filter
Venn Diagram Statistics
Venn Diagram Excel
Venn Diagram R
Venn Diagram PPT Template
Human Venn Diagram
Data Scientist Venn Diagram
6 Circle Venn Dia
Data Science
Data Science Venn Diagram v2.0
DATA SKILLS
DS
STATS SKILLS
PROGRAMMING SKILLS
The Data Scientist Venn Diagram
Substantive Expertise
Social Sciences
Hacking Skills
Statistics
Danger Zone
Traditional Research
Domain Expertise

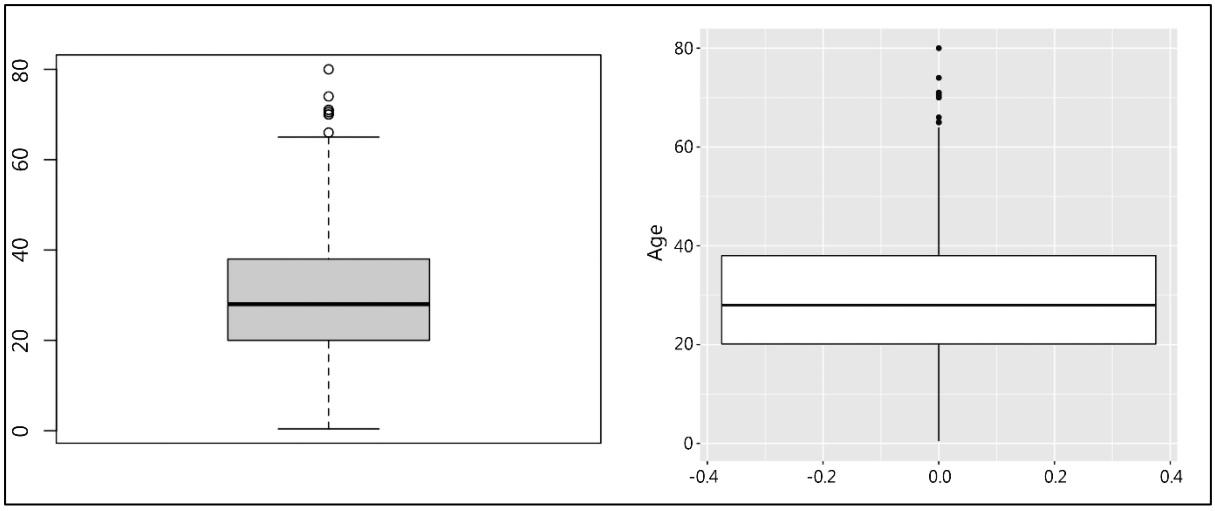

Exploratory Analyses
Feature Engineering
Model Building
M1
M2
M3
M4
Evaluation
Tuning
Feature Engineering
Final Model
Deployment

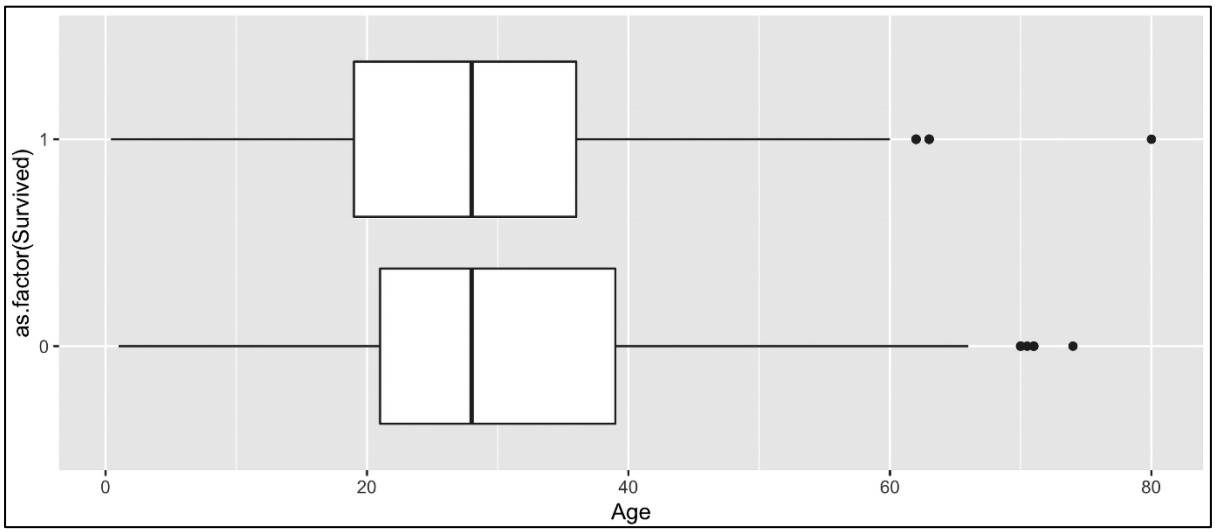

RStudio
Go to file/function
Addins
Project: (None)
R Notebook.Rmd
Preview on Save
Preview
Run
Source
Visual
Outline
1 ---
2 title: "R Notebook"
3 output: html_notebook
4 ---
5
6 This is an [R Markdown](http://rmarkdown.rstudio.com) Notebook. When you execute code within the notebook, the results appear beneath the code.
7
8 Try executing this chunk by clicking the *Run* button within the chunk or by placing your cursor inside it and pressing *Cmd+Shift+Enter*.
9
10 ```{r}
11 plot(cars)
12 ```
13
14 Add a new chunk by clicking the *Insert Chunk* button on the toolbar or by pressing *Cmd+Option+I*.
15
16 When you save the notebook, an HTML file containing the code and output will be saved alongside it (click the *Preview* button or press *Cmd+Shift+K* to preview the HTML file).
17
18 The preview shows you a rendered HTML copy of the contents of the editor. Consequently, unlike *Knit*, *Preview* does not run any R code chunks. Instead, the output of the chunk when it was last run in the editor is displayed.
19
19:1
(Top Level)
R Markdown
Console
Environment
History
Connections
Tutorial
Import Dataset
271 MiB
List
R
Global Environment
Environment is empty
Files
Plots
Packages
Help
Viewer
Presentation
Home
Name
Size
Modified
Applications
Desktop
Documents
Downloads
Git
Library
Movies
Music
Pictures

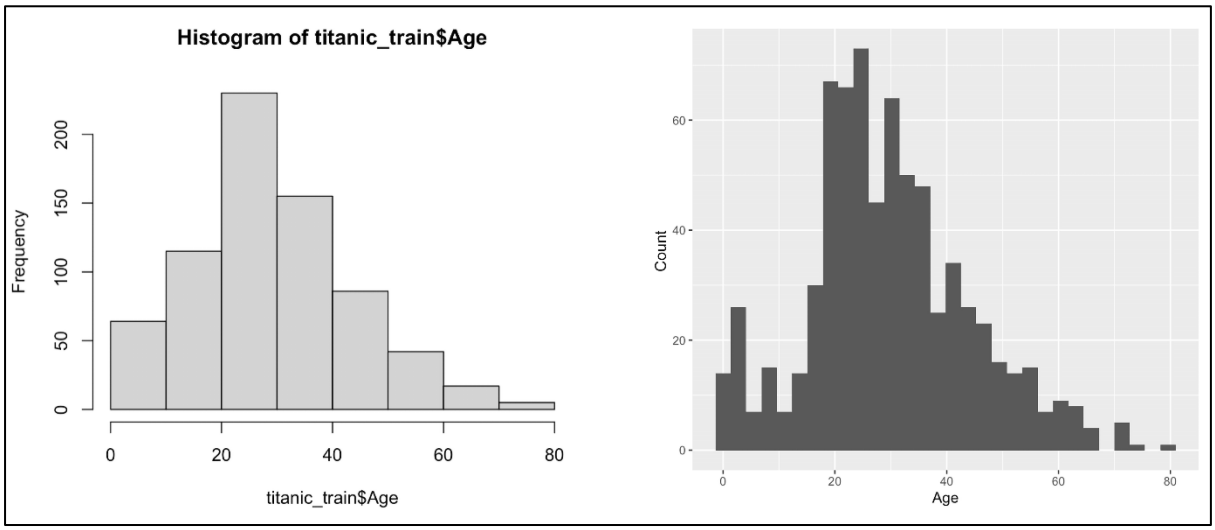

~/R Notebook.nb.html
R Notebook.nb.html
Open in Browser
Find
Publish
Code
R Notebook
This is an R Markdown Notebook. When you execute code within the notebook, the results appear beneath the code.
Try executing this chunk by clicking the Run button within the chunk or by placing your cursor inside it and pressing Cmd+Shift+Enter.
dist
0
20
40
60
80
120
5
10
15
20
25
speed
Add a new chunk by clicking the Insert Chunk button on the toolbar or by pressing Cmd+Option+I.
When you save the notebook, an HTML file containing the code and output will be saved alongside it (click the Preview button or press Cmd+Shift+K to preview the HTML file).
The preview shows you a rendered HTML copy of the contents of the editor. Consequently, unlike Knit, Preview does not run any R code chunks. Instead, the output of the chunk when it was last run in the editor is displayed.

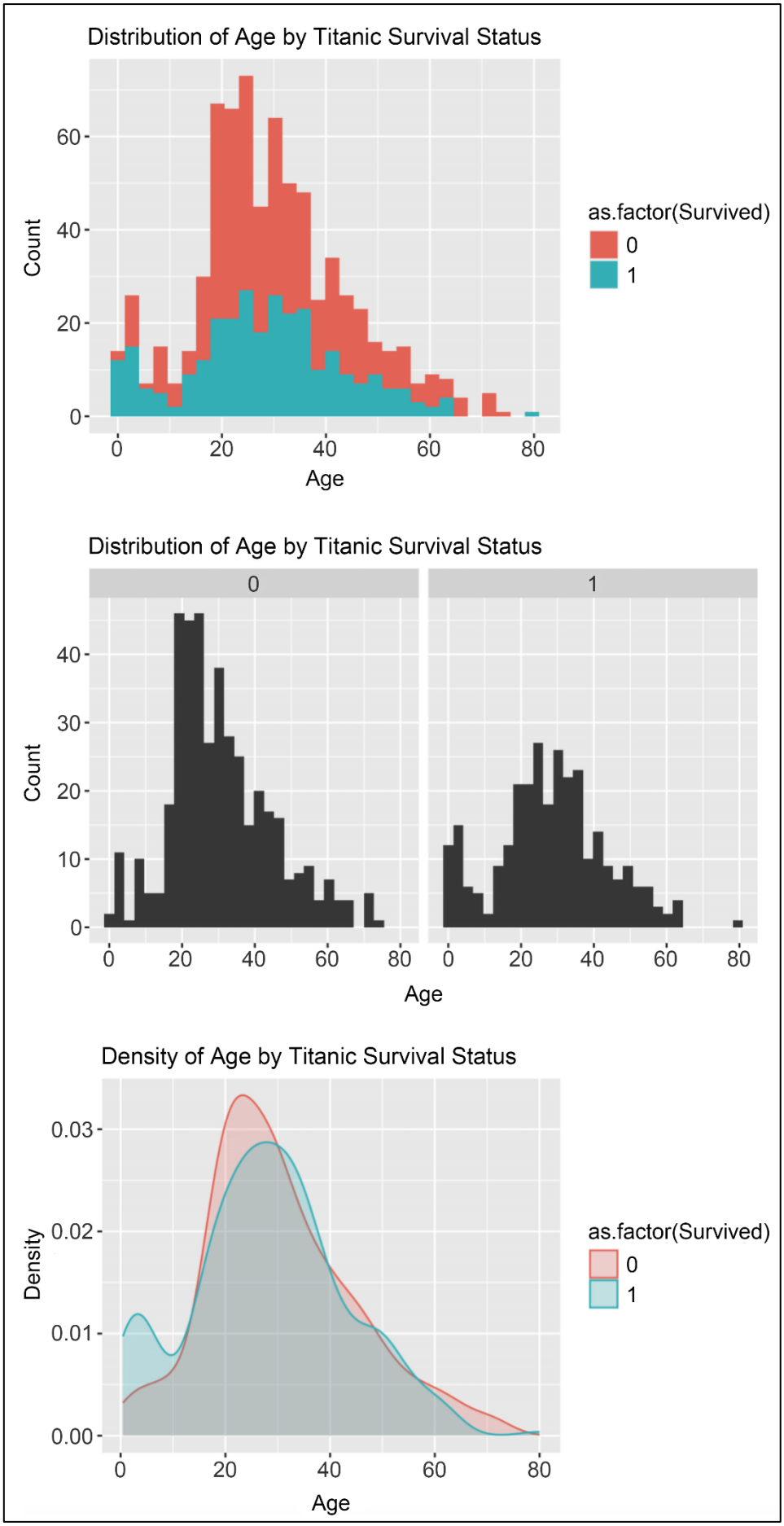
LoanStatNew	Description
acc_now_delinq	The number of accounts on which the borrower is now delinquent.
acc_open_past_24mths	Number of trades opened in past 24 months.
addr_state	The state provided by the borrower in the loan application
all_util	Balance to credit limit on all trades
annual_inc	The self-reported annual income provided by the borrower during registration. ‖ correlated
annual_inc_joint	The combined self-reported annual income provided by the co-borrowers during registration
application_type	Indicates whether the loan is an individual application or a joint application with two co-borrowers
avg_cur_bal	Average current balance of all accounts
bc_open_to_buy	Total open to buy on revolving bankcards.
bc_util	Ratio of total current balance to high credit/credit limit for all bankcard accounts.
chargeoff_within_12_mths	Number of charge-offs within 12 months
collection_recovery_fee	post charge off collection fee
collections_12_mths_ex_med	Number of collections in 12 months excluding medical collections
delinq_2yrs	The number of 30+ days past-due incidences of delinquency in the borrower's credit file for the past 2 years
delinq_amnt	The past-due amount owed for the accounts on which the borrower is now delinquent.
desc	Loan description provided by the borrower text data
dti	A ratio calculated using the borrower's total monthly debt payments on the total debt obligations, excluding mortgage and the requested LC loan
dti_joint	A ratio calculated using the co-borrowers' total monthly payments on the total debt obligations, excluding mortgages and the requested LC loan
earliest_cr_line	The month the borrower's earliest reported credit line was opened is this really month (or month + year?)
emp_length	Employment length in years. Possible values are between 0 and 10 where 0 means less than one year and 10 means ten or more years.
emp_title	The job title supplied by the Borrower when applying for the loan.*
fico_range_high	The upper boundary range the borrower's FICO at loan origination belongs to. ‖ what to do with these?
fico_range_low	The lower boundary range the borrower's FICO at loan origination belongs to.

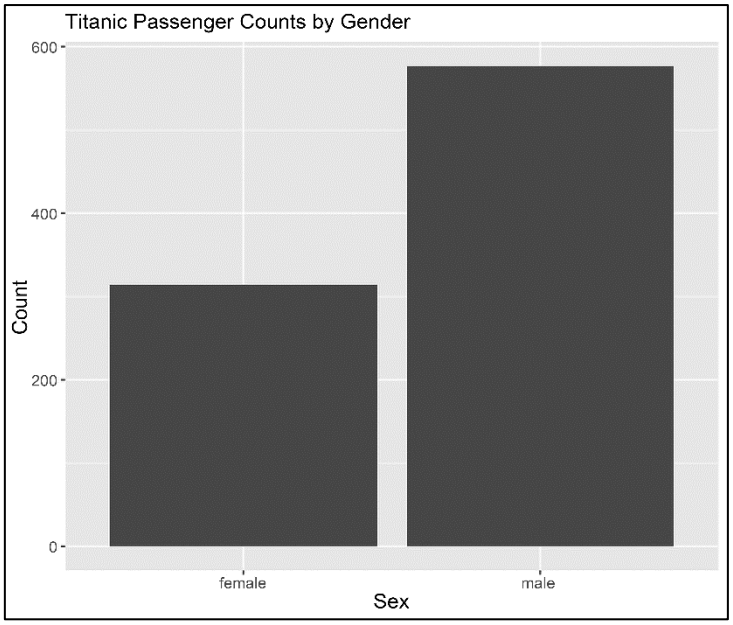


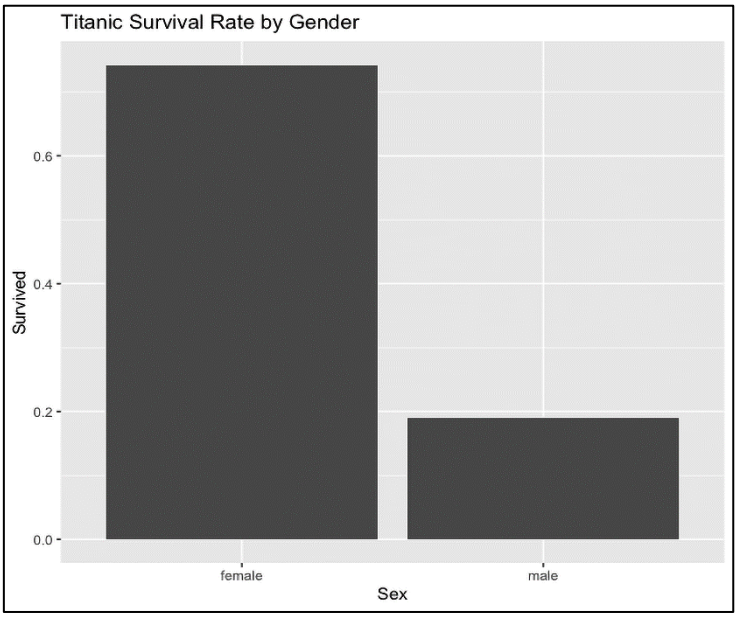

0
20
40
60
80
Age
-0.4
-0.2
0.0
0.2
0.4

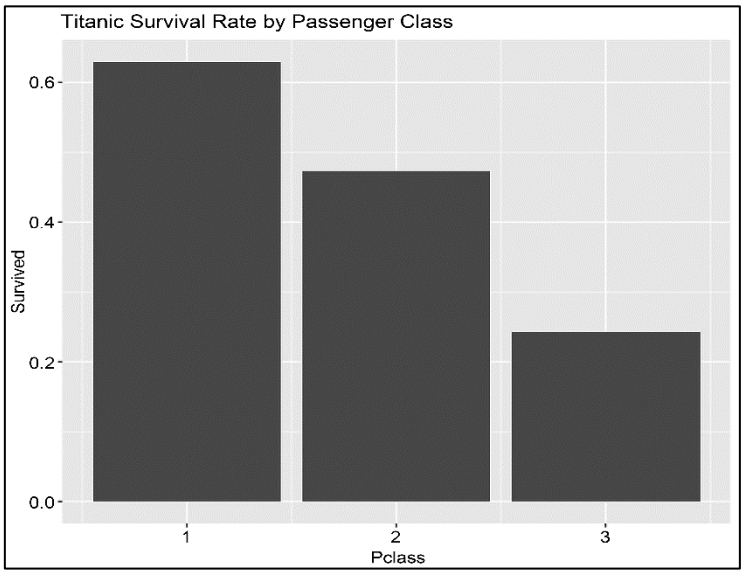

as.factor(Survived)
1
0
0
20
40
60
80
Age

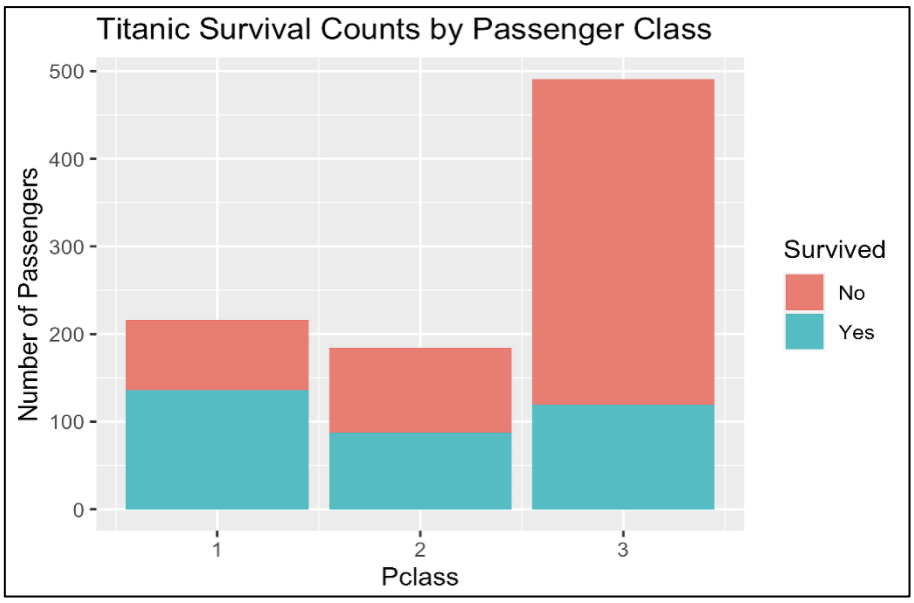

Histogram of titanic_train$Age
Frequency
0
50
100
150
200
0
20
40
60
80
titanic_train$Age
Count
0
20
40
60
0
20
40
60
80
Age

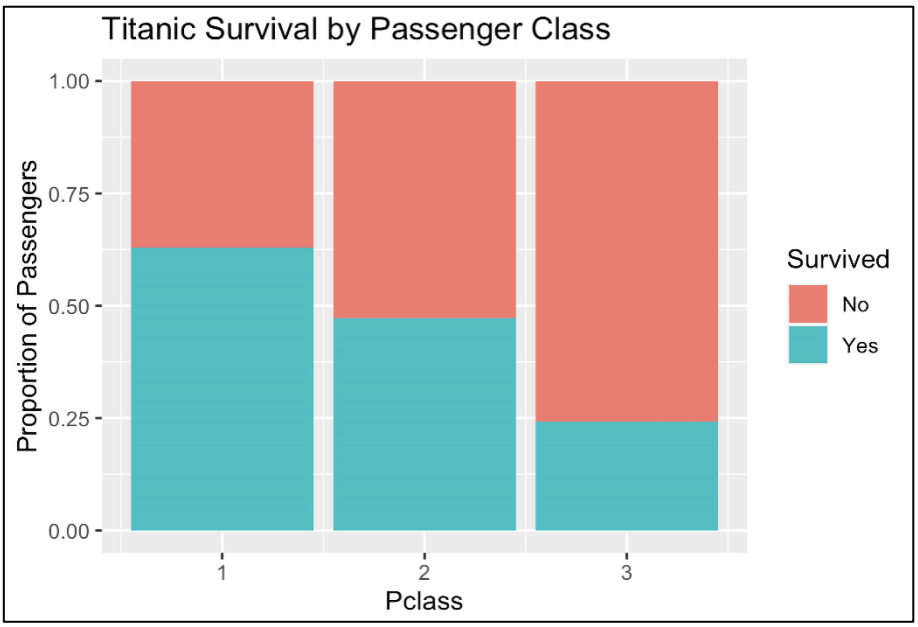

Distribution of Age by Titanic Survival Status
Count
Age
as.factor(Survived)
0
1
Distribution of Age by Titanic Survival Status
0
1
Count
Age
Density of Age by Titanic Survival Status
Density
Age
as.factor(Survived)
0
1

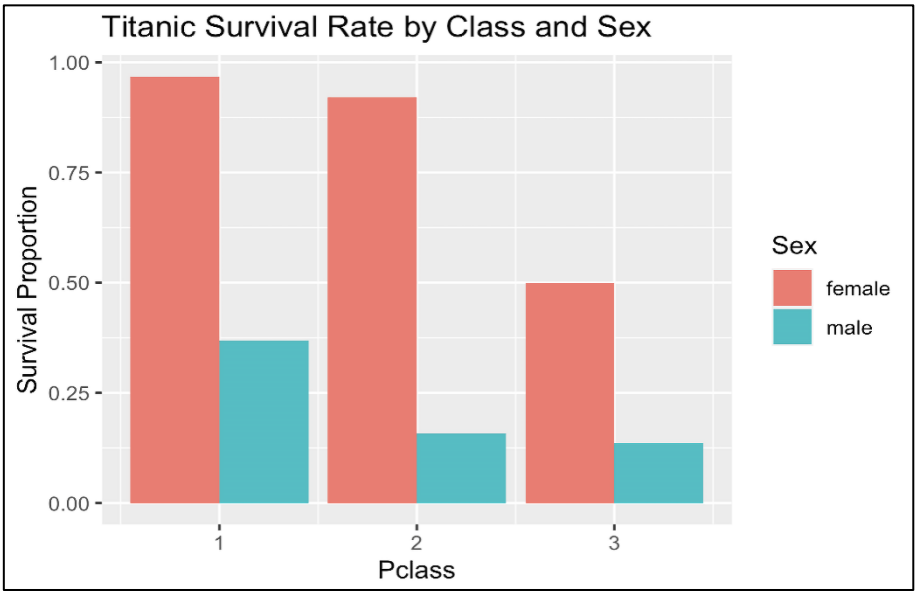

Titanic Passenger Counts by Gender
600
400
200
0
Count
female
male
Sex


Titanic Survival Rate by Gender
0.6
0.4
0.2
0.0
Survived
female
male
Sex

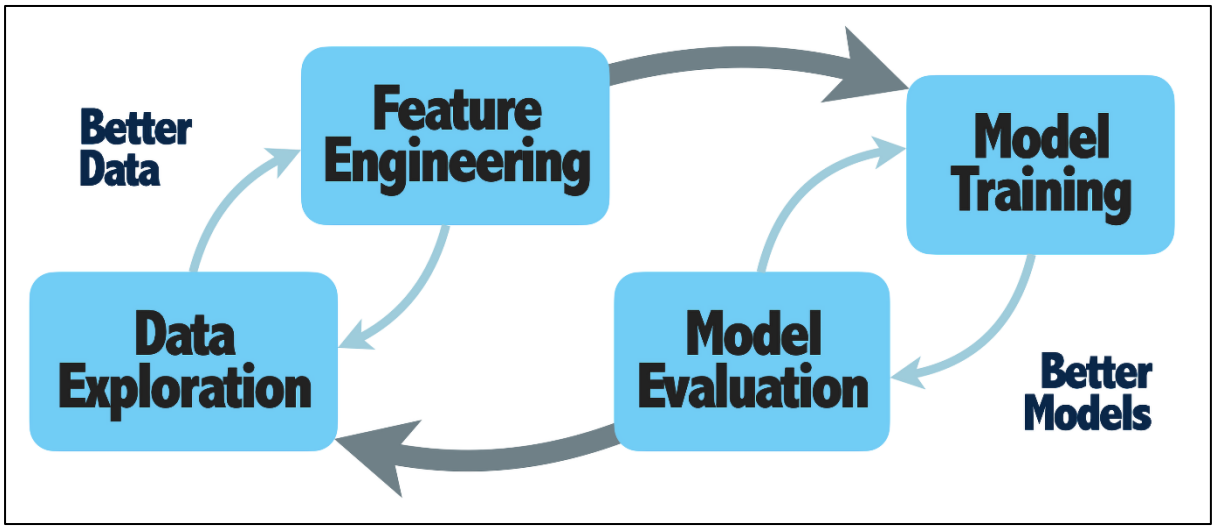

Titanic Survival Rate by Passenger Class
0.6
0.4
0.2
0.0
Survived
1
2
3
Pclass


Titanic Survival Counts by Passenger Class
Number of Passengers
500
400
300
200
100
0
1
2
3
Pclass
Survived
No
Yes

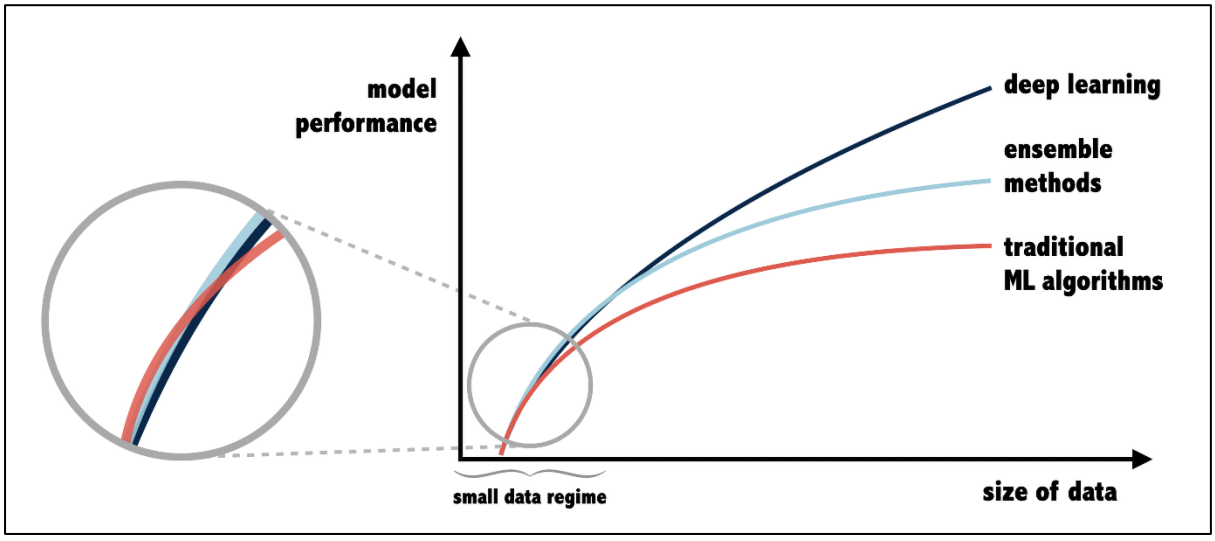

Titanic Survival by Passenger Class
Proportion of Passengers
1.00
0.75
0.50
0.25
0.00
1
2
3
Pclass
Survived
No
Yes

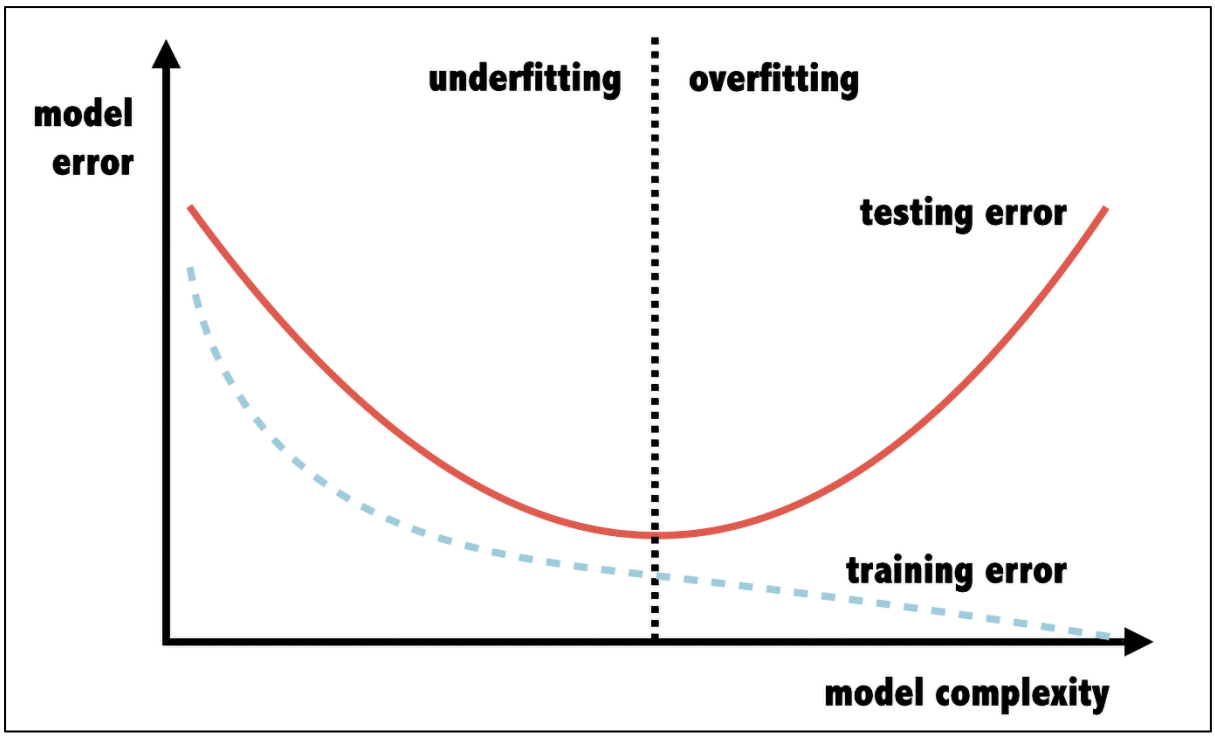

Titanic Survival Rate by Class and Sex
Survival Proportion
1.00
0.75
0.50
0.25
0.00
1
2
3
Pclass
Sex
female
male

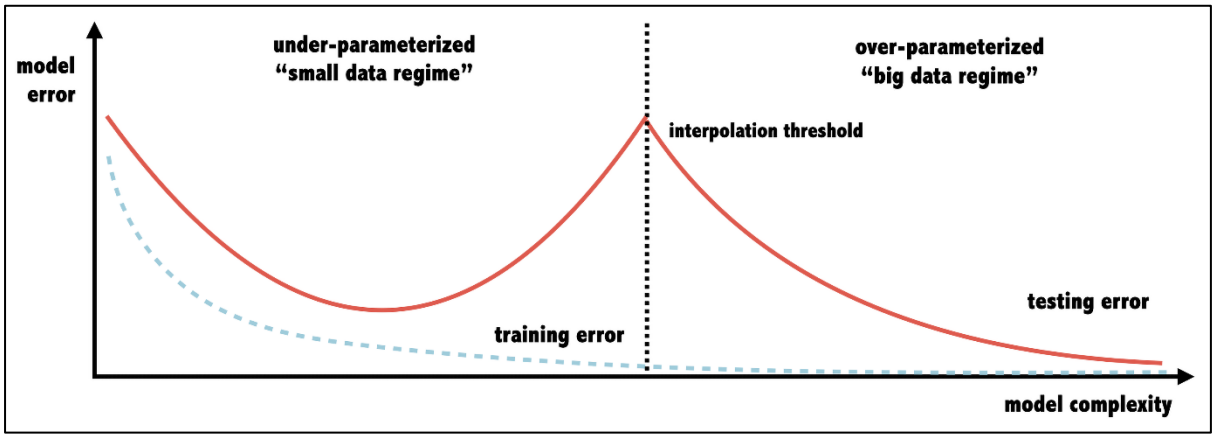
Chapter 12: Advanced Data Preparation

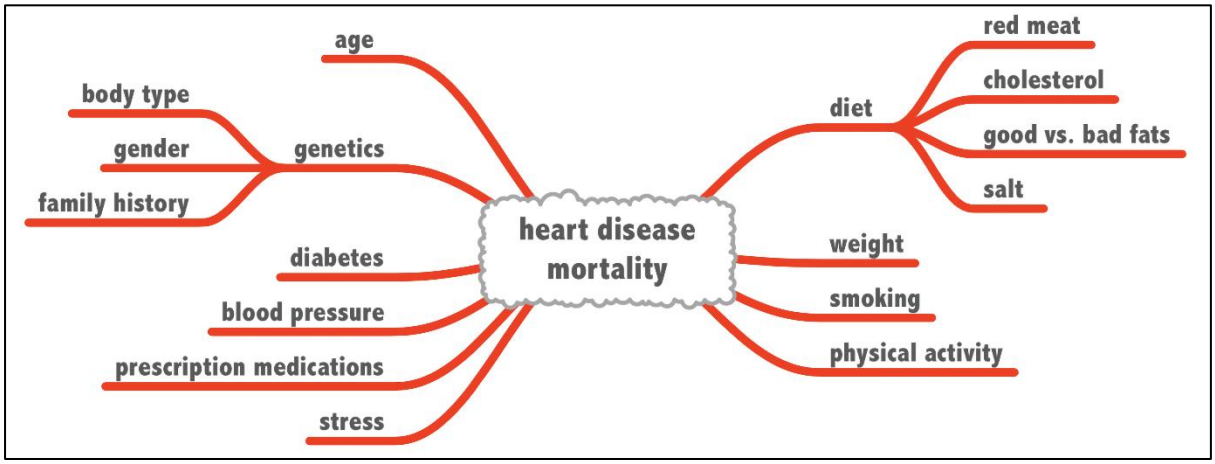


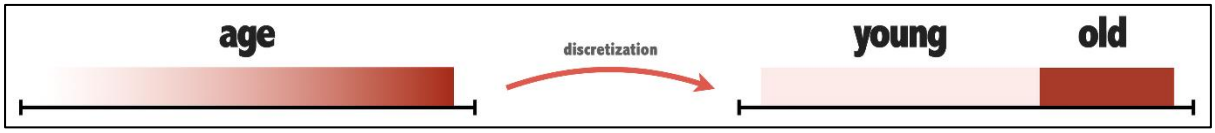
	Small Data	Medium Data	Big Data
Storage Format	Spreadsheets	Databases	+ Data Lakes & NoSQL Databases
Analysis Methods	Traditional Statistics	+ Machine Learning	+ Deep Learning & Artificial Intelligence
Computing Skills	Excel, Visualizations	+ SQL, R, Python	+ Hadoop/Spark, TensorFlow, H2O, etc.
Computing Hardware	Laptop	Laptop/Server	Cloud




model
error
underfitting
overfitting
testing error
training error
model complexity


model
error
under-parameterized
"small data regime"
over-parameterized
"big data regime"
interpolation threshold
training error
testing error
model complexity


age
body type
gender
family history
genetics
diabetes
blood pressure
prescription medications
stress
heart disease
mortality
diet
red meat
cholesterol
good vs. bad fats
salt
weight
smoking
physical activity

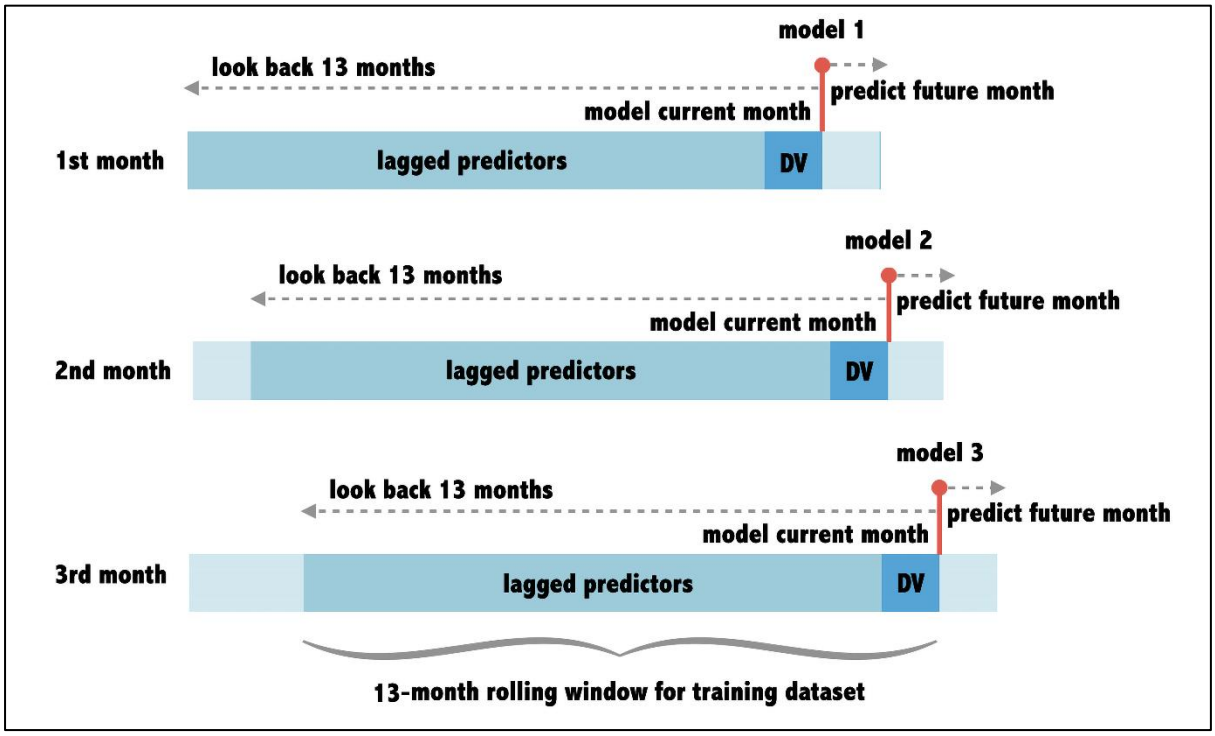

age
discretization
young
old

Long Format

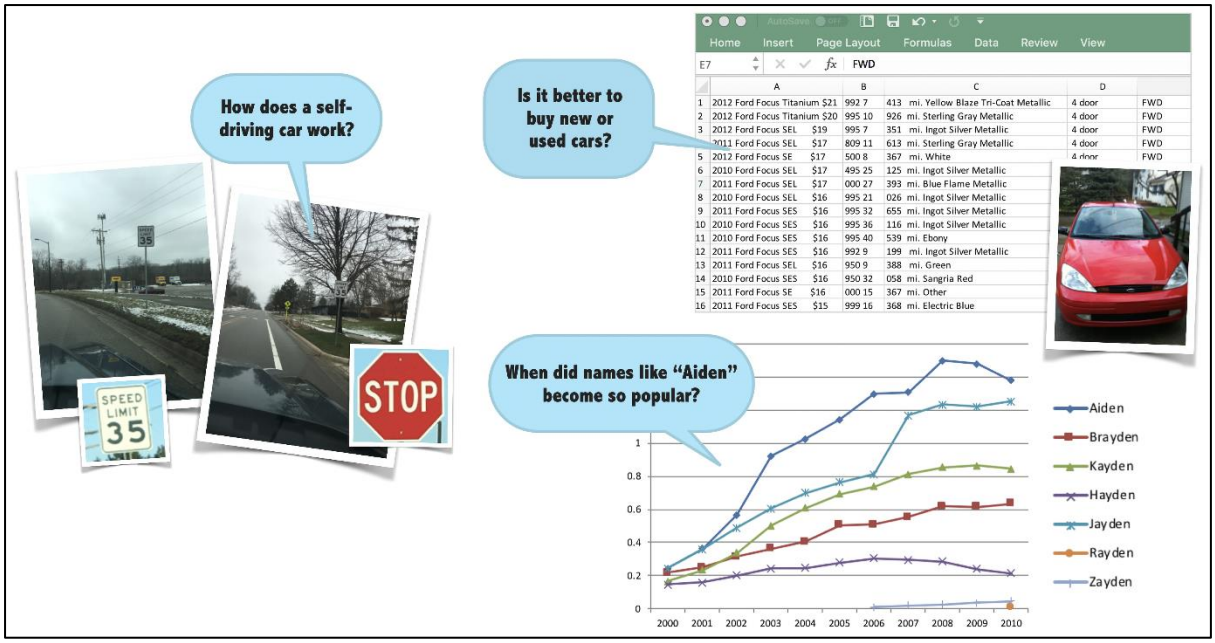
patient_id	month	weight
1	1	78.6 kg
1	2	77.9 kg
1	3	78.3 kg
2	1	56.7 kg
2	2	55.9 kg
2	3	55.3 kg
⋮	⋮	⋮
1000	1	65.8 kg
1000	2	64.9 kg
1000	3	64.1 kg

Wide Format

patient_id	weight_m1	weight_m2	weight_m3
1	78.6 kg	77.9 kg	78.3 kg
2	56.7 kg	55.9 kg	55.3 kg
⋮	⋮	⋮	⋮
1000	65.8 kg	64.9 kg	64.1 kg



household_id	billing_period	energy_use	season	holiday_month	energy_lag1	energy_lag2	energy_lag3
123	December 2022	965	Winter	1	NA	NA	NA
123	January 2023	1034	Winter	0	965	NA	NA
123	February 2023	933	Winter	0	1034	965	NA
123	March 2023	710	Spring	0	933	1034	965
123	April 2023	653	Spring	0	710	933	1034
123	May 2023	545	Spring	0	653	710	933
123	June 2023	748	Summer	0	545	653	710
456	December 2022	899	Winter	1	NA	NA	NA
456	January 2023	932	Winter	0	899	NA	NA
456	February 2023	917	Winter	0	932	899	NA
...	...	...	...	...	...	...	...




```
> titanic_tbl
# A tibble: 891 x 12
   PassengerId Survived Pclass Name                                Sex      Age SibSp Parch Ticket       Fare Cabin Embarked
         <int>    <int>  <int> <chr>                               <chr>  <dbl> <int> <int> <chr>       <dbl> <chr> <chr>
 1           1        0      3 Braund, Mr. Owen Harris             male      22     1     0 A/5 21171    7.25 ""    S
 2           2        1      1 Cumings, Mrs. John Bradley (F…      female    38     1     0 PC 17599    71.3  "C85" C
 3           3        1      3 Heikkinen, Miss. Laina              female    26     0     0 STON/02. 3…  7.92 ""    S
 4           4        1      1 Futrelle, Mrs. Jacques Heath …      female    35     1     0 113803      53.1  "C12… S
 5           5        0      3 Allen, Mr. William Henry            male      35     0     0 373450       8.05 ""    S
 6           6        0      3 Moran, Mr. James                    male      NA     0     0 330877       8.46 ""    Q
 7           7        0      1 McCarthy, Mr. Timothy J             male      54     0     0 17463       51.9  "E46" S
 8           8        0      3 Palsson, Master. Gosta Leonard      male       2     3     1 349909      21.1  ""    S
 9           9        1      3 Johnson, Mrs. Oscar W (Elisab…      female    27     0     2 347742      11.1  ""    S
10          10        1      2 Nasser, Mrs. Nicholas (Adele …      female    14     1     0 237736      30.1  ""    C
# … with 881 more rows
```

Import Text Data

File/URL:

~/Documents/Machine Learning with R/Chapter 12/titanic_train.csv

Browse...

Data Preview:

PassengerId *(double)*	Survived *(double)*	Pclass *(double)*	Name *(character)*	Sex *(character)*	Age *(double)*	SibSp *(double)*
1	0	3	Braund, Mr. Owen Harris	male	22	
2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Thayer)	female	38	
3	1	3	Heikkinen, Miss. Laina	female	26	
4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35	
5	0	3	Allen, Mr. William Henry	male	35	
6	0	3	Moran, Mr. James	male	*NA*	
7	0	1	McCarthy, Mr. Timothy J	male	54	
8	0	3	Palsson, Master. Gosta Leonard	male	2	
9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27	
10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14	

Previewing first 50 entries.

Import Options:

Name: titanic_train

Skip: 0

First Row as Names
Trim Spaces
Open Data Viewer

Delimiter: Comma
Quotes: Default
Locale: Configure...

Escape: None
Comment: Default
NA: Default

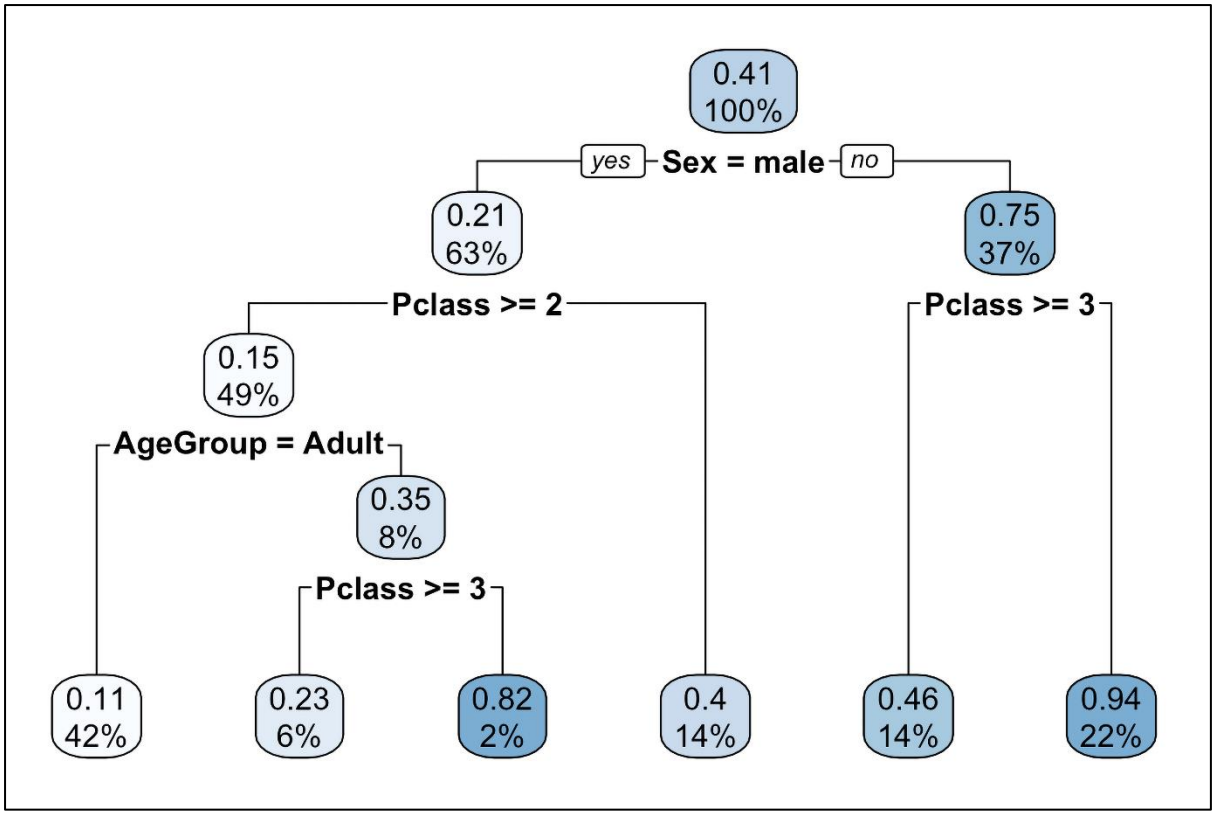
Code Preview:

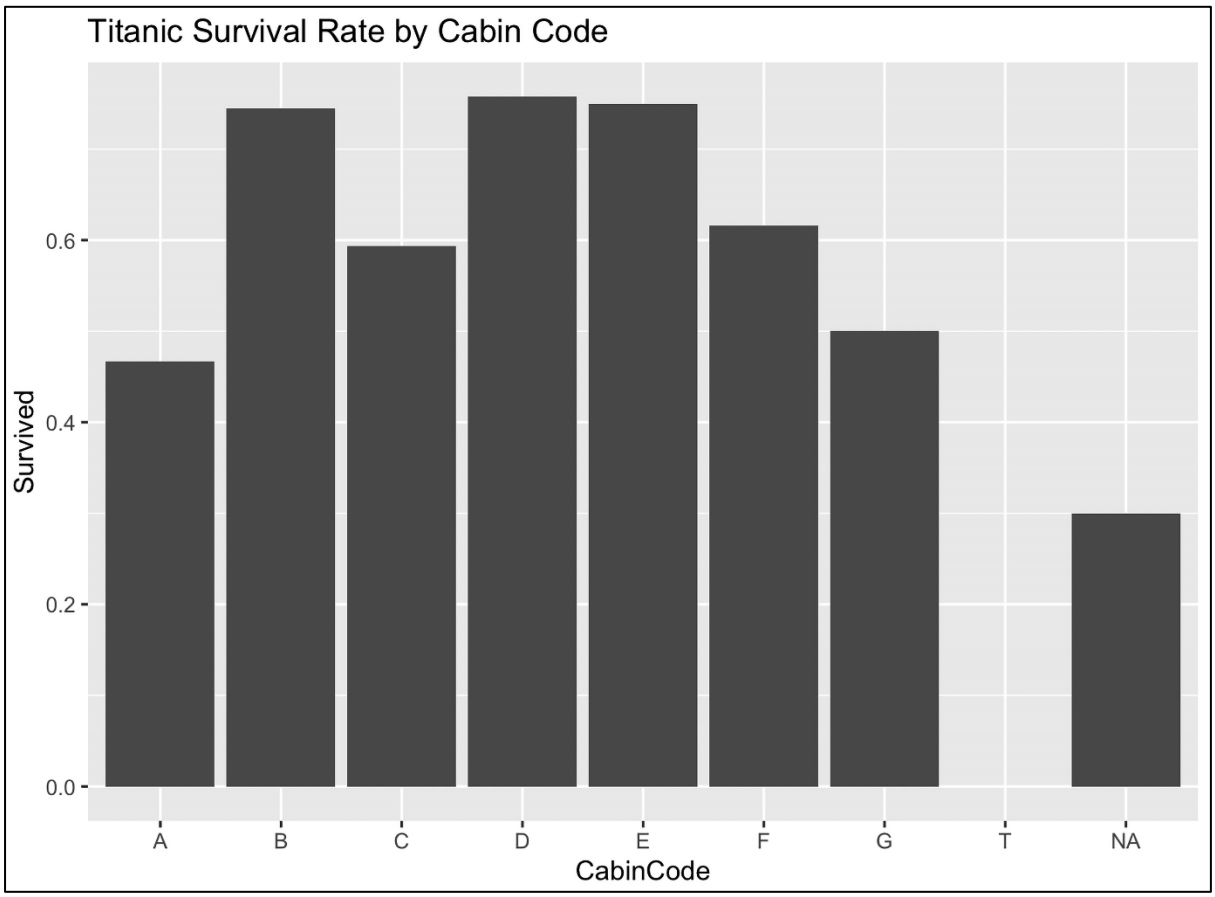
```
library(readr)
titanic_train <-
read_csv
("titanic_train.csv"
)
View(titanic_train)
```

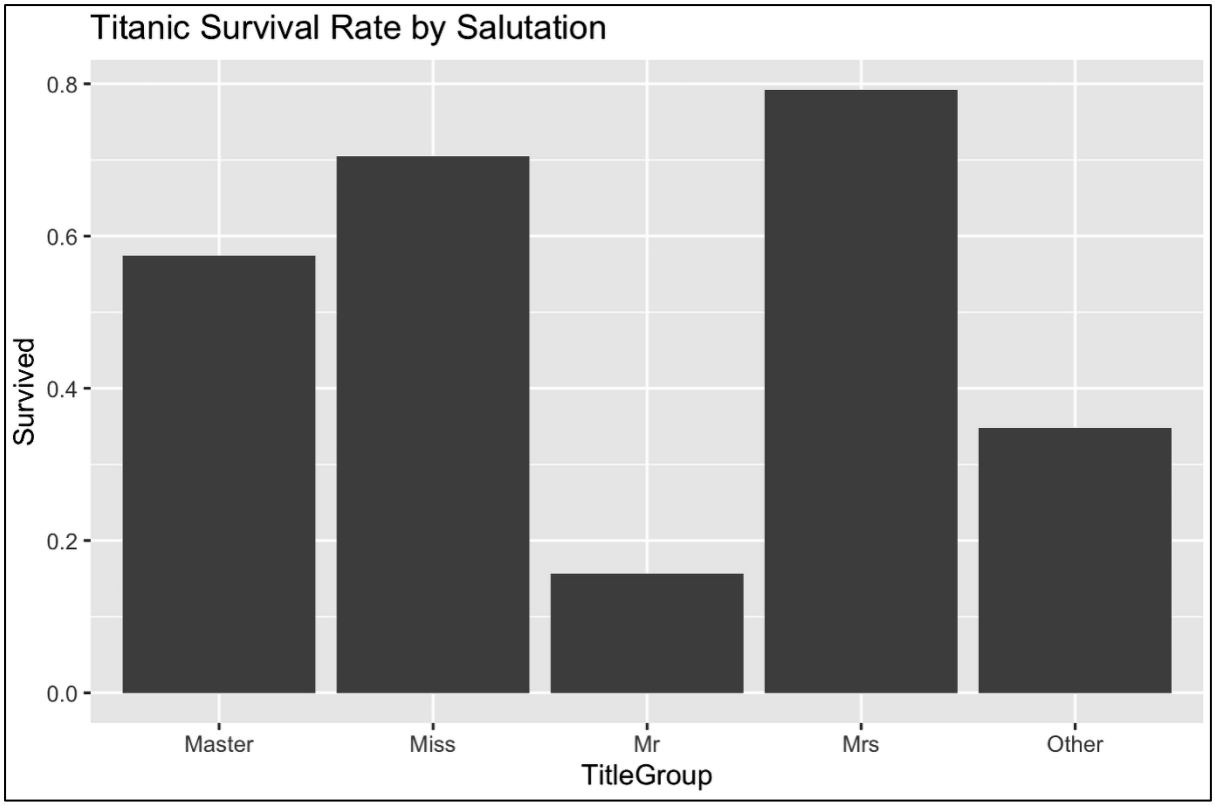
Reading rectangular data using readr

Import

Cancel

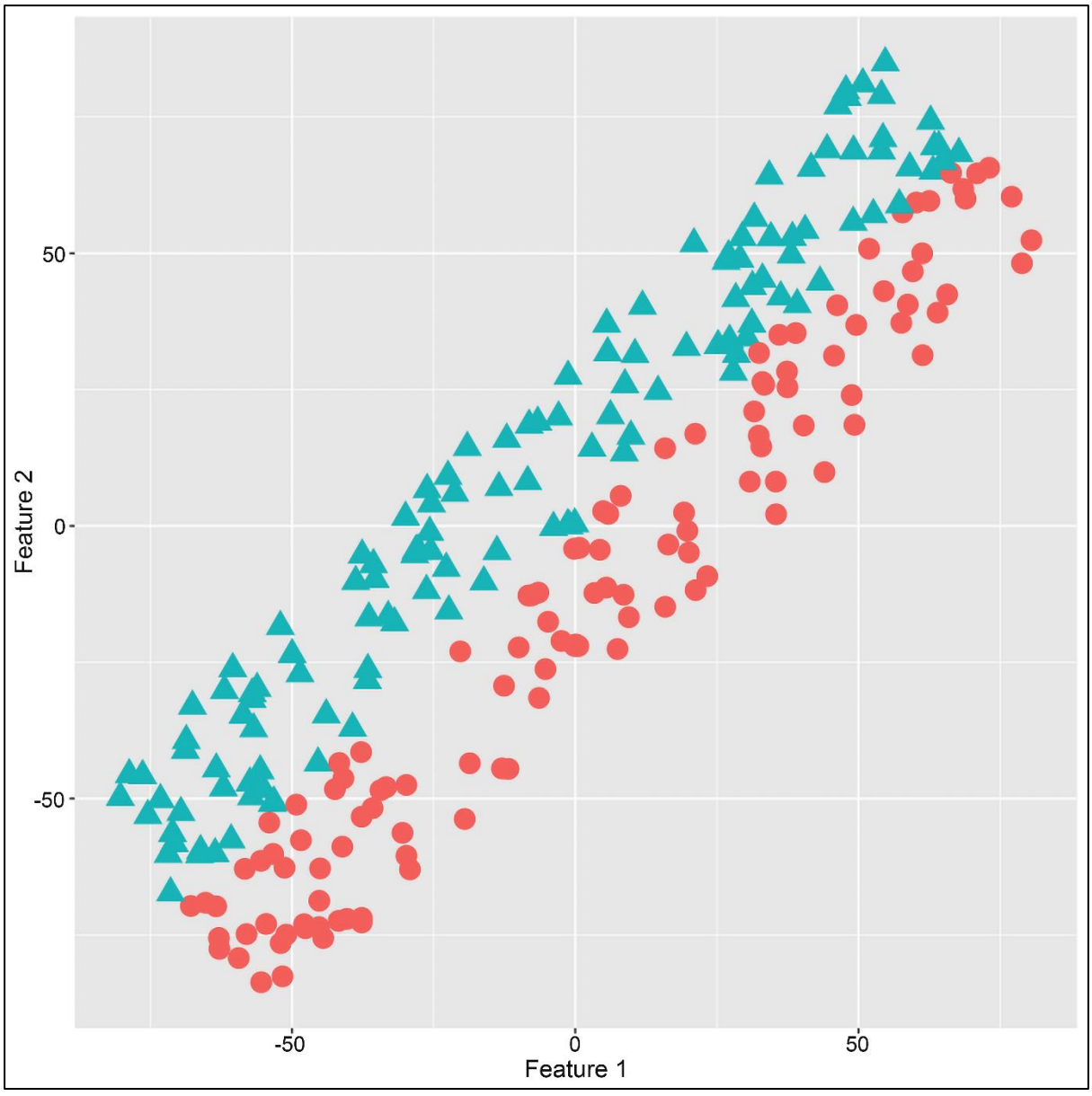

0.41
100%
yes
Sex = male
no
0.21
63%
0.75
37%
Pclass >= 2
Pclass >= 3
0.15
49%
AgeGroup = Adult
0.35
8%
Pclass >= 3
0.11
42%
0.23
6%
0.82
2%
0.4
14%
0.46
14%
0.94
22%

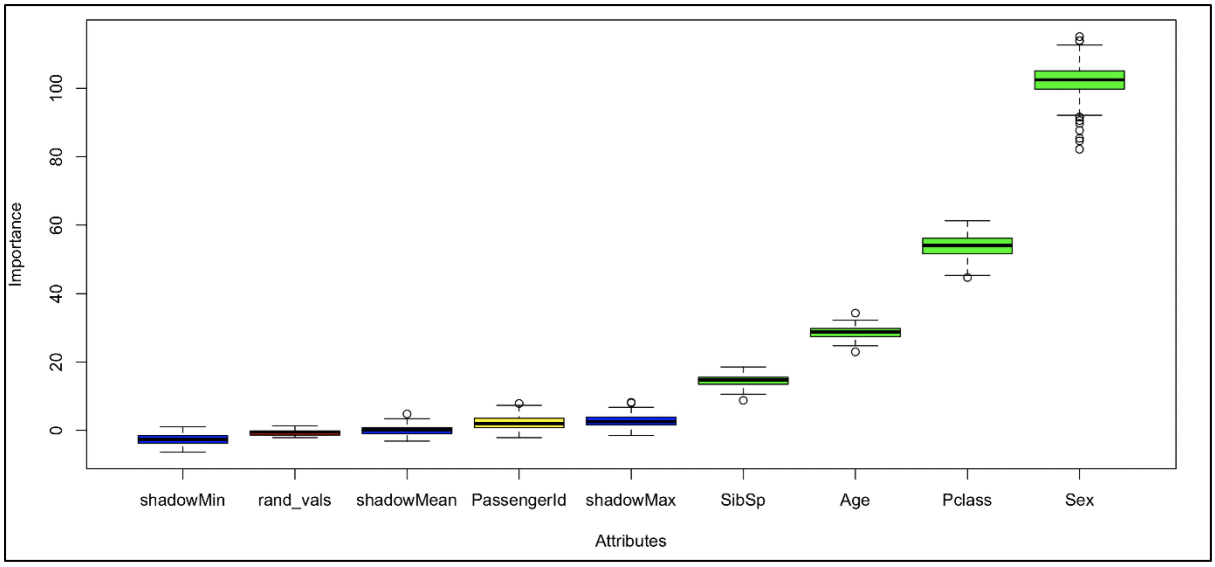

Titanic Survival Rate by Cabin Code
Survived
0.0
0.2
0.4
0.6
A
B
C
D
E
F
G
T
NA
CabinCode

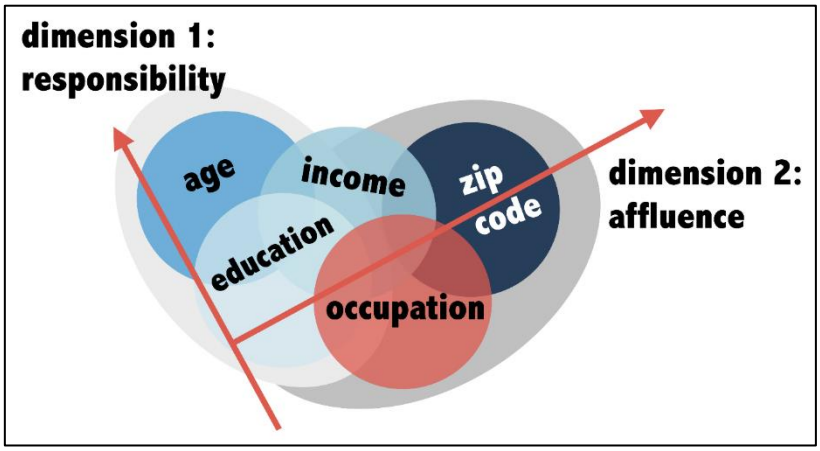

Titanic Survival Rate by Salutation
Survived
0.8
0.6
0.4
0.2
0.0
Master
Miss
Mr
Mrs
Other
TitleGroup

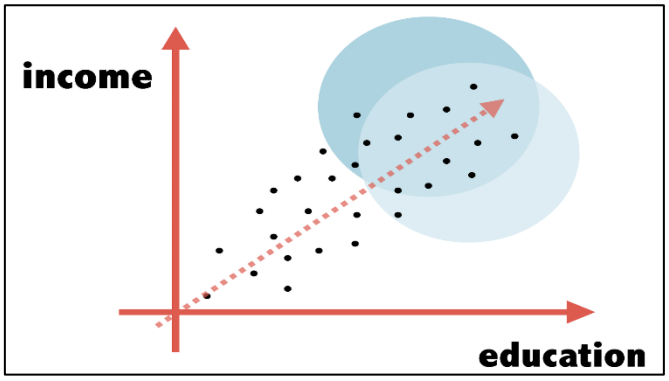
Chapter 13: Challenging Data - Too Much, Too Little, Too Complex

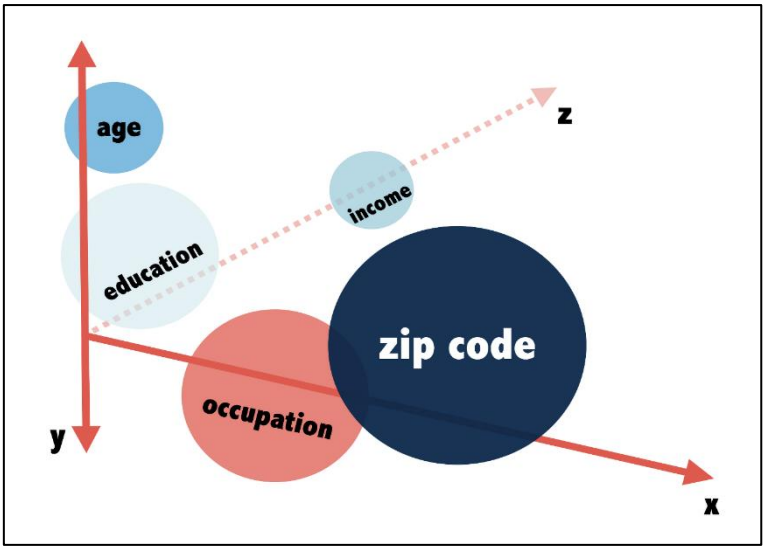


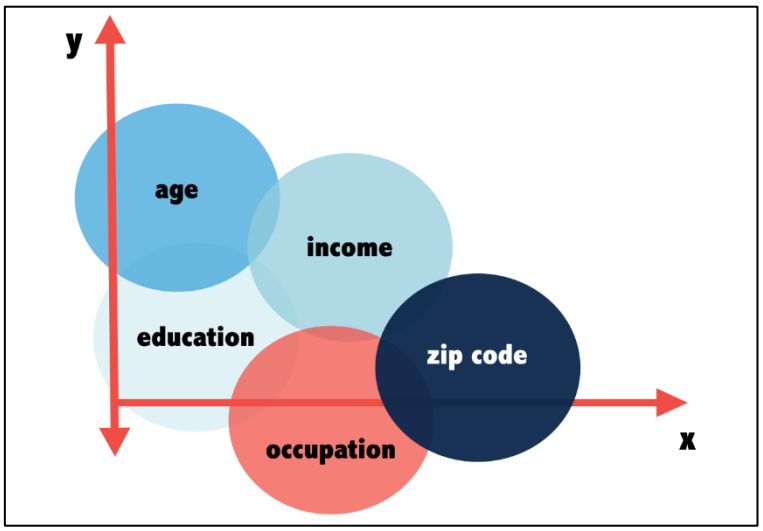


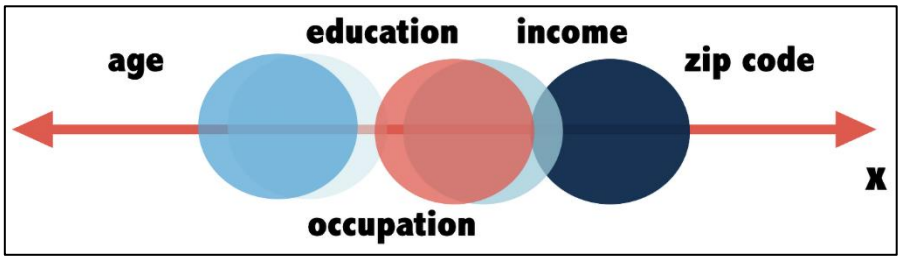

Importance
0
20
40
60
80
100
shadowMin
rand_vals
shadowMean
PassengerId
shadowMax
SibSp
Age
Pclass
Sex
Attributes

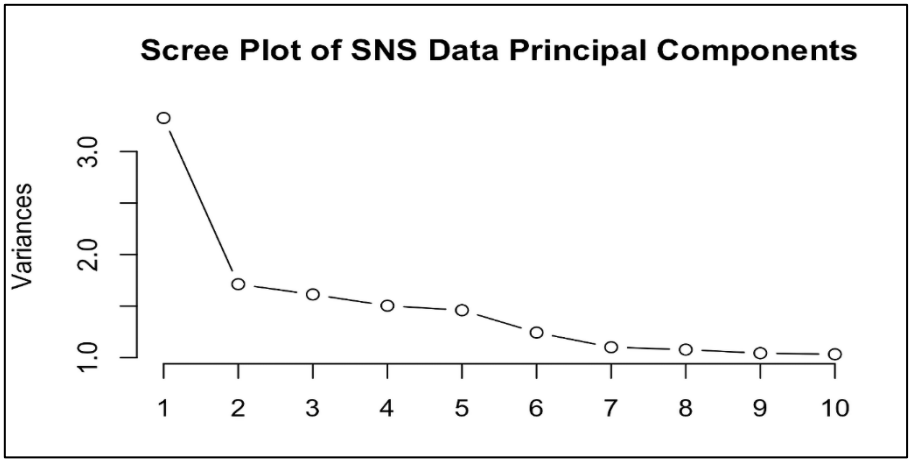

dimension 1:
responsibility
age
income
zip
code
education
occupation
dimension 2:
affluence

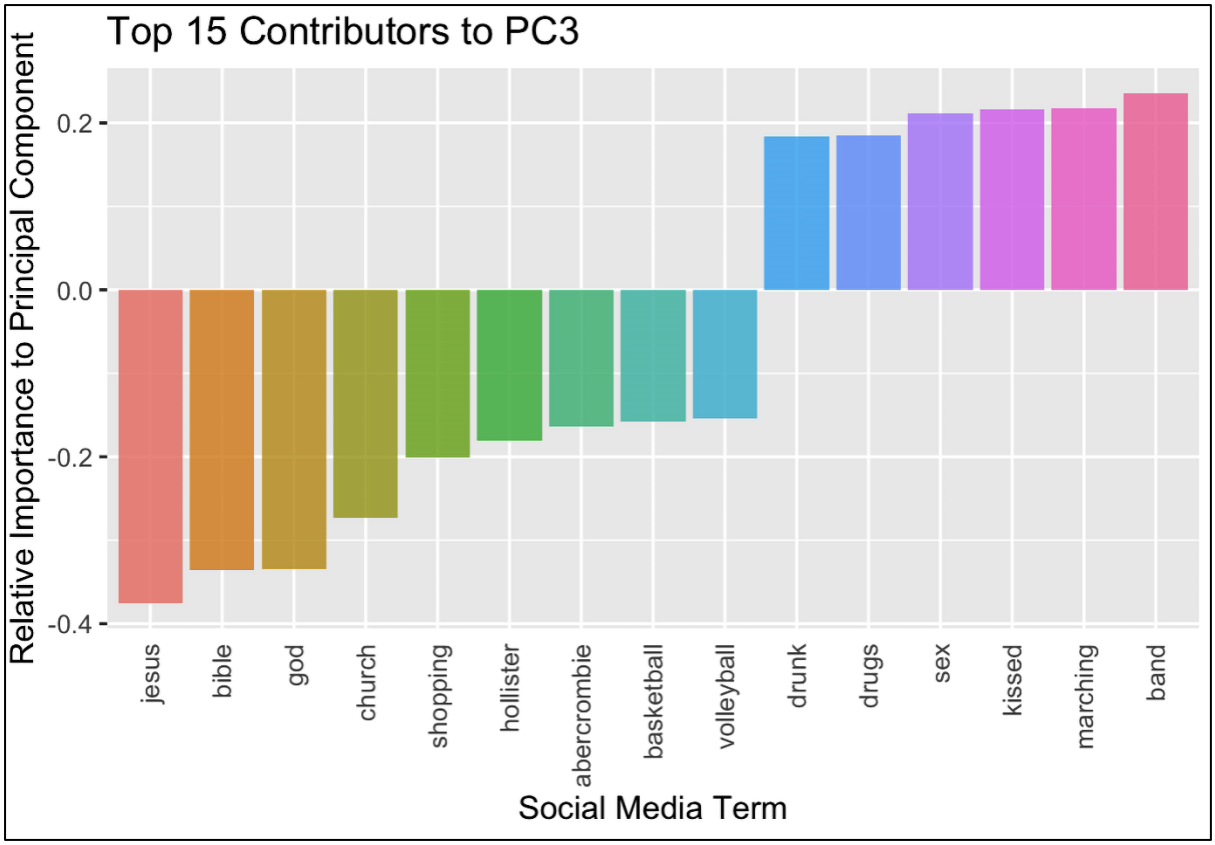

income
education

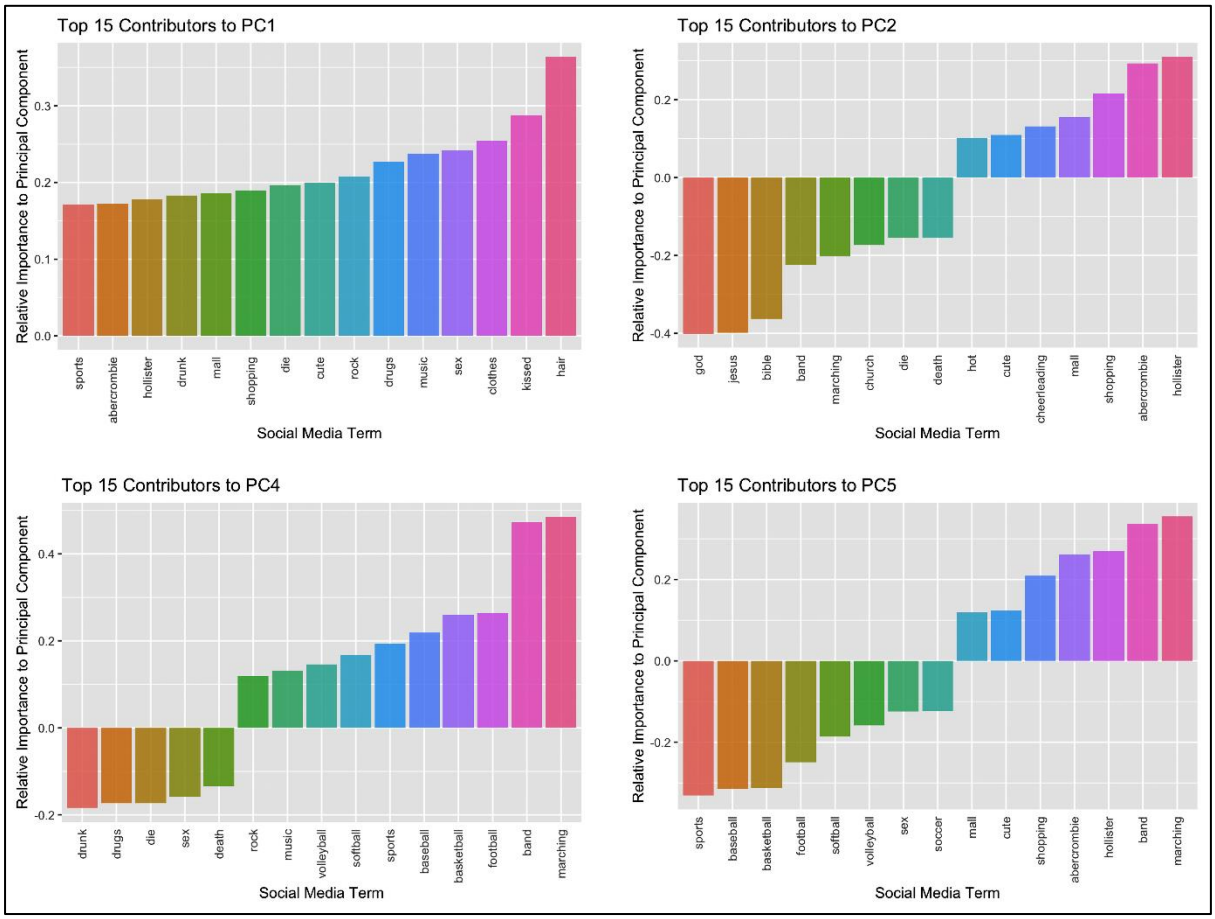

age
z
income
education
zip code
occupation
y
x


y
age
income
education
zip code
occupation
x


education
income
age
zip code
x
occupation

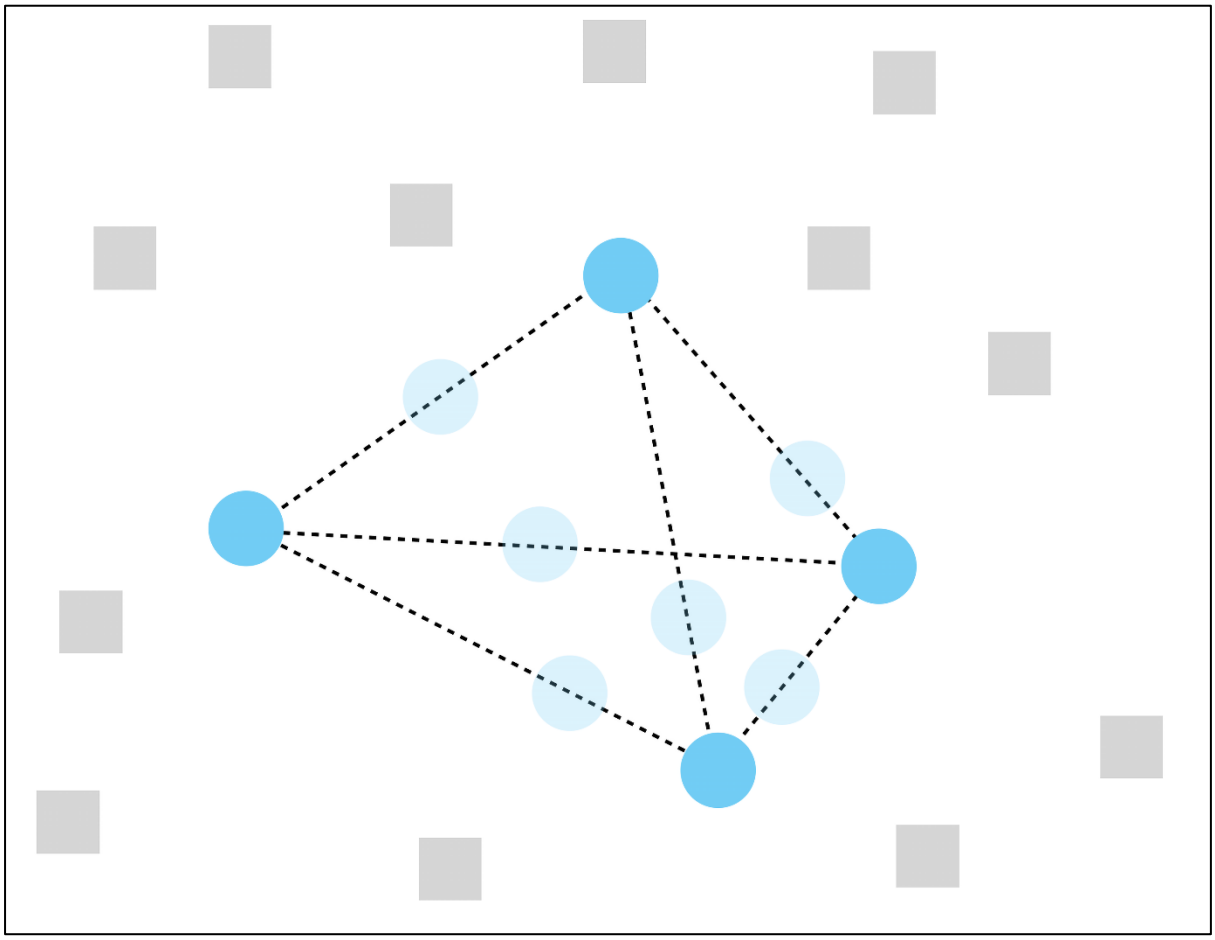

Scree Plot of SNS Data Principal Components
Variances
3.0
2.0
1.0
1
2
3
4
5
6
7
8
9
10


Top 15 Contributors to PC3
Relative Importance to Principal Component
0.2
0.0
-0.2
-0.4
jesus
bible
god
church
shopping
hollister
abercrombie
basketball
volleyball
drunk
drugs
sex
kissed
marching
band
Social Media Term

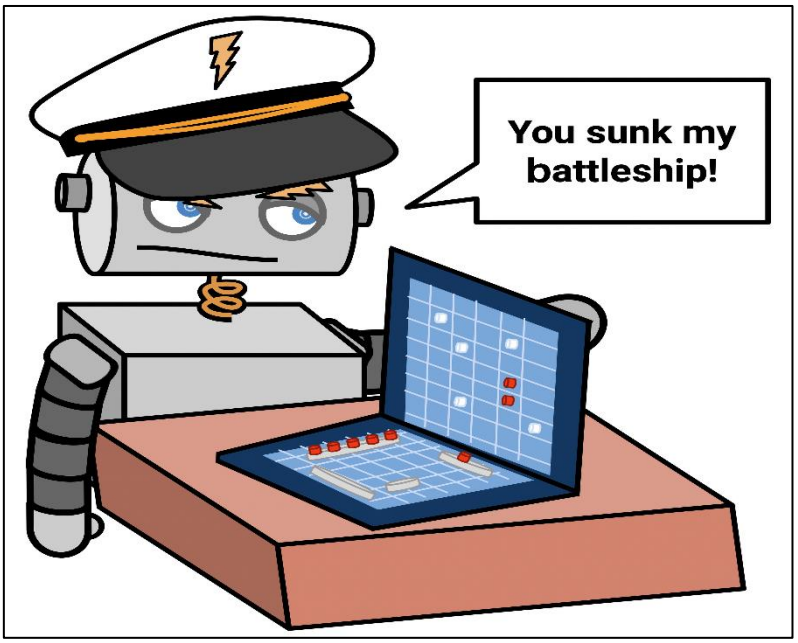

Top 15 Contributors to PC1
Relative Importance to Principal Component
0.3
0.2
0.1
0.0
sports
abercrombie
hollister
drunk
mall
shopping
die
cute
rock
drugs
music
sex
clothes
kissed
hair
Social Media Term
Top 15 Contributors to PC2
Relative Importance to Principal Component
0.2
0.0
-0.2
-0.4
god
jesus
bible
band
marching
church
die
death
hot
cute
cheerleading
mall
shopping
abercrombie
hollister
Social Media Term
Top 15 Contributors to PC4
Relative Importance to Principal Component
0.4
0.2
0.0
-0.2
drunk
drugs
die
sex
death
rock
music
volleyball
softball
sports
baseball
basketball
football
band
marching
Social Media Term
Top 15 Contributors to PC5
Relative Importance to Principal Component
0.2
0.0
-0.2
sports
baseball
basketball
football
softball
volleyball
sex
soccer
mall
cute
shopping
abercrombie
hollister
band
marching
Social Media Term

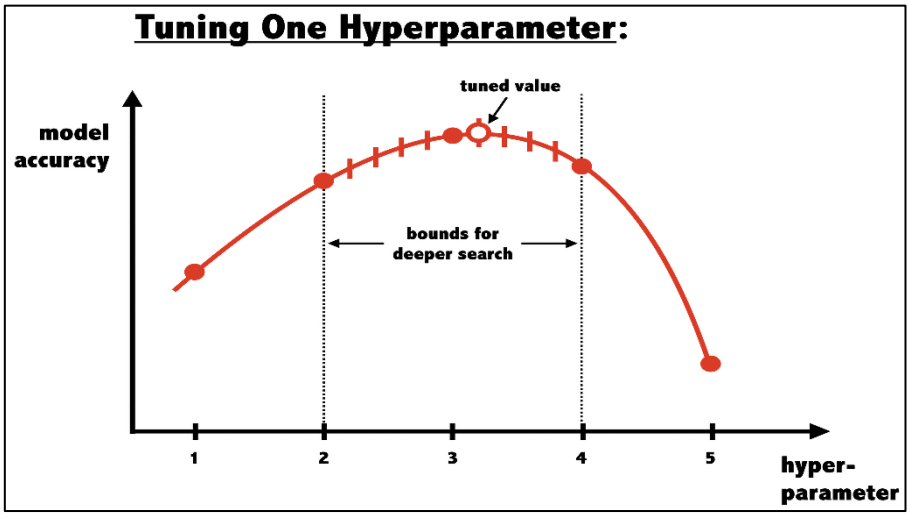
married	age	gender	country	emp_len	income	zipcode	creditscore
NA	*NA*	*NA*	USA	10	$54,000	90210	670
Yes	48	M	USA	6	$99,000	46656	780
Yes	37	F	USA	5	*NA*	*NA*	*NA*
No	*NA*	M	*NA*	*NA*	*NA*	32817	*NA*

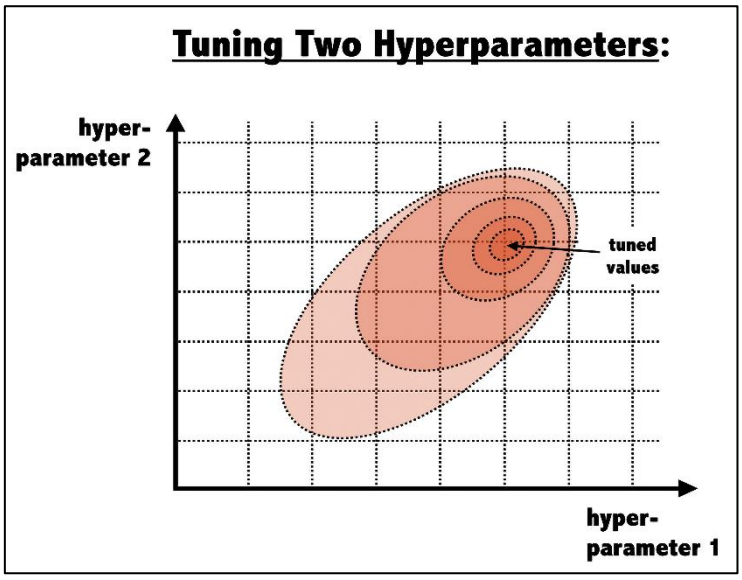
x1_mvi	x2_mvi	x3_mvi	x4_mvi	x5_mvi	x6_mvi	x7_mvi	x8_mvi
1	1	1	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	1	1	1
0	1	0	1	1	1	0	1



Chapter 14: Building Better Learners





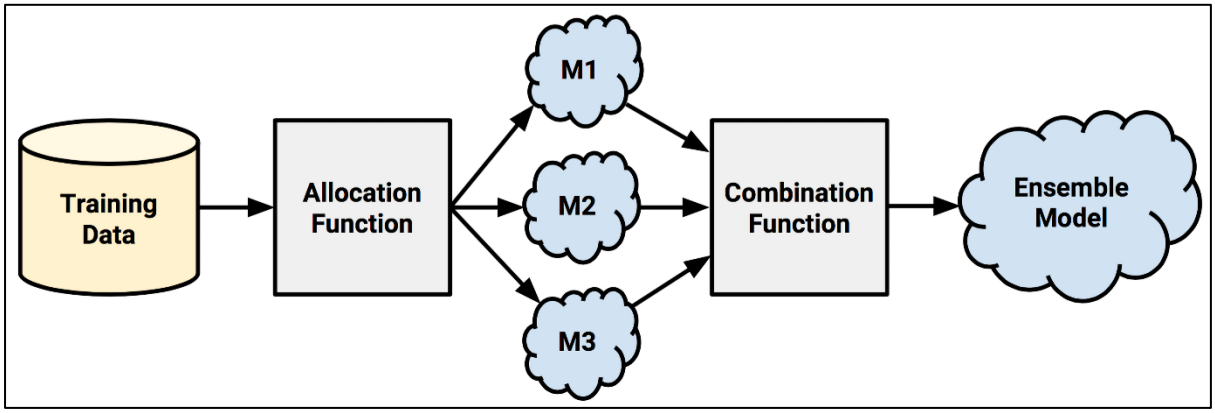


```
1000 samples
  16 predictor
   2 classes: 'no', 'yes'

No pre-processing
Resampling: Bootstrapped (25 reps)
Summary of sample sizes: 1000, 1000, 1000, 1000, 1000, 1000, ...
Resampling results across tuning parameters:

  model  winnow  trials  Accuracy   Kappa
  rules  FALSE     1      0.6918852  0.2749843
  rules  FALSE    10      0.7144478  0.3112184
  rules  FALSE    20      0.7270463  0.3344054
  rules   TRUE     1      0.6947856  0.2752201
  rules   TRUE    10      0.7144334  0.3150388
  rules   TRUE    20      0.7268154  0.3393965
  tree   FALSE     1      0.6909682  0.2569822
  tree   FALSE    10      0.7256597  0.2996244
  tree   FALSE    20      0.7322016  0.3157605
  tree    TRUE     1      0.6920217  0.2604578
  tree    TRUE    10      0.7279264  0.3099733
  tree    TRUE    20      0.7299631  0.3131074

Accuracy was used to select the optimal model using the largest value.
The final values used for the model were trials = 20, model = tree and winnow = FALSE.
```



Random forest syntax

Using the `randomForest()` function in the `randomForest` package

Building the classifier:

```
m <- randomForest(train, class, ntree = 500, mtry = sqrt(p))
```

- `train` is a data frame containing training data
- `class` is a factor vector with the class for each row in the training data
- `ntree` is an integer specifying the number of trees to grow
- `mtry` is an optional integer specifying the number of features to randomly select at each split (uses `sqrt(p)` by default, where `p` is the number of features in the data)

The function will return a random forest object that can be used to make predictions.

Making predictions:

```
p <- predict(m, test, type = "response")
```

- `m` is a model trained by the `randomForest()` function
- `test` is a data frame containing test data with the same features as the training data used to build the classifier
- `type` is either `"response"`, `"prob"`, or `"votes"` and is used to indicate whether the predictions vector should contain the predicted class, the predicted probabilities, or a matrix of vote counts, respectively

The function will return predictions according to the value of the `type` parameter.

Example:

```
credit_model <- randomForest(credit_train, loan_default)
credit_prediction <- predict(credit_model, credit_test)
```

Gradient Boosting Machine (GBM) syntax

Using the `gbm()` function in the `gbm` package

Building the classifier:

```
m <- gbm(target ~ predictors, data = mydata,
  distribution = "bernoulli", n.trees = 100, interaction.depth = 1,
  n.minobsinnode = 10, shrinkage = 0.1)
```

- `target` is the outcome in the `mydata` data frame to be modeled
- `predictors` is an R formula specifying the features in the `mydata` data frame to use for prediction
- `data` specifies the data frame in which the `target` and `predictors` variables can be found
- `distribution` is the form of target variable; can generally omit this to allow `gbm()` to determine it automatically
- `n.trees` is the number of boosting iterations (total number of trees to fit)
- `interaction.depth` is an integer specifying the maximum depth of each decision tree; deeper trees allow more interactions among predictors
- `n.minobsinnode` is an integer specifying the minimum number of observations in the trees' leaf nodes; smaller or larger values may lead to over- or under-fitting
- `shrinkage` refers to the learning rate, usually between 0.001 and 0.1; smaller values require more trees but may result in better models

The function will return a `gbm` object that can be used to make predictions.

Making predictions:

```
p <- predict(m, test, type = "link")
```

- `m` is a model trained by the `gbm()` function
- `test` is a data frame containing test data with the same features as the training data used to build the classifier
- `type` is either `"link"` or `"response"` and for classification models is used to indicate whether the predictions vector should contain the predicted class or the predicted probabilities, respectively. For numeric prediction this is unnecessary.

The function will return predictions according to the value of the `type` parameter.

Example:

```
credit_model <- gbm(default ~ ., data = credit_train)
credit_prediction <- predict(credit_model, credit_test,
                             type = "response")
```

XGBoost (XGB) syntax

Using the `xgboost()` function in the `xgboost` package

Setting model parameters:

```
params.xgb = list(objective = "binary:logistic",
  max_depth = 6, eta = 0.3, gammma = 0, colsample_bytree = 1,
  min_child_weight = 1, subsample = 1)
```

- `objective` is determined by the target to be modeled; `"binary:logistic"` is for binary classification; use `"multi:softprob"` for categorical outcomes, `"reg:squarederror"` for regression, or `"count:poisson"` for count data
- `max_depth` is between 0 and infinity and defines the maximum depth of any tree; larger values can find more specific patterns but risk overfitting
- `eta` is between 0 and 1 and affects the learning rate; low values limit overfitting, but increase training time and will require more boosting iterations (`nrounds`)
- `gamma` is between 0 and 1 and governs whether the algorithm will continue splitting; lower values can find more specific patterns but risk overfitting
- `colsample_bytree` is between 0 and 1 and defines the % of features that will be selected at random for each tree; use `colsample_bynode` to instead select random features at each split
- `min_child_weight` is between 0 and infinity and is analogous to the minimum examples needed to split; small values find more specific patterns but may overfit
- `subsample` is between 0 and 1 and defines the % of randomly selected examples for each iteration; small values may prevent overfitting but require more iterations

Note that the above includes only a subset of parameters and hyperparameters. See **`?xgboost`** or https://xgboost.readthedocs.io/en/latest/parameter.html for the full list.

Building the classifier:

```
m <- xgboost(params, data = mydata, label = mylabels, rounds = n)
```

- `params` is a list of XGBoost hyperparameters (as defined above)
- `data` is a matrix or sparse matrix with the features to be used for training
- `label` is a vector of target values to be used for training
- `rounds` is the maximum number of boosting iterations

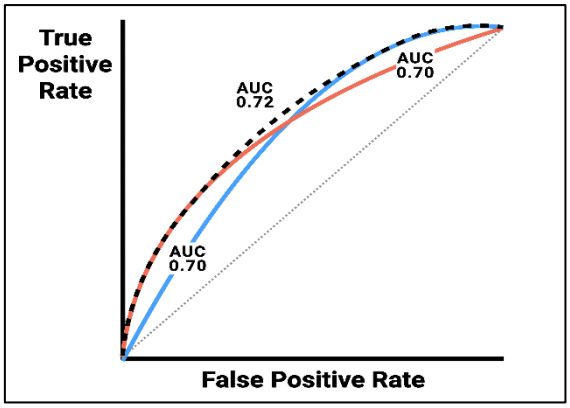
The function will return an `xgb` object that can be used to make predictions.

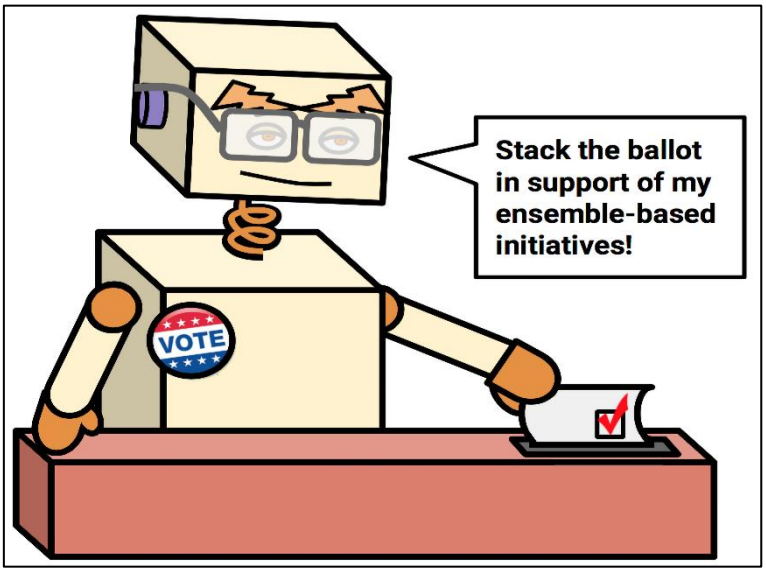
Making predictions:

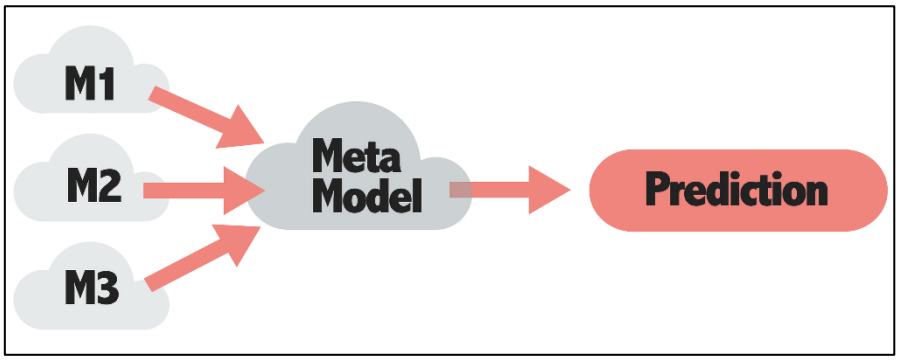
```
p <- predict(m, test)
```

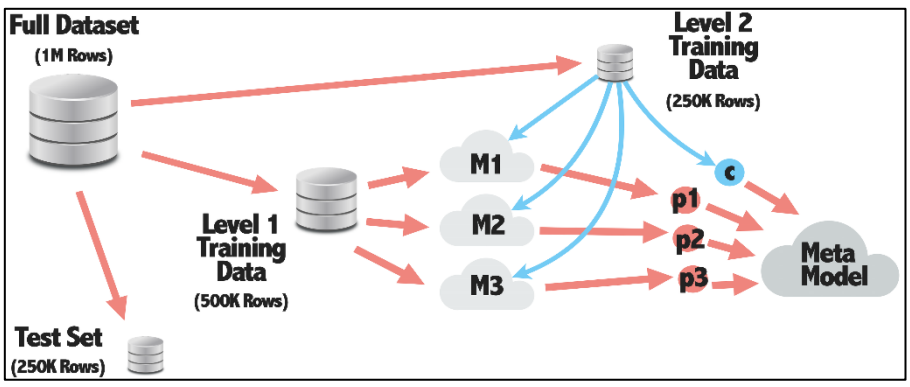
- `m` is a model trained by the `xgboost()` function
- `test` is a data frame containing test data in the same form as the training data

The function returns predicted probabilities (for classifiers) or values (for numeric models).

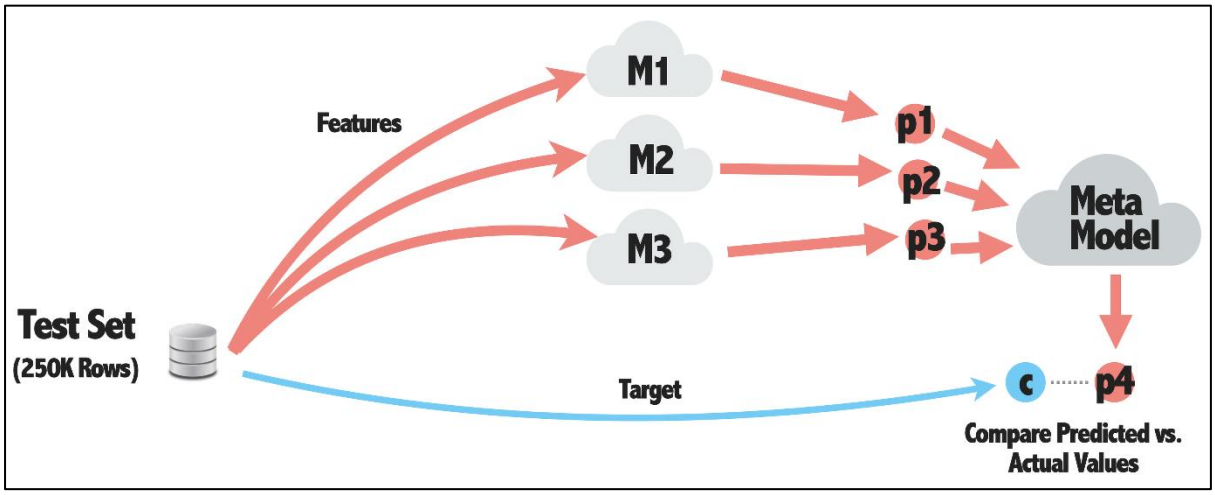

True Positive Rate
AUC 0.72
AUC 0.70
AUC 0.70
False Positive Rate

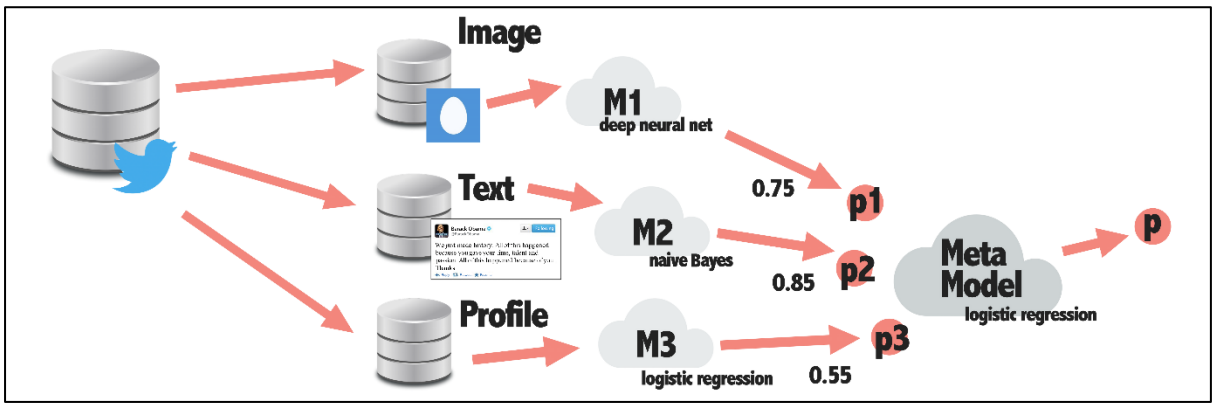

Stack the ballot in support of my ensemble-based initiatives!
VOTE


M1
M2
M3
Meta Model
Prediction

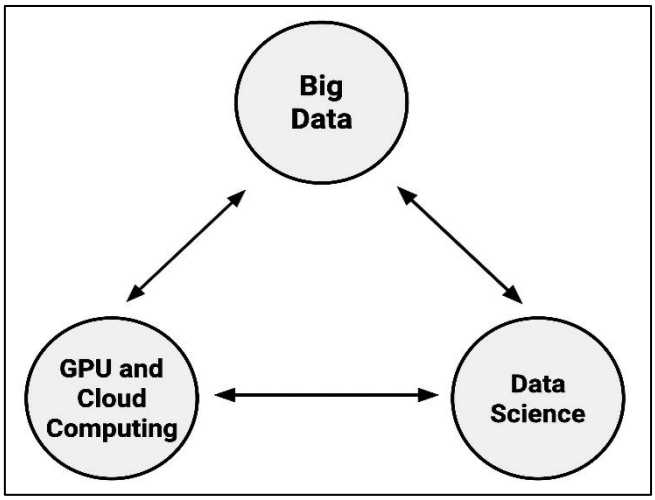

Full Dataset
(1M Rows)
Level 2 Training Data
(250K Rows)
Level 1 Training Data
(500K Rows)
M1
M2
M3
c
p1
p2
p3
Meta Model
Test Set
(250K Rows)

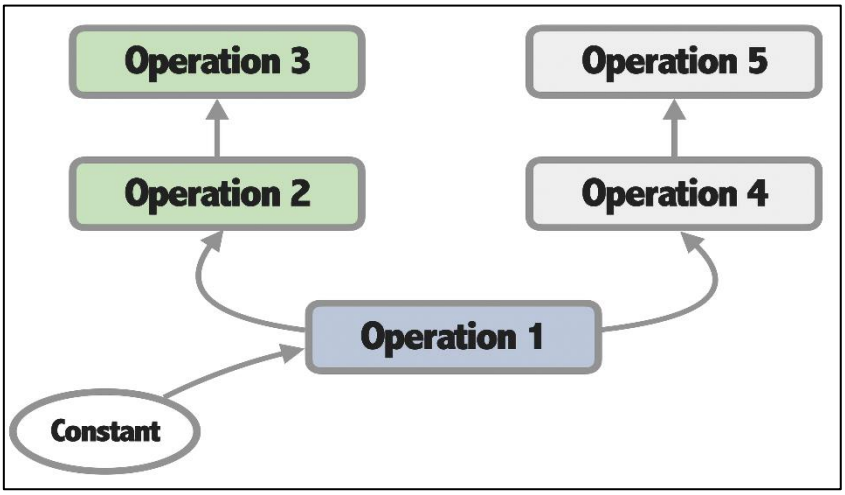
M_1 Prediction	M_2 Prediction	M_3 Prediction	Actual Value
Yes	Yes	Yes	Yes
No	No	Yes	Yes
No	Yes	Yes	Yes
Yes	No	No	No



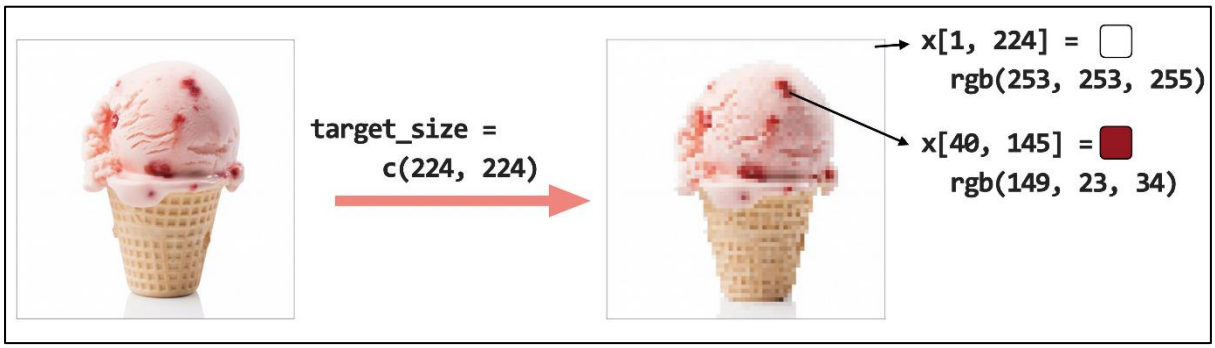


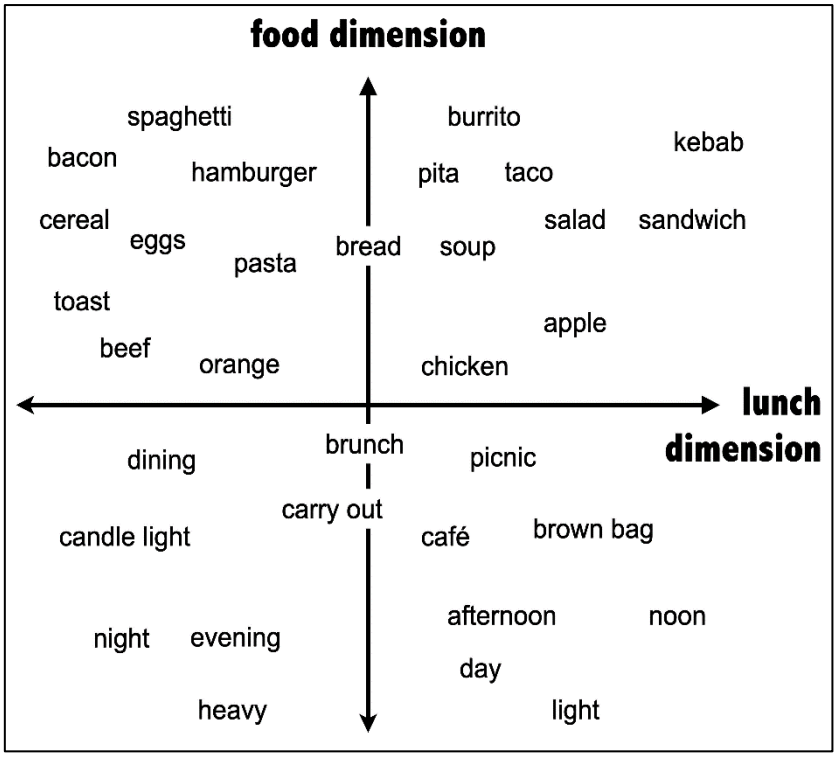
Chapter 15: Making Use of Big Data

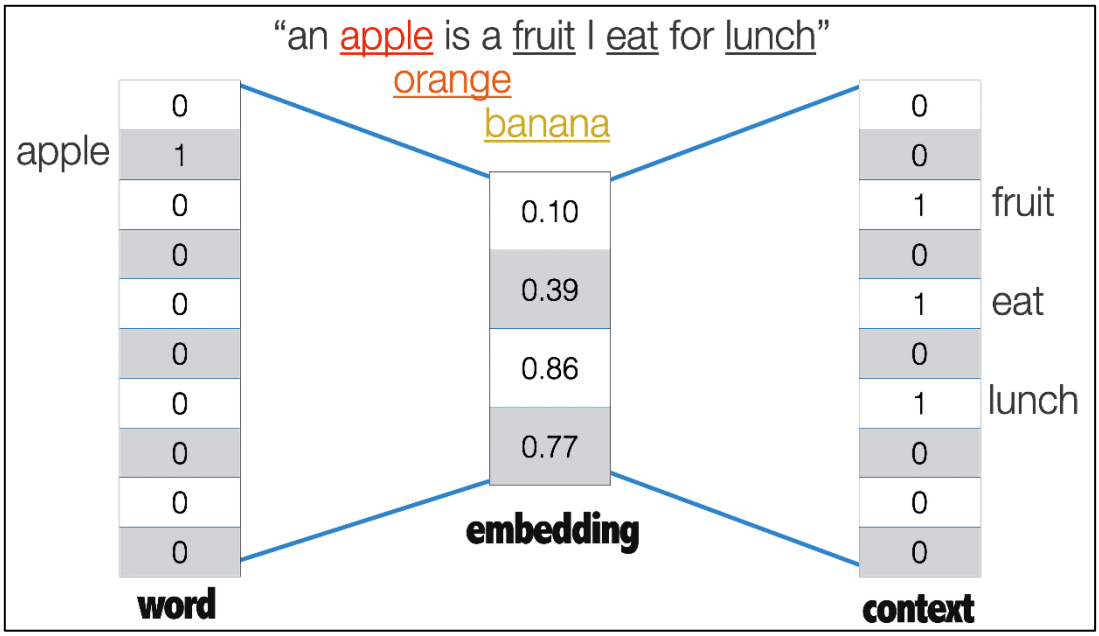


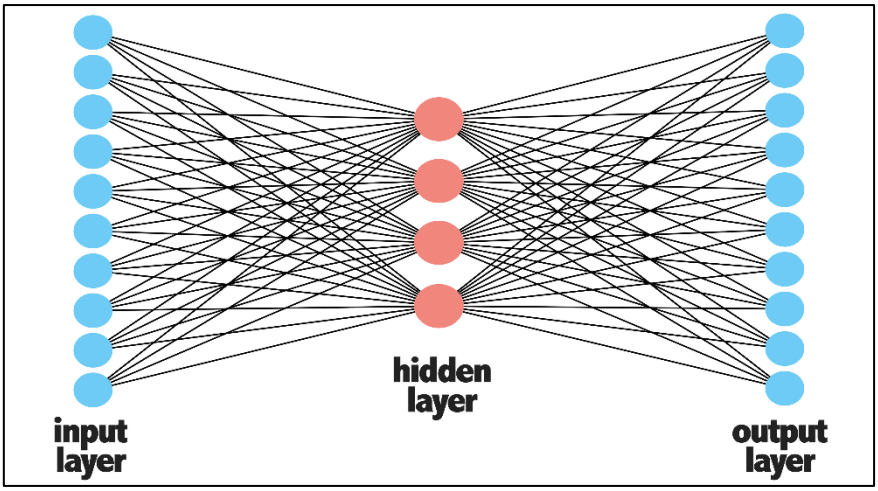


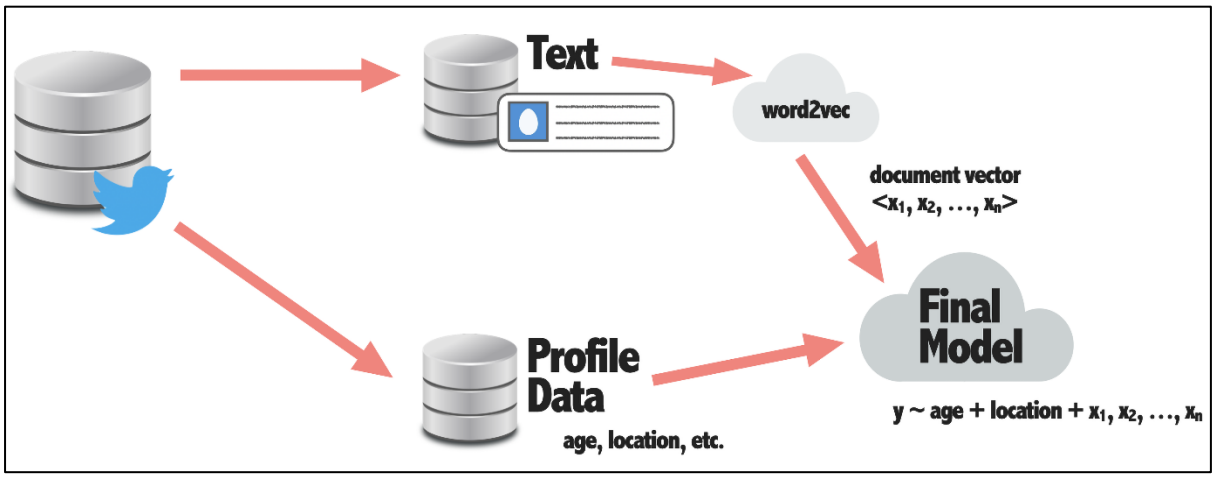


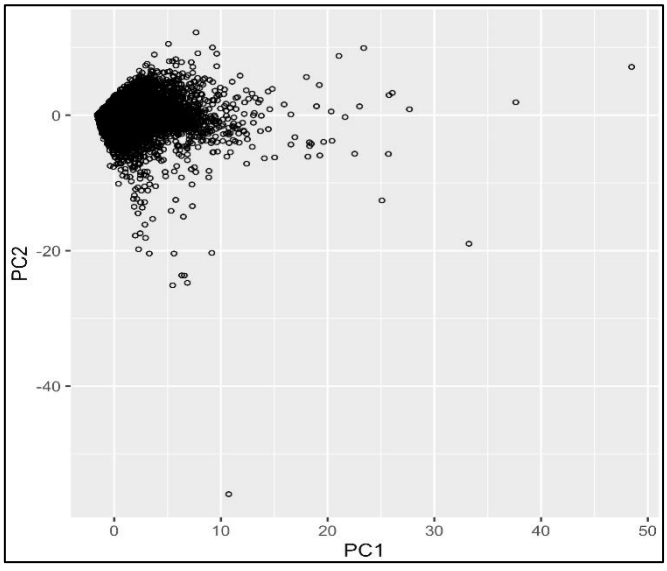


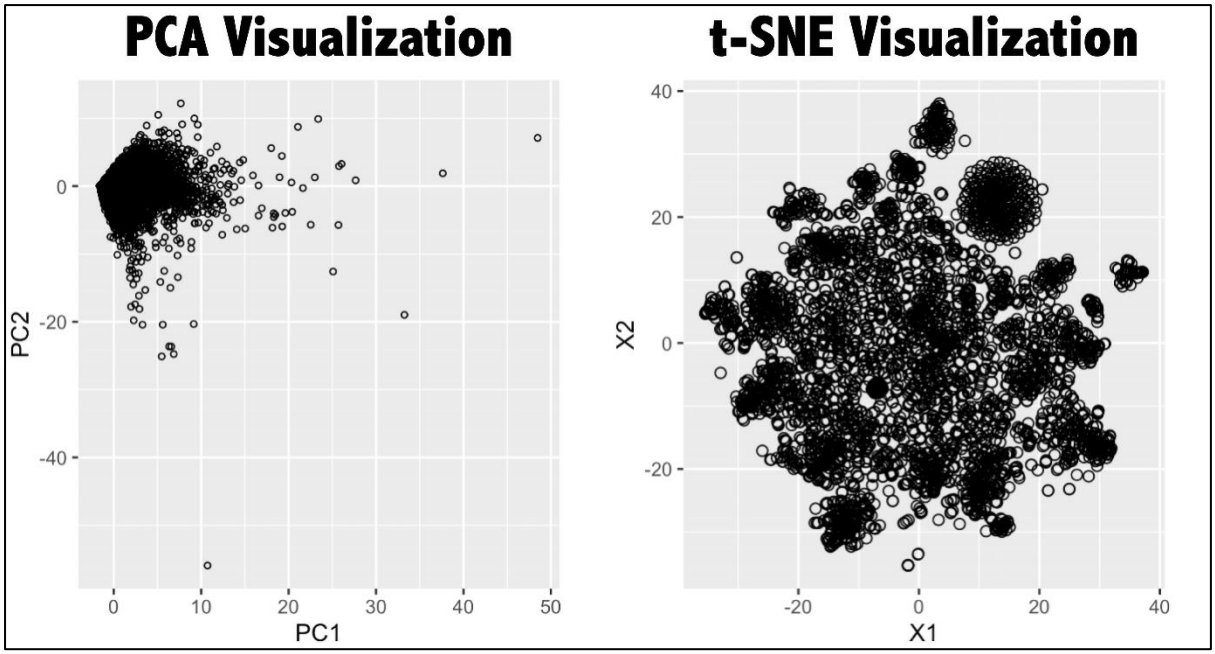

food dimension
spaghetti
burrito
kebab
bacon
hamburger
pita
taco
cereal
salad
sandwich
eggs
bread
soup
pasta
toast
apple
beef
orange
chicken
lunch
dimension
brunch
dining
picnic
carry out
candle light
café
brown bag
afternoon
noon
night
evening
day
heavy
light

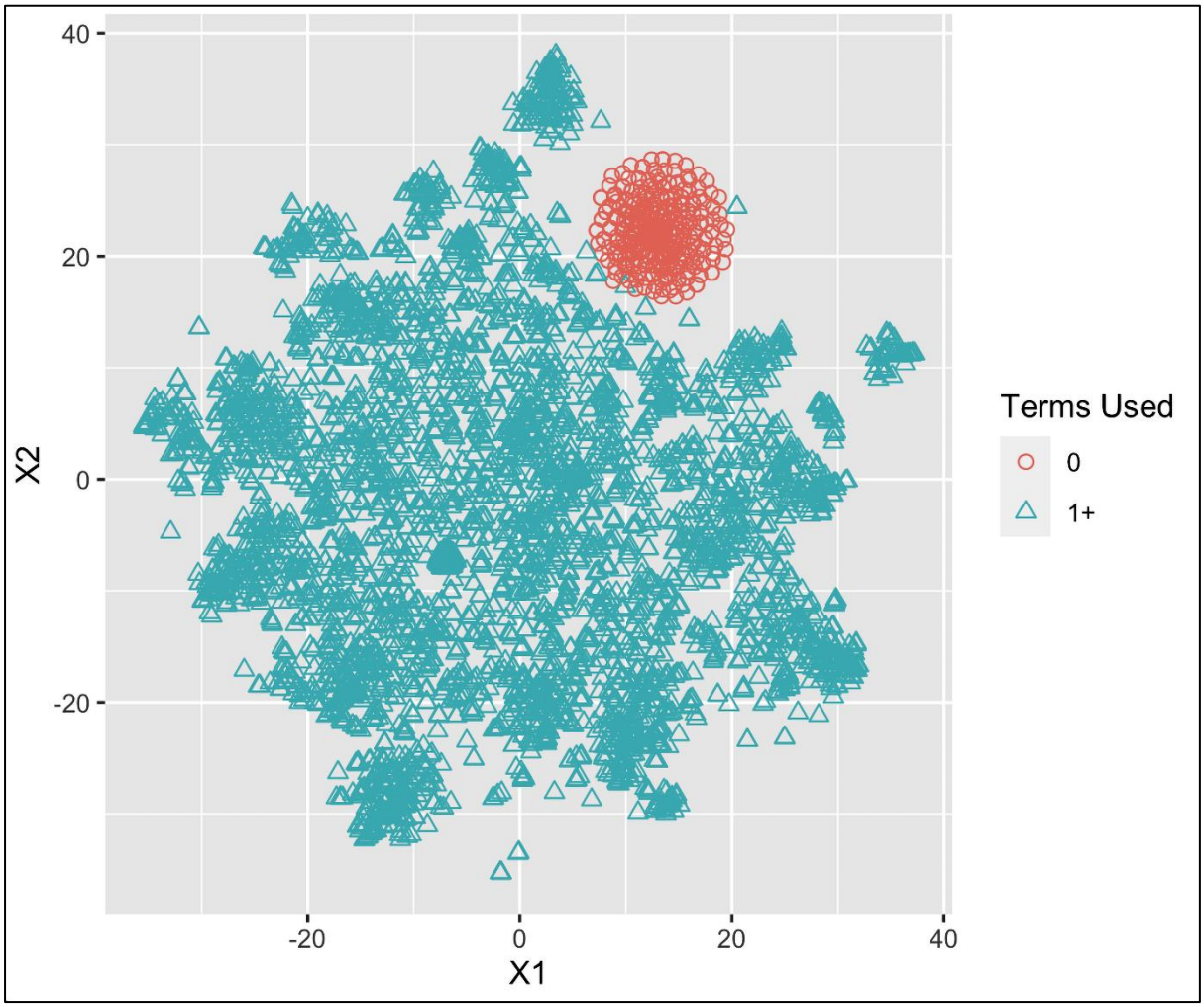

"an apple is a fruit I eat for lunch"
orange
banana
apple
0
1
0
0
0
0
0
0
0
0
0.10
0.39
0.86
0.77
embedding
0
0
1
0
1
0
1
0
0
0
fruit
eat
lunch
word
context

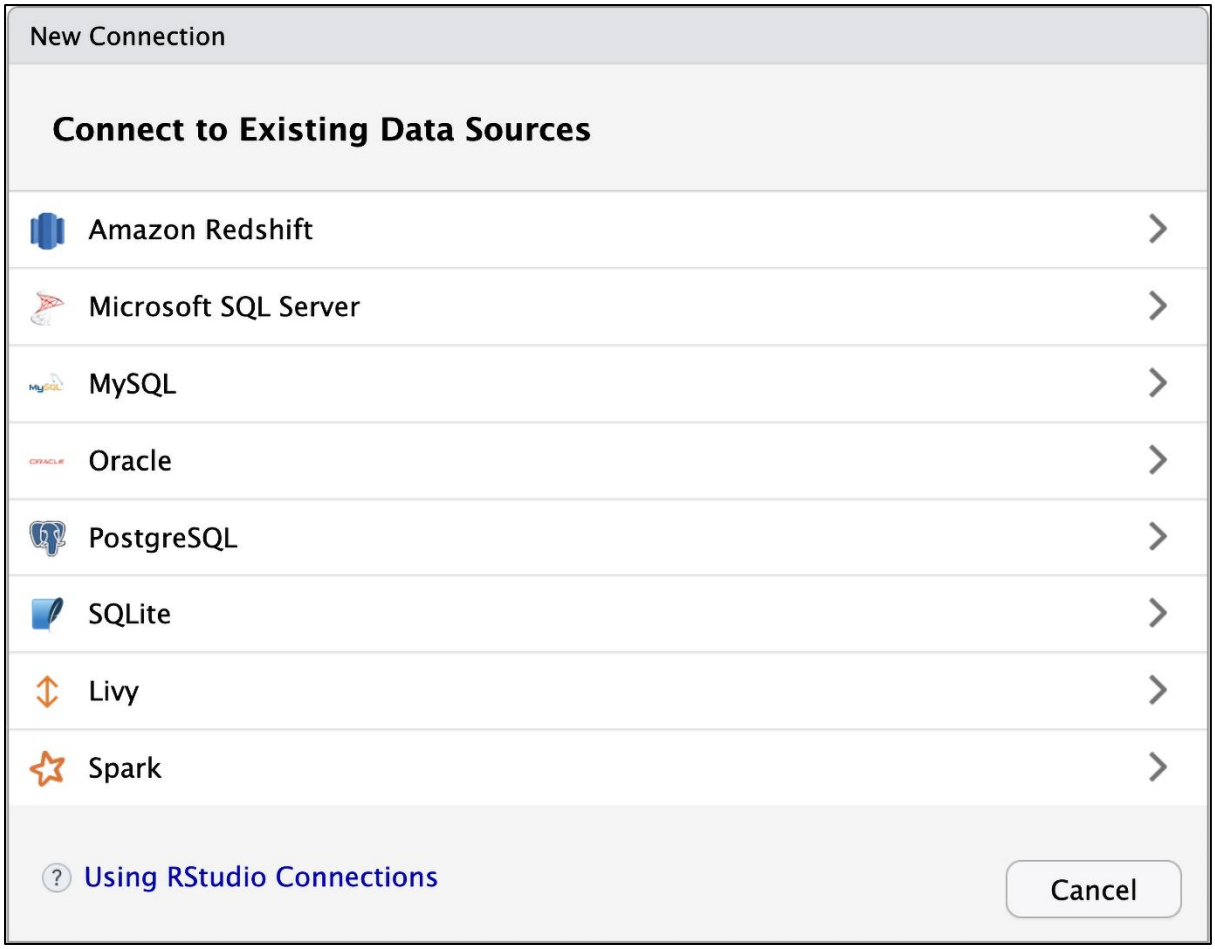

input
layer
hidden
layer
output
layer

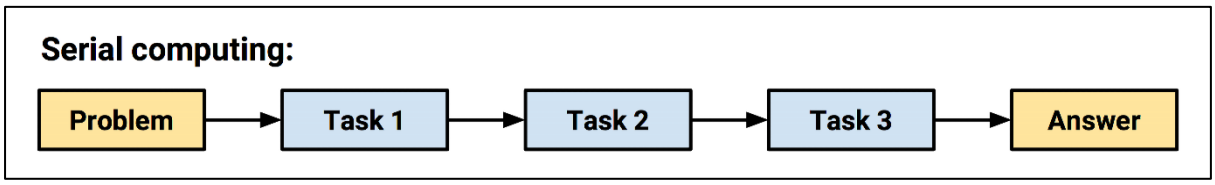

Text
word2vec
document vector
<x1, x2, ..., xn>
Final
Model
Profile
Data
age, location, etc.
y ~ age + location + x1, x2, ..., xn

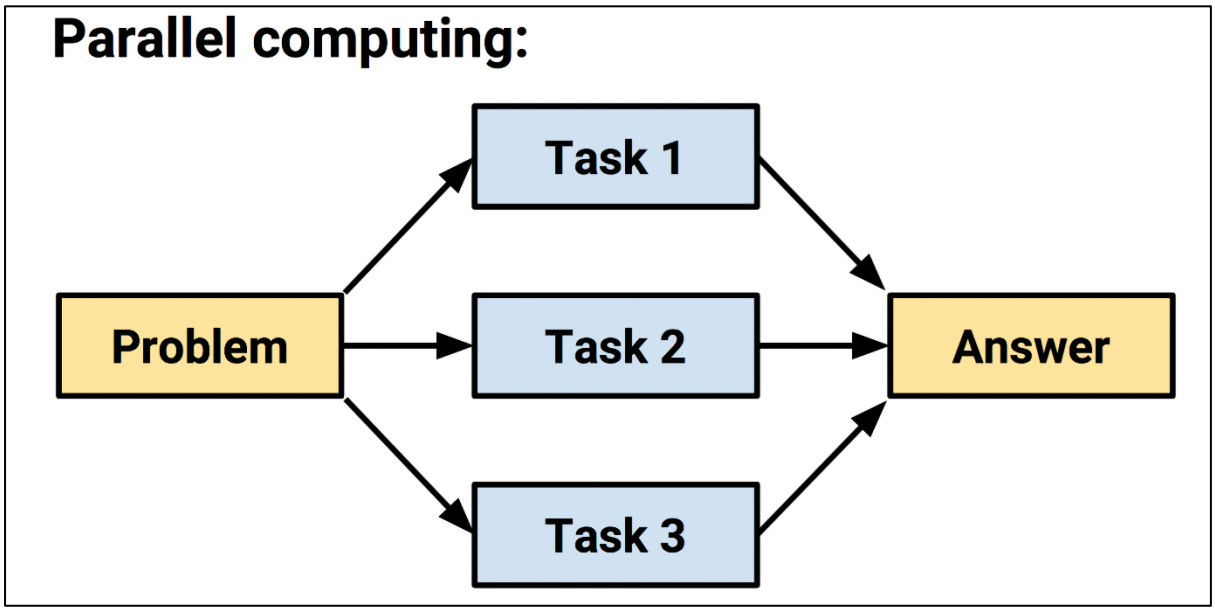

0
-20
-40
PC2
0
10
20
30
40
50
PC1

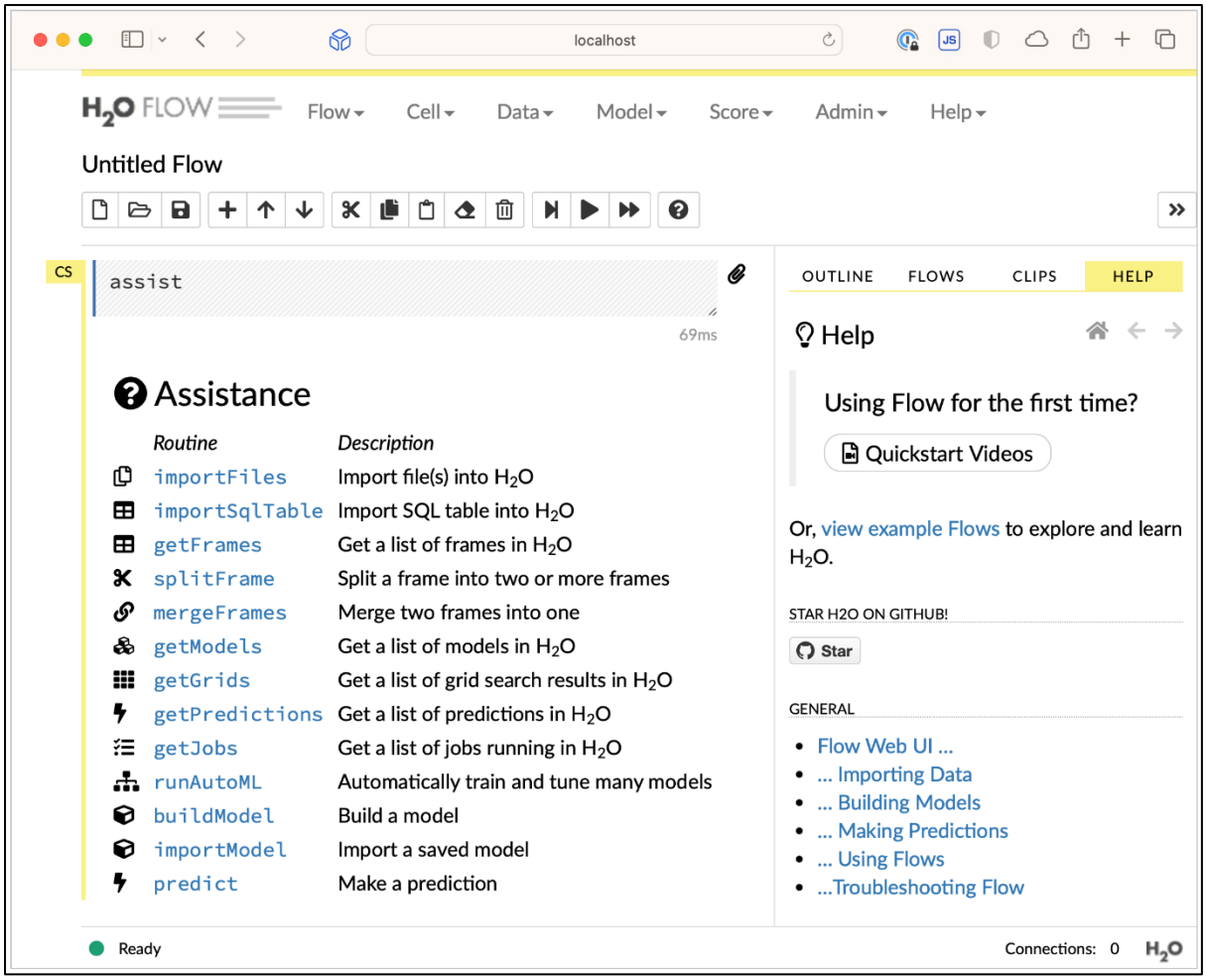

PCA Visualization
t-SNE Visualization
PC2
PC1
X2
X1

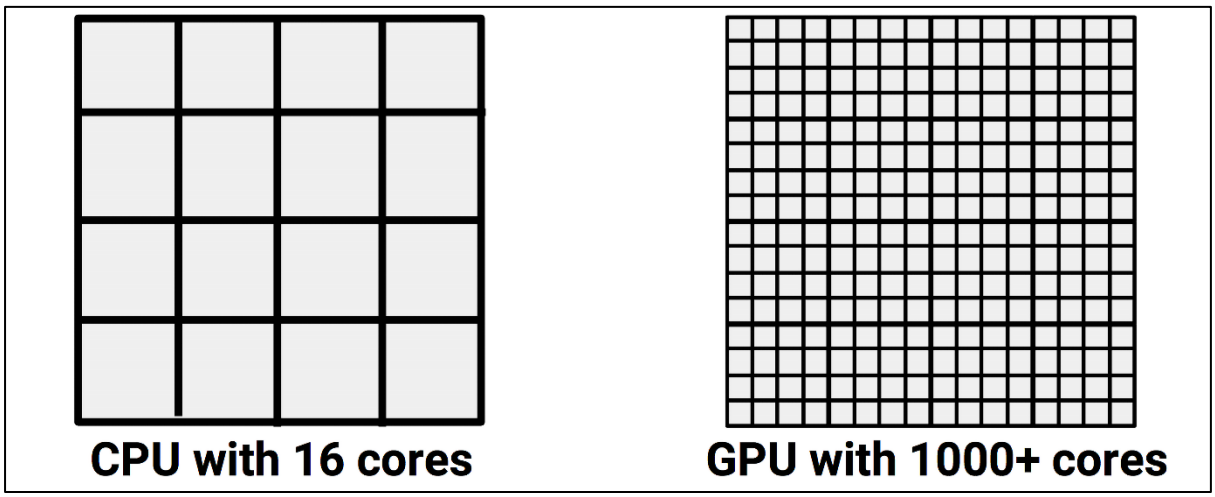

X2
X1
Terms Used
0
1+


New Connection
Connect to Existing Data Sources
Amazon Redshift
Microsoft SQL Server
MySQL
Oracle
PostgreSQL
SQLite
Livy
Spark
Using RStudio Connections
Cancel


Serial computing:
Problem
Task 1
Task 2
Task 3
Answer


Parallel computing:
Task 1
Problem
Task 2
Answer
Task 3


localhost
H2O FLOW
Flow
Cell
Data
Model
Score
Admin
Help
Untitled Flow
CS
assist
69ms
Assistance
Routine
Description
importFiles
Import file(s) into H2O
importSqlTable
Import SQL table into H2O
getFrames
Get a list of frames in H2O
splitFrame
Split a frame into two or more frames
mergeFrames
Merge two frames into one
getModels
Get a list of models in H2O
getGrids
Get a list of grid search results in H2O
getPredictions
Get a list of predictions in H2O
getJobs
Get a list of jobs running in H2O
runAutoML
Automatically train and tune many models
buildModel
Build a model
importModel
Import a saved model
predict
Make a prediction
OUTLINE
FLOWS
CLIPS
HELP
Help
Using Flow for the first time?
Quickstart Videos
Or, view example Flows to explore and learn H2O.
STAR H2O ON GITHUB!
Star
GENERAL
Flow Web UI ...
... Importing Data
... Building Models
... Making Predictions
... Using Flows
...Troubleshooting Flow
Ready
Connections: 0
H2O


CPU with 16 cores
GPU with 1000+ cores