Data Preparation

MACHINE LEARNING WITH PYSPARK

Andrew Collier
Data Scientist, Exegetic Analytics

Do you need all of those columns?

```
+-----+-------+-------+------+----+----+------+------+----+-----------+
|maker|  model| origin|  type| cyl|size|weight|length| rpm|consumption|
+-----+-------+-------+------+----+----+------+------+----+-----------+
|Mazda|   RX-7|non-USA|Sporty|null| 1.3|  2895| 169.0|6500|       9.41|
|  Geo|  Metro|non-USA| Small|   3| 1.0|  1695| 151.0|5700|        4.7|
| Ford|Festiva|    USA| Small|   4| 1.3|  1845| 141.0|5000|       7.13|
+-----+-------+-------+------+----+----+------+------+----+-----------+
```

Remove the `maker` and `model` fields.

Dropping columns

```
# Either drop the columns you don't want...
cars = cars.drop('maker', 'model')

# ... or select the columns you want to retain.
cars = cars.select('origin', 'type', 'cyl', 'size', 'weight', 'length', 'rpm', 'consumption')
```

```
+-------+------+----+----+------+------+----+-----------+
| origin|  type| cyl|size|weight|length| rpm|consumption|
+-------+------+----+----+------+------+----+-----------+
|non-USA|Sporty|null| 1.3|  2895| 169.0|6500|       9.41|
|non-USA| Small|   3| 1.0|  1695| 151.0|5700|        4.7|
|    USA| Small|   4| 1.3|  1845| 141.0|5000|       7.13|
+-------+------+----+----+------+------+----+-----------+
```

Filtering out missing data

```
# How many missing values?
cars.filter('cyl IS NULL').count()
```

```
1
```

Drop records with missing values in the `cylinders` column.

```
cars = cars.filter('cyl IS NOT NULL')
```

Drop records with missing values in *any* column.

```
cars = cars.dropna()
```

Mutating columns

```
from pyspark.sql.functions import round

# Create a new 'mass' column
cars = cars.withColumn('mass', round(cars.weight / 2.205, 0))
# Convert length to metres
cars = cars.withColumn('length', round(cars.length * 0.0254, 3))
```

```
+-------+-----+---+----+------+------+----+-----------+-----+
| origin| type|cyl|size|weight|length| rpm|consumption| mass|
+-------+-----+---+----+------+------+----+-----------+-----+
|non-USA|Small|  3| 1.0|  1695| 3.835|5700|        4.7|769.0|
|    USA|Small|  4| 1.3|  1845| 3.581|5000|       7.13|837.0|
|non-USA|Small|  3| 1.3|  1965| 4.089|6000|       5.47|891.0|
+-------+-----+---+----+------+------+----+-----------+-----+
```

Indexing categorical data

```
from pyspark.ml.feature import StringIndexer

indexer = StringIndexer(inputCol='type',
                        outputCol='type_idx')


# Assign index values to strings
indexer = indexer.fit(cars)


# Create column with index values
cars = indexer.transform(cars)
```

```
+-------+--------+
|   type|type_idx|
+-------+--------+
|Midsize|     0.0| <- most frequent value
|  Small|     1.0|
|Compact|     2.0|
| Sporty|     3.0|
|  Large|     4.0|
|    Van|     5.0| <- least frequent value
+-------+--------+
```

Use `stringOrderType` to change order.

Indexing country of origin

```
# Index country of origin:
#
# USA     -> 0
# non-USA -> 1
#
cars = StringIndexer(
  inputCol="origin",
  outputCol="label"
).fit(cars).transform(cars)
```

```
+-------+-----+
| origin|label|
+-------+-----+
|    USA|  0.0|
|non-USA|  1.0|
+-------+-----+
```

Assembling columns

Use a vector assembler to transform the data.

```
from pyspark.ml.feature import VectorAssembler

assembler = VectorAssembler(inputCols=['cyl', 'size'], outputCol='features')
assembler.transform(cars)
```

```
+---+----+---------+
|cyl|size| features|
+---+----+---------+
|  3| 1.0|[3.0,1.0]|
|  4| 1.3|[4.0,1.3]|
|  3| 1.3|[3.0,1.3]|
+---+----+---------+
```

Let's practice!

MACHINE LEARNING WITH PYSPARK

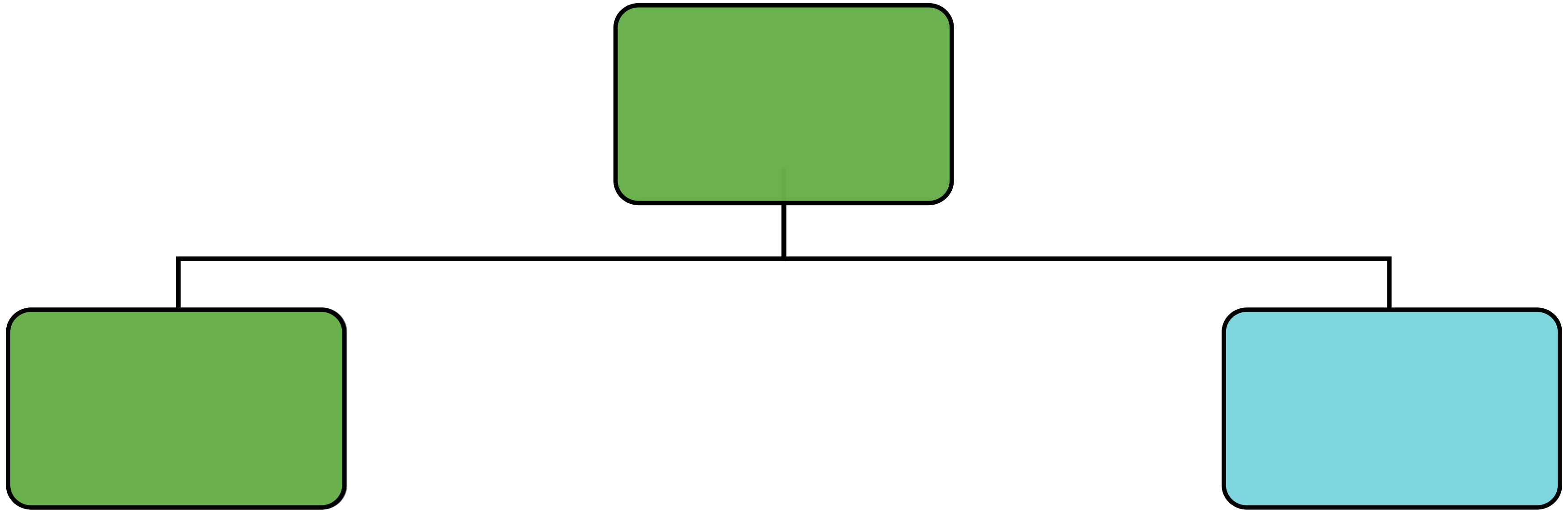
Decision Tree

MACHINE LEARNING WITH PYSPARK

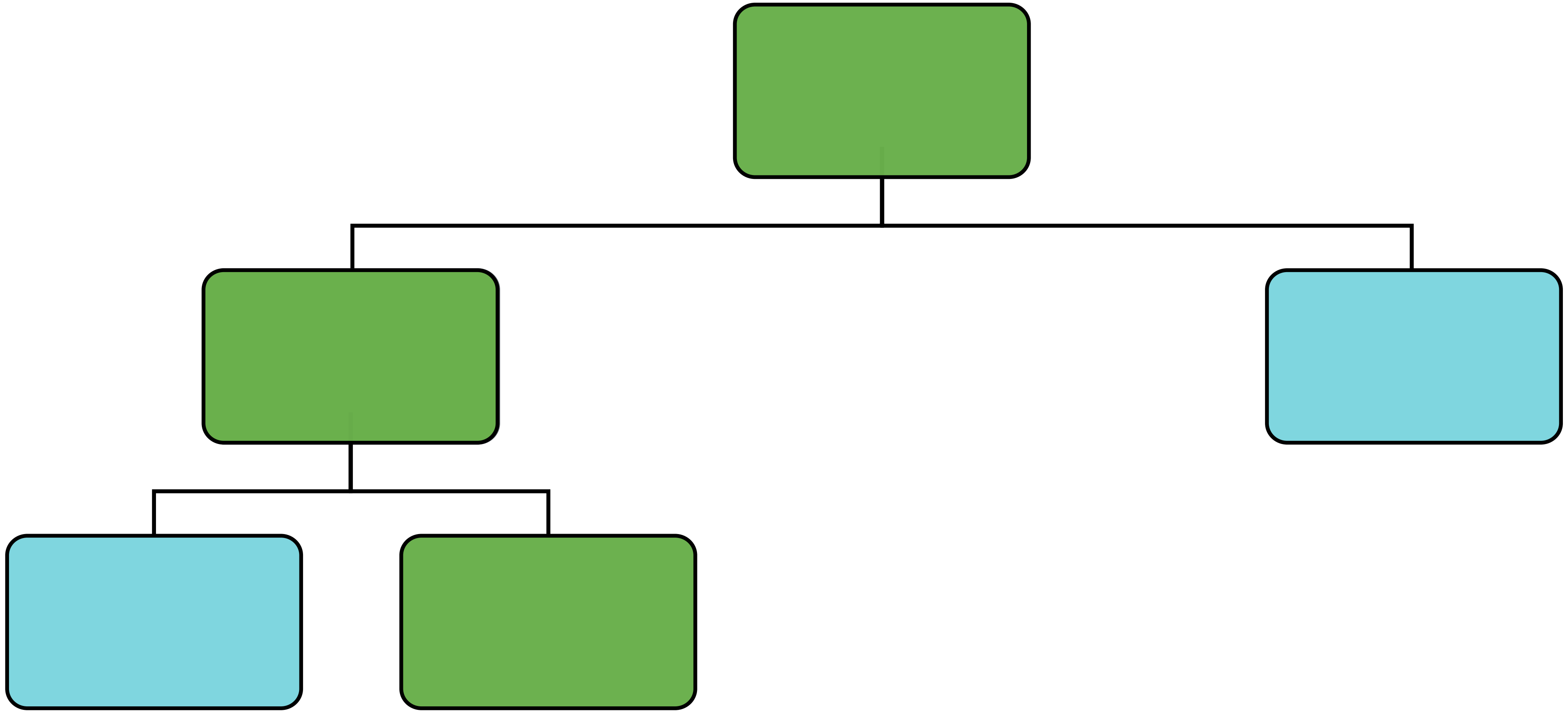
Andrew Collier
Data Scientist, Exegetic Analytics

Anatomy of a Decision Tree: Root node

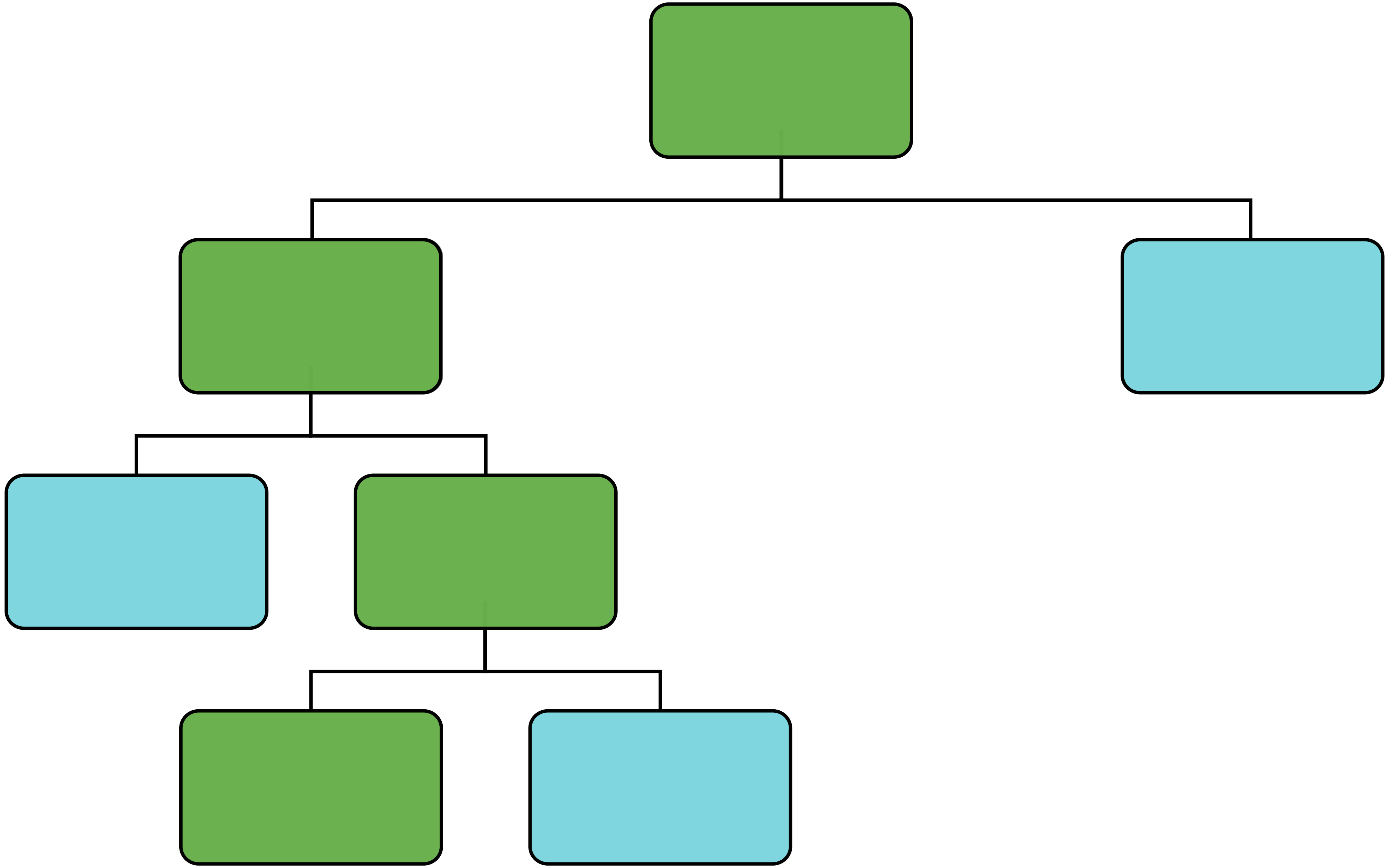
Anatomy of a Decision Tree: First split



Anatomy of a Decision Tree: Second split



Anatomy of a Decision Tree: Third split



Classifying cars

Classify cars according to country of manufacture.

```
+---+----+------+------+----+-----------+----------------------------------+-----+
|cyl|size|mass  |length|rpm |consumption|features                          |label|
+---+----+------+------+----+-----------+----------------------------------+-----+
|6  |3.0 |1451.0|4.775 |5200|9.05       |[6.0,3.0,1451.0,4.775,5200.0,9.05]|1.0  |
|4  |2.2 |1129.0|4.623 |5200|6.53       |[4.0,2.2,1129.0,4.623,5200.0,6.53]|0.0  |
|4  |2.2 |1399.0|4.547 |5600|7.84       |[4.0,2.2,1399.0,4.547,5600.0,7.84]|1.0  |
|4  |1.8 |1147.0|4.343 |6500|7.84       |[4.0,1.8,1147.0,4.343,6500.0,7.84]|0.0  |
|4  |1.6 |1111.0|4.216 |5750|9.05       |[4.0,1.6,1111.0,4.216,5750.0,9.05]|0.0  |
+---+----+------+------+----+-----------+----------------------------------+-----+

label = 0 -> manufactured in the USA
      = 1 -> manufactured elsewhere
```

Split train/test

Split data into training and testing sets.

```
# Specify a seed for reproducibility
cars_train, cars_test = cars.randomSplit([0.8, 0.2], seed=23)
```

Two DataFrames: `cars_train` and `cars_test` .

```
[cars_train.count(), cars_test.count()]
```

```
[79, 13]
```

Build a Decision Tree model

```
from pyspark.ml.classification import DecisionTreeClassifier
```

Create a Decision Tree classifier.

```
tree = DecisionTreeClassifier()
```

Learn from the training data.

```
tree_model = tree.fit(cars_train)
```

Evaluating

Make predictions on the testing data and compare to known values.

```
prediction = tree_model.transform(cars_test)
```

```
+-----+----------+----------------------------------------+
|label|prediction|probability                             |
+-----+----------+----------------------------------------+
|1.0  |0.0       |[0.9615384615384616,0.0384615384615385]|
|1.0  |1.0       |[0.2222222222222222,0.7777777777777778]|
|1.0  |1.0       |[0.2222222222222222,0.7777777777777778]|
|0.0  |0.0       |[0.9615384615384616,0.0384615384615385]|
|1.0  |1.0       |[0.2222222222222222,0.7777777777777778]|
+-----+----------+----------------------------------------+
```

Confusion matrix

A confusion matrix is a table which describes performance of a model on testing data.

```
prediction.groupBy("label", "prediction").count().show()
```

```
+-----+----------+-----+
|label|prediction|count|
+-----+----------+-----+
|  1.0|       1.0|    8| <- True positive  (TP)
|  0.0|       1.0|    2| <- False positive (FP)
|  1.0|       0.0|    3| <- False negative (FN)
|  0.0|       0.0|    6| <- True negative  (TN)
+-----+----------+-----+
```

Accuracy = (TN + TP) / (TN + TP + FN + FP) — proportion of correct predictions.

Let's build Decision Trees!

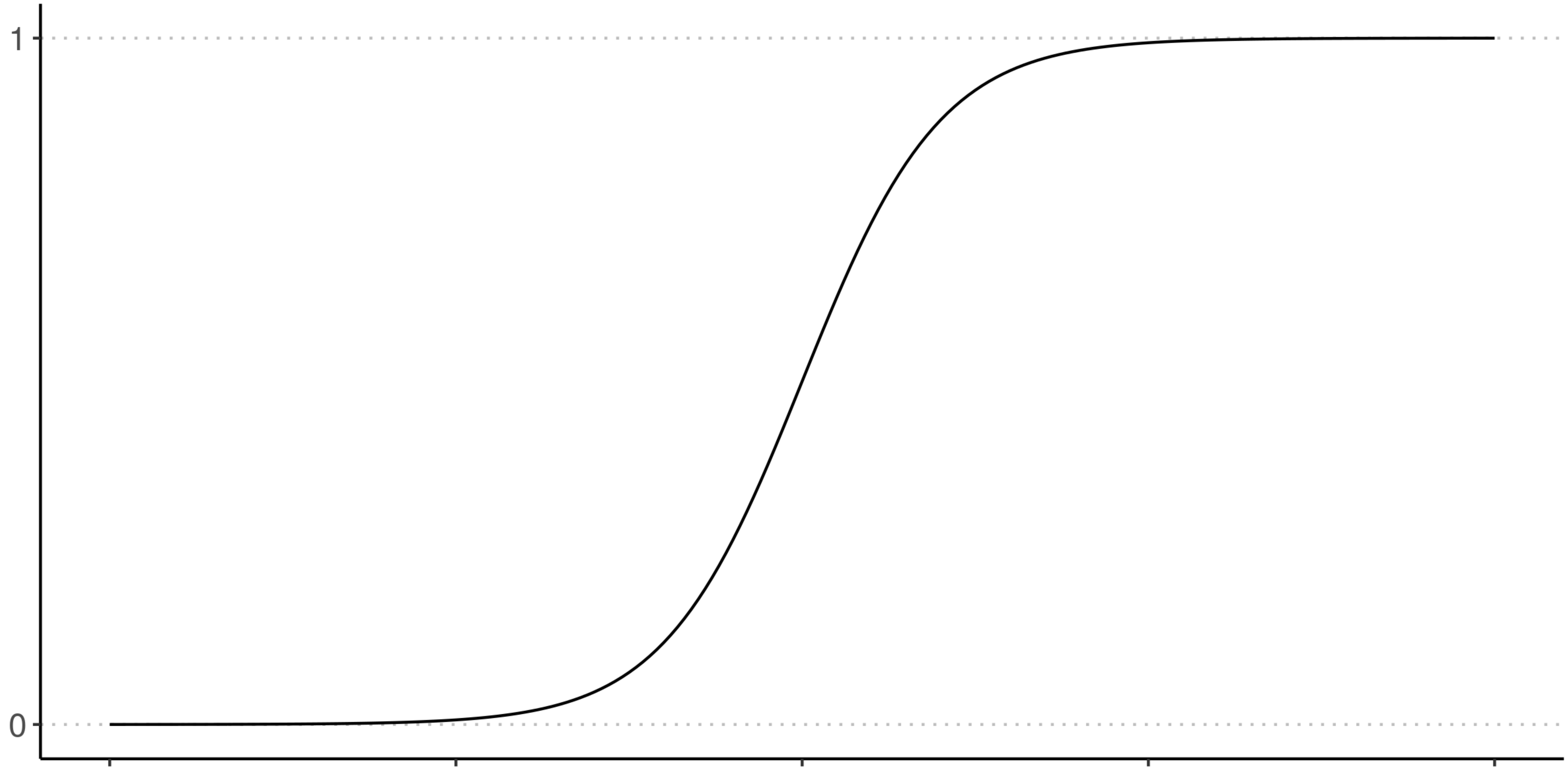
MACHINE LEARNING WITH PYSPARK

Logistic Regression

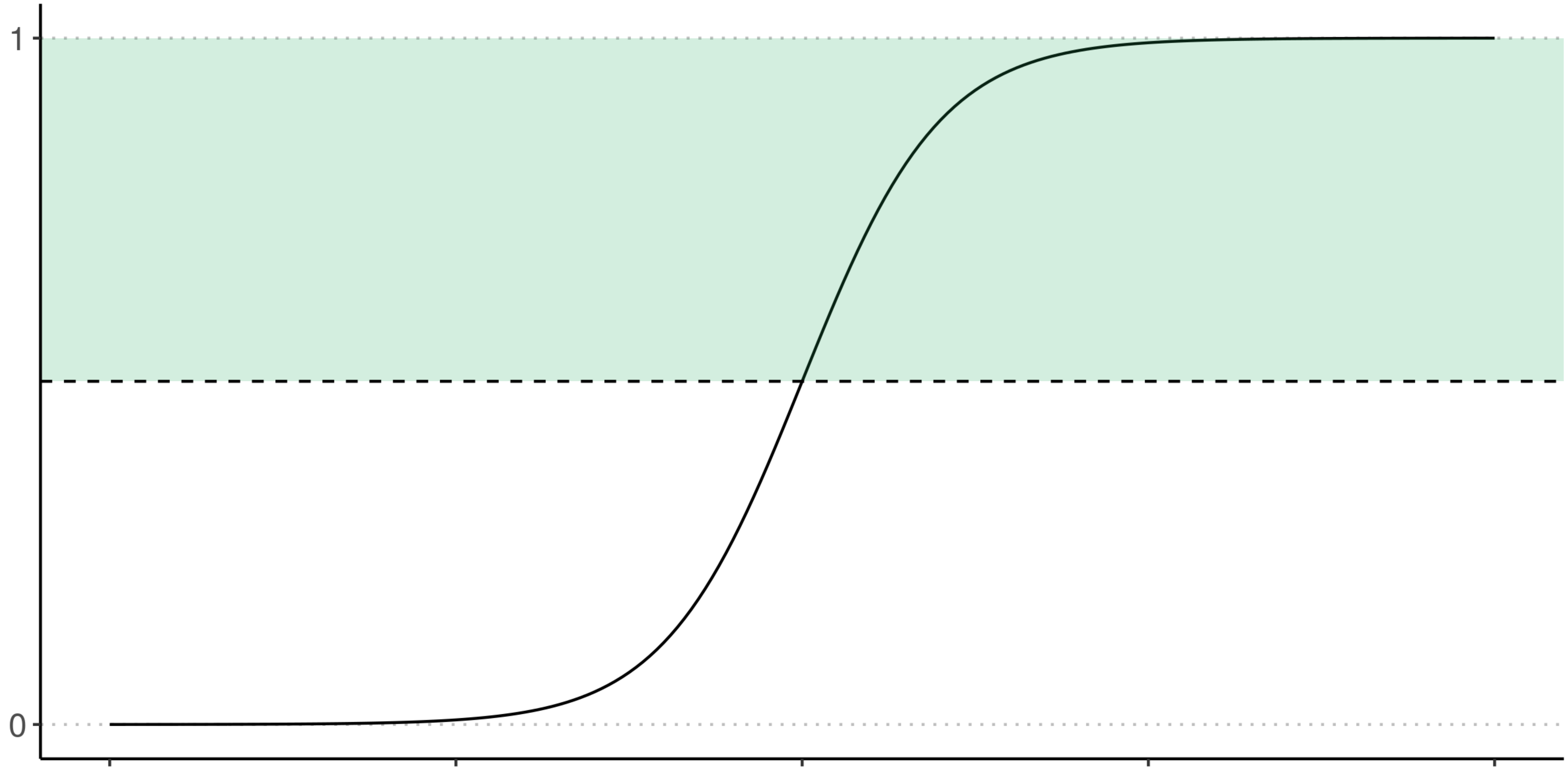
MACHINE LEARNING WITH PYSPARK

Andrew Collier
Data Scientist, Exegetic Analytics

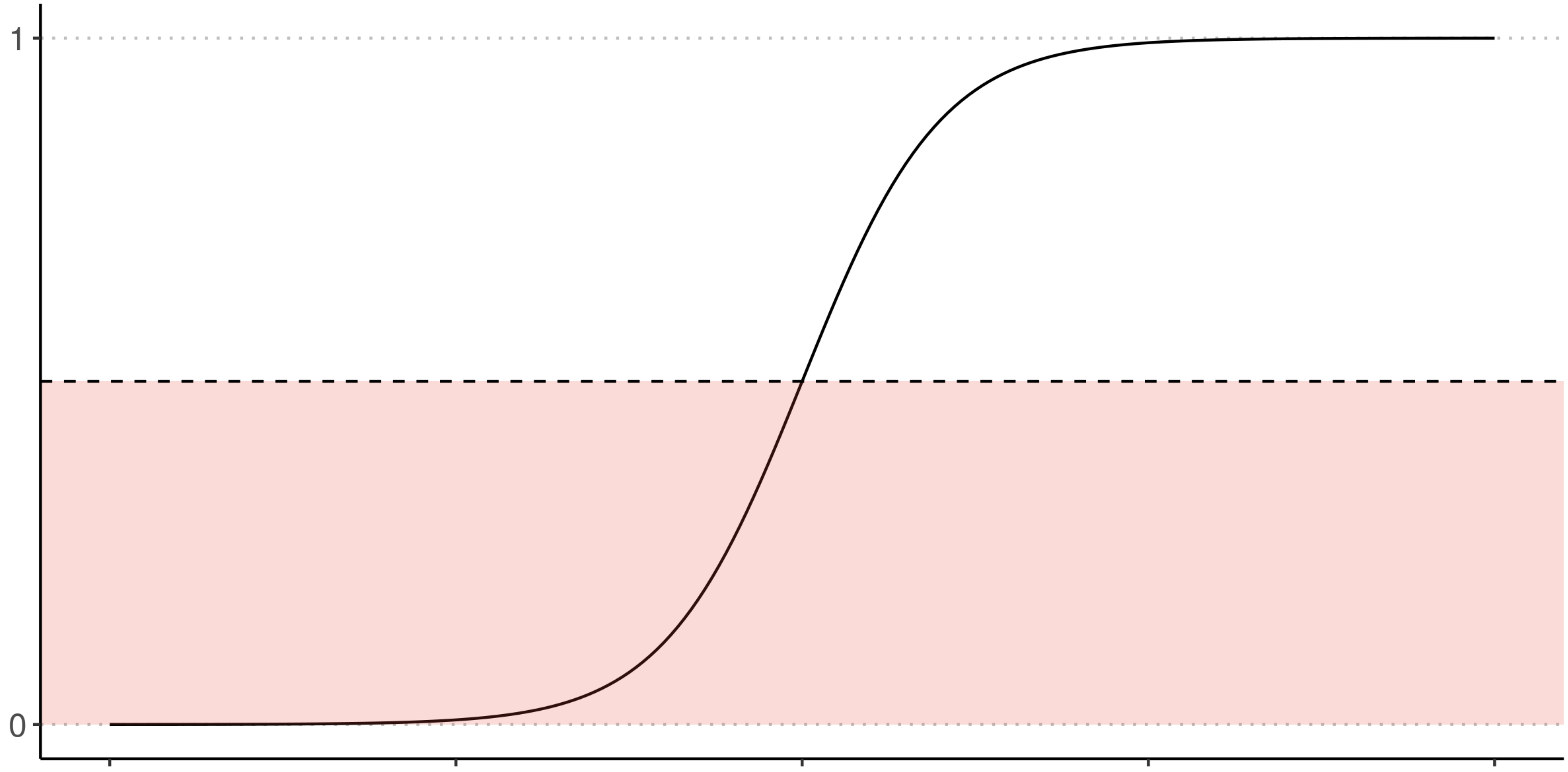
Logistic Curve



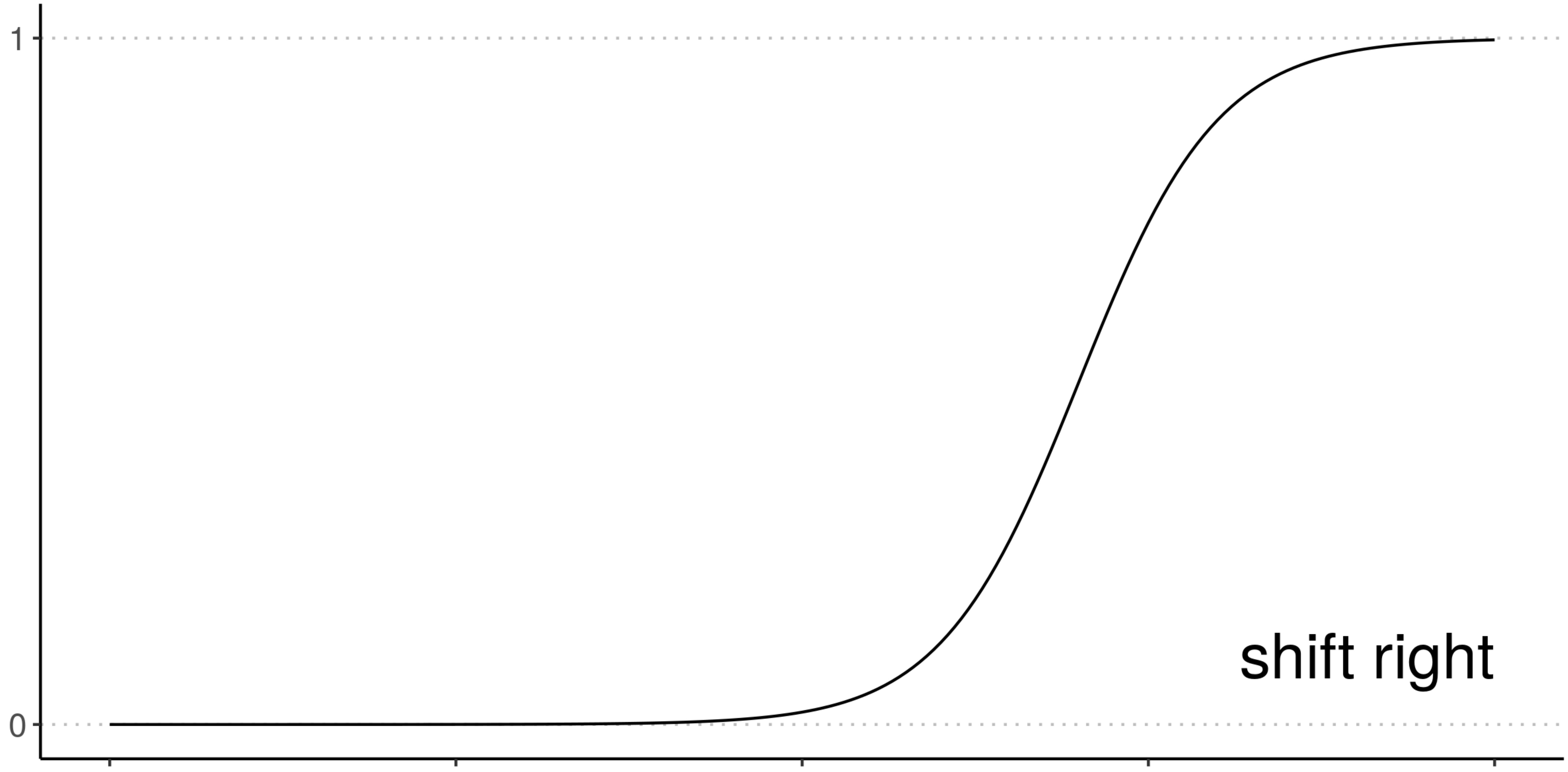
Logistic Curve



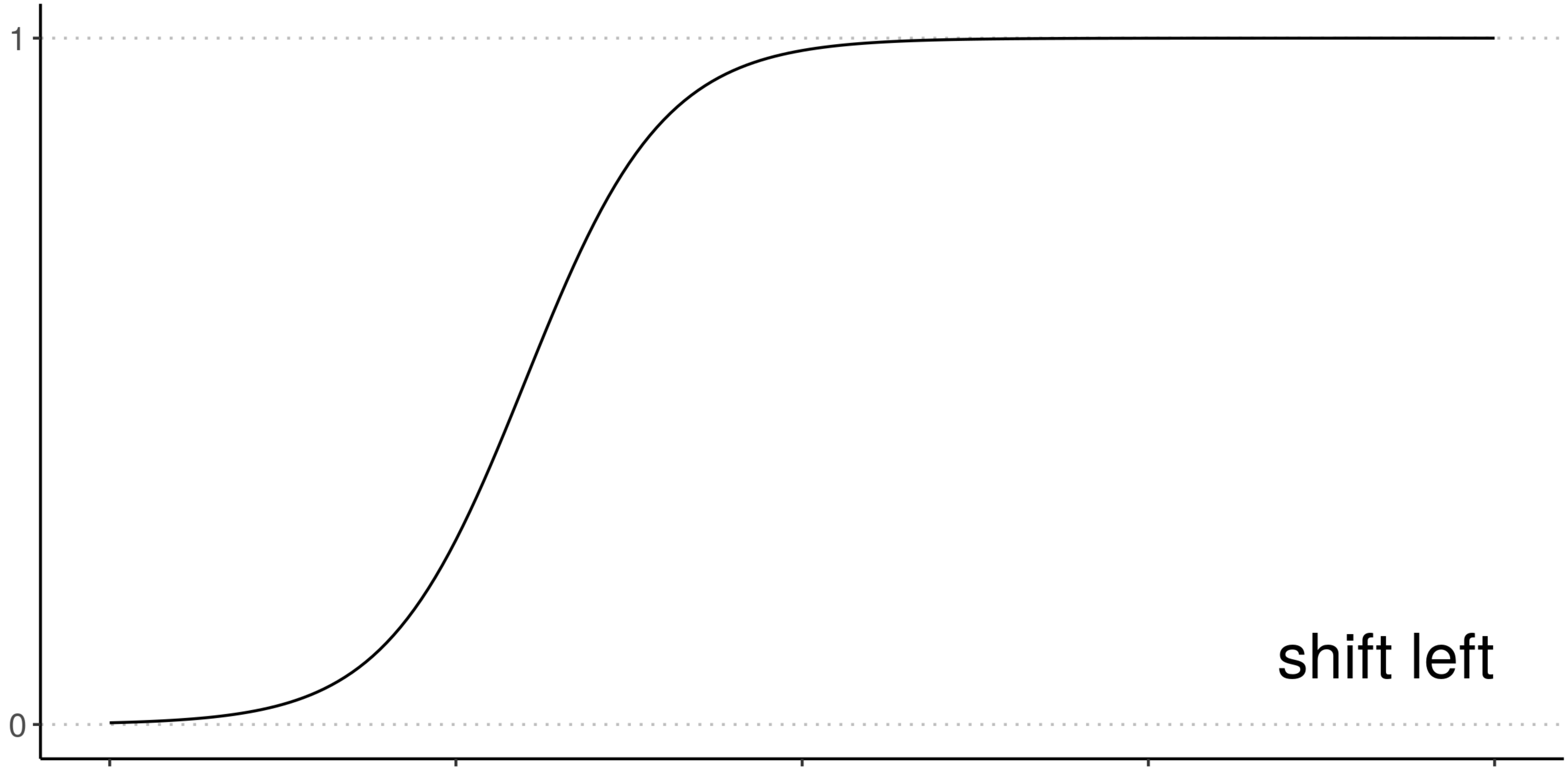
Logistic Curve



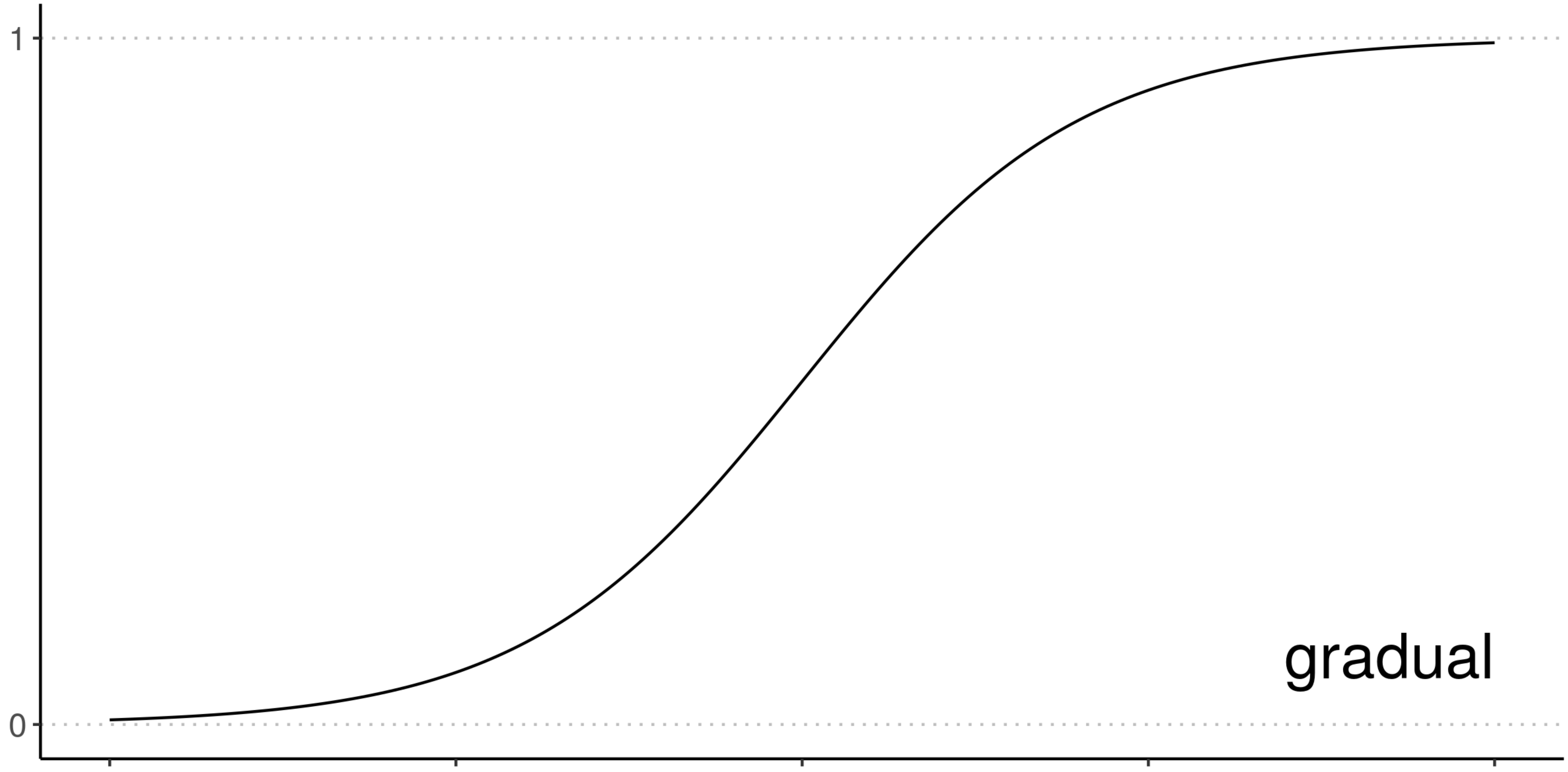
Logistic Curve



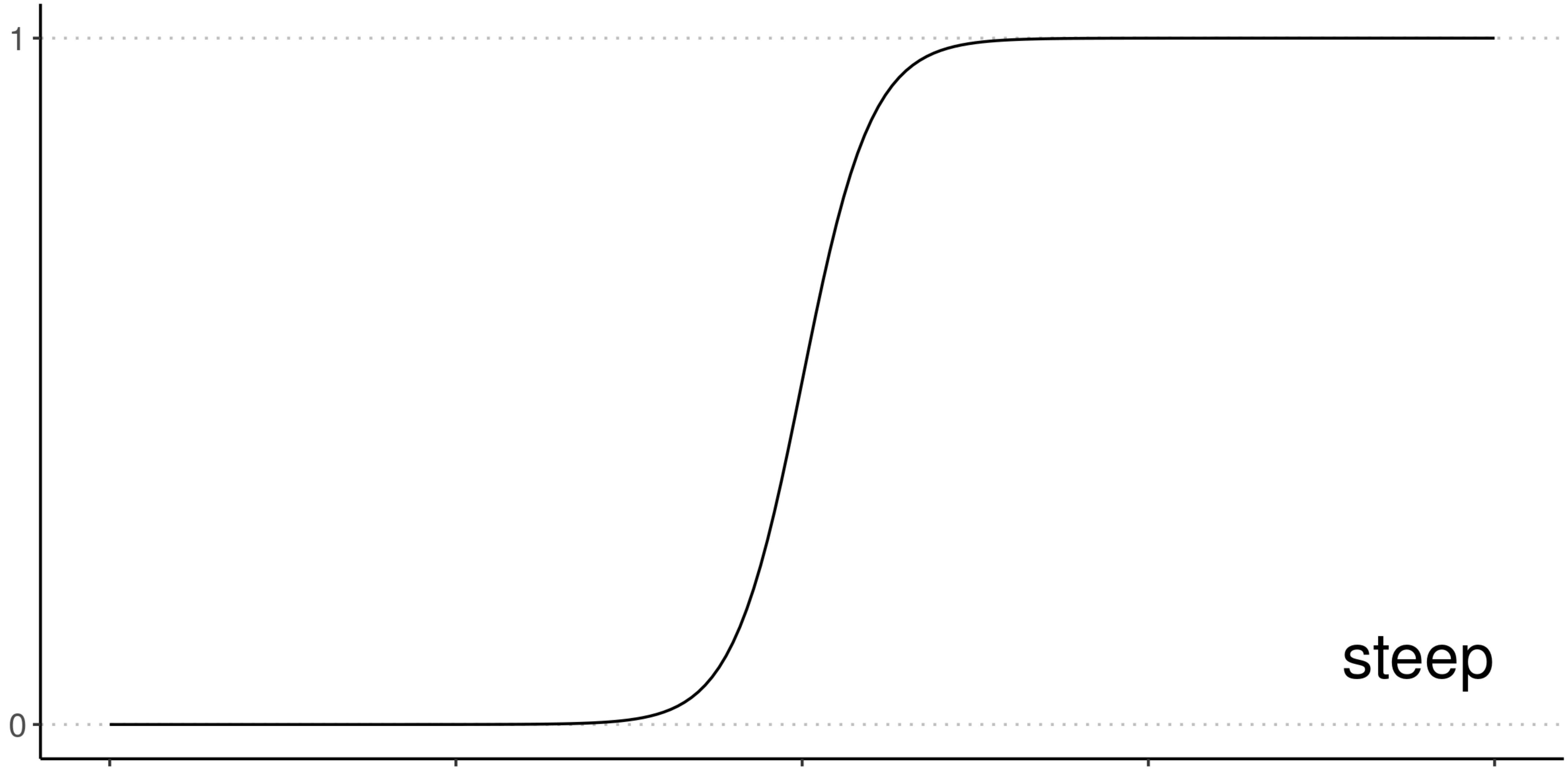
Logistic Curve



Logistic Curve



Logistic Curve



Cars revisited

Prepare for modeling:

- assemble the predictors into a single column (called `features`) and
- split data into training and testing sets.

```
+---+----+------+------+----+-----------+----------------------------------+-----+
|cyl|size|mass  |length|rpm |consumption|features                          |label|
+---+----+------+------+----+-----------+----------------------------------+-----+
|6  |3.0 |1451.0|4.775 |5200|9.05       |[6.0,3.0,1451.0,4.775,5200.0,9.05]|1.0  |
|4  |2.2 |1129.0|4.623 |5200|6.53       |[4.0,2.2,1129.0,4.623,5200.0,6.53]|0.0  |
|4  |2.2 |1399.0|4.547 |5600|7.84       |[4.0,2.2,1399.0,4.547,5600.0,7.84]|1.0  |
|4  |1.8 |1147.0|4.343 |6500|7.84       |[4.0,1.8,1147.0,4.343,6500.0,7.84]|0.0  |
|4  |1.6 |1111.0|4.216 |5750|9.05       |[4.0,1.6,1111.0,4.216,5750.0,9.05]|0.0  |
+---+----+------+------+----+-----------+----------------------------------+-----+
```

Build a Logistic Regression model

```
from pyspark.ml.classification import LogisticRegression
```

Create a Logistic Regression classifier.

```
logistic = LogisticRegression()
```

Learn from the training data.

```
logistic = logistic.fit(cars_train)
```

Predictions

```
prediction = logistic.transform(cars_test)
```

```
+-----+----------+---------------------------------------+
|label|prediction|probability                            |
+-----+----------+---------------------------------------+
|0.0  |0.0       |[0.8683802216422138,0.1316197783577862]|
|0.0  |1.0       |[0.1343792056399585,0.8656207943600416]|
|0.0  |0.0       |[0.9773546766387631,0.0226453233612368]|
|1.0  |1.0       |[0.0170508265586195,0.9829491734413806]|
|1.0  |0.0       |[0.6122241729292978,0.3877758270707023]|
+-----+----------+---------------------------------------+
```

Precision and recall

How well does model work on testing data?

Consult the confusion matrix.

```
+-----+----------+-----+
|label|prediction|count|
+-----+----------+-----+
|  1.0|       1.0|    8| - TP (true positive)
|  0.0|       1.0|    4| - FP (false positive)
|  1.0|       0.0|    2| - FN (false negative)
|  0.0|       0.0|   10| - TN (true negative)
+-----+----------+-----+
```

```
# Precision (positive)
TP / (TP + FP)
```

```
0.6666666666666666
```

```
# Recall (positive)
TP / (TP + FN)
```

```
0.8
```

Weighted metrics

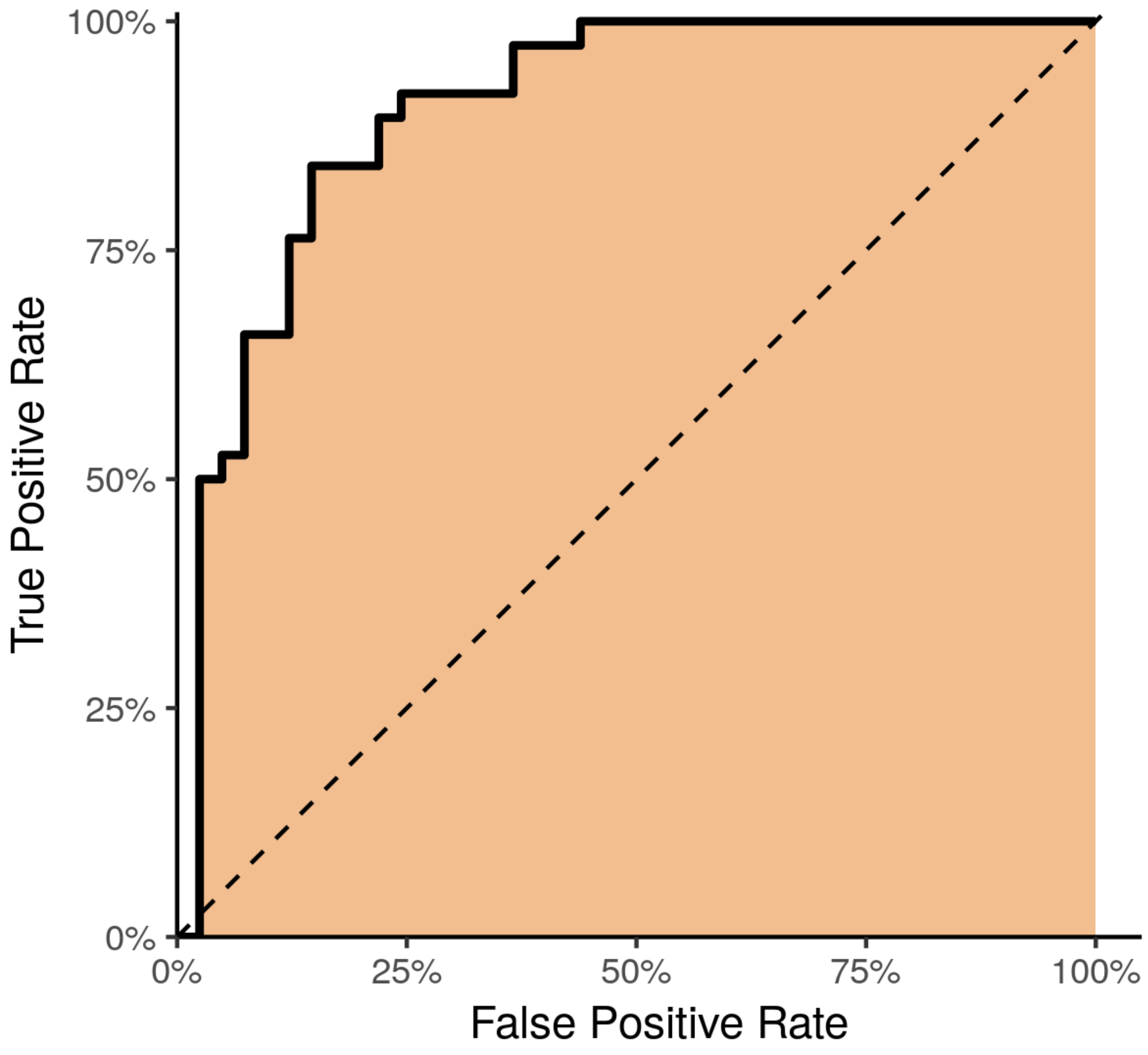
```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

evaluator = MulticlassClassificationEvaluator()
evaluator.evaluate(prediction, {evaluator.metricName: 'weightedPrecision'})
```

```
0.7638888888888888
```

Other metrics:

- `weightedRecall`
- `accuracy`
- `f1`

ROC and AUC



ROC = "Receiver Operating Characteristic"

- TP versus FP
- threshold = 0 (top right)
- threshold = 1 (bottom left)

AUC = "Area under the curve"

- ideally AUC = 1

Let's do Logistic Regression!

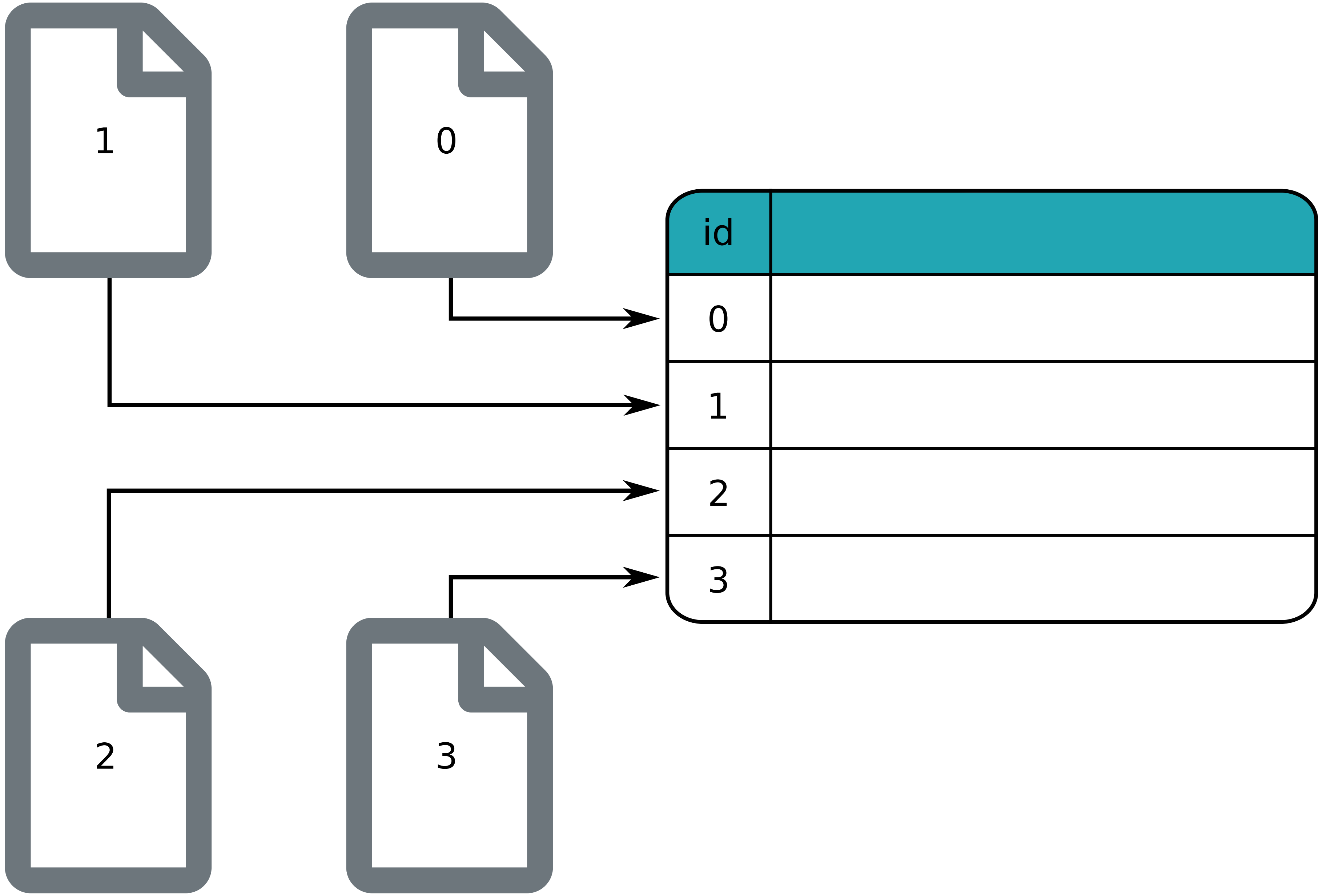
MACHINE LEARNING WITH PYSPARK

Turning Text into Tables

MACHINE LEARNING WITH PYSPARK

Andrew Collier
Data Scientist, Exegetic Analytics

One record per document



One document, many columns

Ten Little Fingers and Ten Little Toes

Tokenize

Ten | Little | Fingers | and | Ten | Little | Toes

Remove stop words

Ten | Little | Fingers | Ten | Little | Toes

Ten	Little	Fingers	Toes
2	2	1	1

A selection of children's books

```
books.show(truncate=False)
```

```
+---+------------------------------------+
|id |text                                |
+---+------------------------------------+
|0  |Forever, or a Long, Long Time       | ---> 'Long' is only present in this title
|1  |Winnie-the-Pooh                     |
|2  |Ten Little Fingers and Ten Little Toes|
|3  |Five Get into Trouble               | -+-> 'Five' is present in all of these titles
|4  |Five Have a Wonderful Time          |  |
|5  |Five Get into a Fix                 |  |
|6  |Five Have Plenty of Fun             | -+
+---+------------------------------------+
```

Removing punctuation

```
from pyspark.sql.functions import regexp_replace

# Regular expression (REGEX) to match commas and hyphens
REGEX = '[,\\-]'

books = books.withColumn('text', regexp_replace(books.text, REGEX, ' '))
```

```
Before                                    ->    After
+---+------------------------------+            +---+------------------------------+
|id |text                          |            |id |text                          |
+---+------------------------------+            +---+------------------------------+
|0  |Forever, or a Long, Long Time|             |0  |Forever  or a Long  Long Time|
|1  |Winnie-the-Pooh               |            |1  |Winnie the Pooh               |
+---+------------------------------+            +---+------------------------------+
```

Text to tokens

```
from pyspark.ml.feature import Tokenizer

books = Tokenizer(inputCol="text", outputCol="tokens").transform(books)
```

```
+---------------------------------------+---------------------------------------------+
|text                                   |tokens                                       |
+---------------------------------------+---------------------------------------------+
|Forever or a Long Long Time            |[forever, or, a, long, long, time]           |
|Winnie the Pooh                        |[winnie, the, pooh]                          |
|Ten Little Fingers and Ten Little Toes|[ten, little, fingers, and, ten, little, toes]|
|Five Get into Trouble                  |[five, get, into, trouble]                   |
|Five Have a Wonderful Time             |[five, have, a, wonderful, time]             |
+---------------------------------------+---------------------------------------------+
```

What are stop words?

```
from pyspark.ml.feature import StopWordsRemover

stopwords = StopWordsRemover()

# Take a look at the list of stop words
stopwords.getStopWords()
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', 'your', 'yours',
'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', 'her', 'hers', 'herself',
'it', 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which',
'who', 'whom', 'this', 'that', 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be',
'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', ...]
```

Removing stop words

```
# Specify the input and output column names
stopwords = stopwords.setInputCol('tokens').setOutputCol('words')

books = stopwords.transform(books)
```

```
+----------------------------------------------+------------------------------------------+
|tokens                                        |words                                     |
+----------------------------------------------+------------------------------------------+
|[forever, or, a, long, long, time]            |[forever, long, long, time]               |
|[winnie, the, pooh]                           |[winnie, pooh]                            |
|[ten, little, fingers, and, ten, little, toes]|[ten, little, fingers, ten, little, toes]|
|[five, get, into, trouble]                    |[five, get, trouble]                      |
|[five, have, a, wonderful, time]              |[five, wonderful, time]                   |
+----------------------------------------------+------------------------------------------+
```

Feature hashing

```
from pyspark.ml.feature import HashingTF

hasher = HashingTF(inputCol="words", outputCol="hash", numFeatures=32)
books = hasher.transform(books)
```

```
+---+--------------------------------------+------------------------------------+
|id |words                                 |hash                                |
+---+--------------------------------------+------------------------------------+
|0  |[forever, long, long, time]           |(32,[8,13,14],[2.0,1.0,1.0])        |
|1  |[winnie, pooh]                        |(32,[1,31],[1.0,1.0])               |
|2  |[ten, little, fingers, ten, little, toes]|(32,[1,15,25,30],[2.0,2.0,1.0,1.0])|
|3  |[five, get, trouble]                  |(32,[6,7,23],[1.0,1.0,1.0])         |
|4  |[five, wonderful, time]               |(32,[6,13,25],[1.0,1.0,1.0])        |
+---+--------------------------------------+------------------------------------+
```

Dealing with common words

```
from pyspark.ml.feature import IDF

books = IDF(inputCol="hash", outputCol="features").fit(books).transform(books)
```

```
+--------------------------------------+--------------------------------------+
|words                                 |features                              |
+--------------------------------------+--------------------------------------+
|[forever, long, long, time]           |(32,[8,13,14],[2.598,1.299,1.704])    |
|[winnie, pooh]                        |(32,[1,31],[1.299,1.704])             |
|[ten, little, fingers, ten, little, toes]|(32,[1,15,25,30],[2.598,3.409,1.011,1.704])|
|[five, get, trouble]                  |(32,[6,7,23],[0.788,1.704,1.299])     |
|[five, wonderful, time]               |(32,[6,13,25],[0.788,1.299,1.011])    |
+--------------------------------------+--------------------------------------+
```

Text ready for Machine Learning!

MACHINE LEARNING WITH PYSPARK