One-Hot Encoding

MACHINE LEARNING WITH PYSPARK

Andrew Collier
Data Scientist, Fathom Data

The problem with indexed values

```
# Counts for 'type' category

+-------+-----+
|   type|count|
+-------+-----+
|Midsize|   22|
|  Small|   21|
|Compact|   16|
| Sporty|   14|
|  Large|   11|
|    Van|    9|
+-------+-----+
```

```
# Numerical indices for 'type' category

+-------+--------+
|   type|type_idx|
+-------+--------+
|Midsize|     0.0|
|  Small|     1.0|
|Compact|     2.0|
| Sporty|     3.0|
|  Large|     4.0|
|    Van|     5.0|
+-------+--------+
```

Dummy variables

```
+-------+      +-------+-------+-------+-------+-------+-------+
|   type|      |Midsize|  Small|Compact| Sporty|  Large|    Van|
+-------+      +-------+-------+-------+-------+-------+-------+
|Midsize|      |   X   |       |       |       |       |       |
|  Small|      |       |   X   |       |       |       |       |
|Compact| ===> |       |       |   X   |       |       |       |
| Sporty|      |       |       |       |   X   |       |       |
|  Large|      |       |       |       |       |   X   |       |
|    Van|      |       |       |       |       |       |   X   |
+-------+      +-------+-------+-------+-------+-------+-------+
```

Each categorical level becomes a column.

Dummy variables: binary encoding

```
+-------+      +-------+-------+-------+-------+-------+-------+
|   type|      |Midsize|  Small|Compact| Sporty|  Large|    Van|
+-------+      +-------+-------+-------+-------+-------+-------+
|Midsize|      |   1   |   0   |   0   |   0   |   0   |   0   |
|  Small|      |   0   |   1   |   0   |   0   |   0   |   0   |
|Compact| ===> |   0   |   0   |   1   |   0   |   0   |   0   |
| Sporty|      |   0   |   0   |   0   |   1   |   0   |   0   |
|  Large|      |   0   |   0   |   0   |   0   |   1   |   0   |
|    Van|      |   0   |   0   |   0   |   0   |   0   |   1   |
+-------+      +-------+-------+-------+-------+-------+-------+
```

Binary values indicate the presence (`1`) or absence (`0`) of the corresponding level.

Dummy variables: sparse representation

```
+-------+      +-------+------+-------+-------+------+-------+      +------+-----+
|   type|      |Midsize|  Small|Compact| Sporty|  Large|    Van|      |Column|Value|
+-------+      +-------+------+-------+-------+------+-------+      +------+-----+
|Midsize|      |   1   |   0   |   0   |   0   |   0   |   0   |      |     0|    1|
|  Small|      |   0   |   1   |   0   |   0   |   0   |   0   |      |     1|    1|
|Compact| ===> |   0   |   0   |   1   |   0   |   0   |   0   | ===> |     2|    1|
| Sporty|      |   0   |   0   |   0   |   1   |   0   |   0   |      |     3|    1|
|  Large|      |   0   |   0   |   0   |   0   |   1   |   0   |      |     4|    1|
|    Van|      |   0   |   0   |   0   |   0   |   0   |   1   |      |     5|    1|
+-------+      +-------+------+-------+-------+------+-------+      +------+-----+
```

Sparse representation: store column index and value.

Dummy variables: redundant column

```
+--------+      +-------+-------+-------+-------+-------+      +------+-----+
|    type|      |Midsize|  Small|Compact| Sporty|  Large|      |Column|Value|
+--------+      +-------+-------+-------+-------+-------+      +------+-----+
|Midsize|       |   1   |   0   |   0   |   0   |   0   |      |     0|    1|
|   Small|      |   0   |   1   |   0   |   0   |   0   |      |     1|    1|
|Compact| ===> |   0   |   0   |   1   |   0   |   0   | ===> |     2|    1|
|  Sporty|      |   0   |   0   |   0   |   1   |   0   |      |     3|    1|
|   Large|      |   0   |   0   |   0   |   0   |   1   |      |     4|    1|
|     Van|      |   0   |   0   |   0   |   0   |   0   |      |      |     |
+--------+      +-------+-------+-------+-------+-------+      +------+-----+
```

Levels are mutually exclusive, so drop one.

One-hot encoding

```
from pyspark.ml.feature import OneHotEncoder

onehot = OneHotEncoder(inputCols=['type_idx'], outputCols=['type_dummy'])
```

Fit the encoder to the data.

```
onehot = onehot.fit(cars)
```

```
# How many category levels?
onehot.categorySizes
```

```
[6]
```

One-hot encoding

```
cars = onehot.transform(cars)
cars.select('type', 'type_idx', 'type_dummy').distinct().sort('type_idx').show()
```

```
+-------+--------+-------------+
|   type|type_idx|   type_dummy|
+-------+--------+-------------+
|Midsize|     0.0|(5,[0],[1.0])|
|  Small|     1.0|(5,[1],[1.0])|
|Compact|     2.0|(5,[2],[1.0])|
| Sporty|     3.0|(5,[3],[1.0])|
|  Large|     4.0|(5,[4],[1.0])|
|    Van|     5.0|    (5,[],[])|
+-------+--------+-------------+
```

Dense versus sparse

```
from pyspark.mllib.linalg import DenseVector, SparseVector
```

Store this vector: [1, 0, 0, 0, 0, 7, 0, 0].

```
DenseVector([1, 0, 0, 0, 0, 7, 0, 0])
```

```
DenseVector([1.0, 0.0, 0.0, 0.0, 0.0, 7.0, 0.0, 0.0])
```

```
SparseVector(8, [0, 5], [1, 7])
```

```
SparseVector(8, {0: 1.0, 5: 7.0})
```

One-Hot Encode categoricals

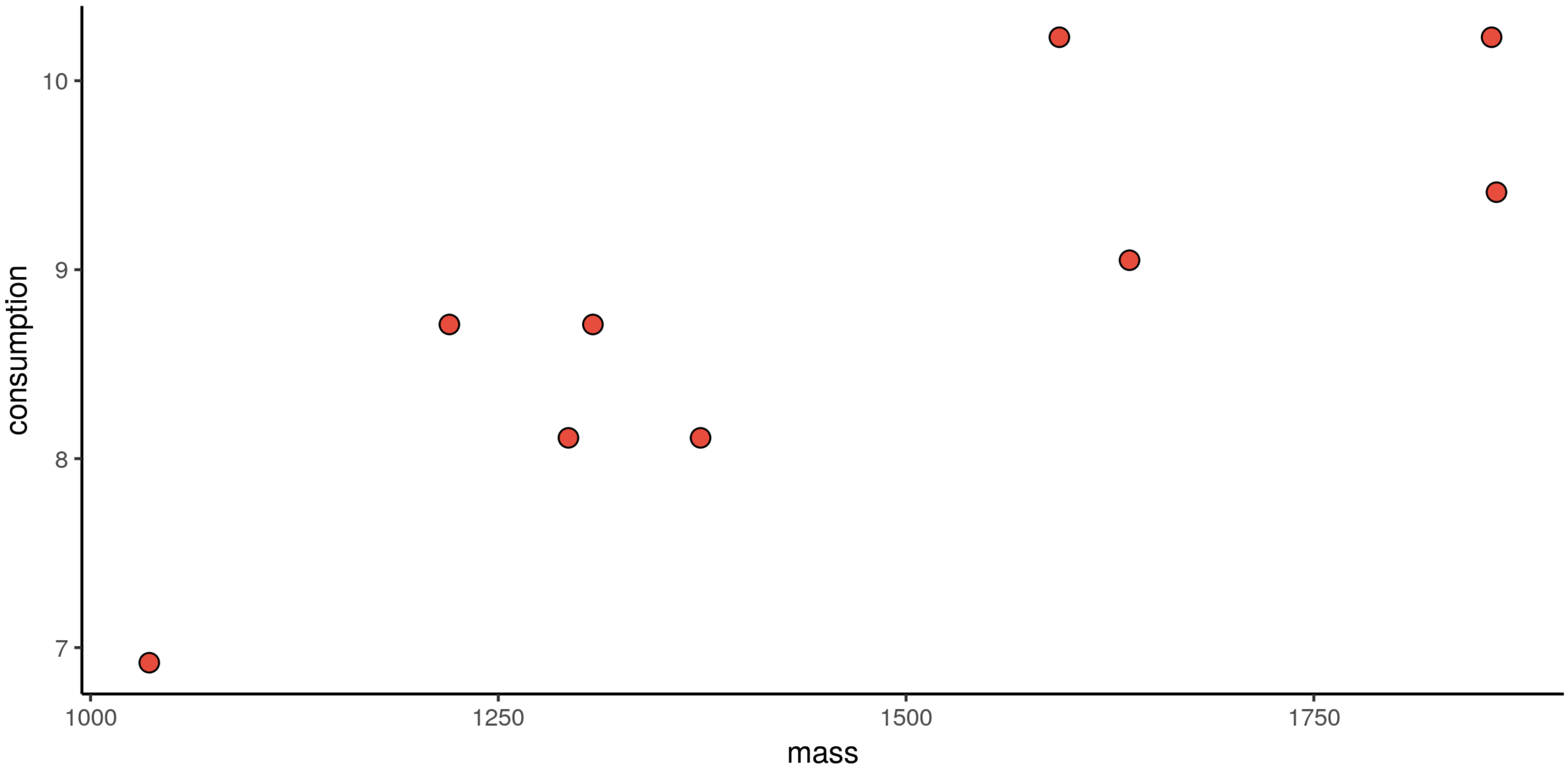
MACHINE LEARNING WITH PYSPARK

Regression

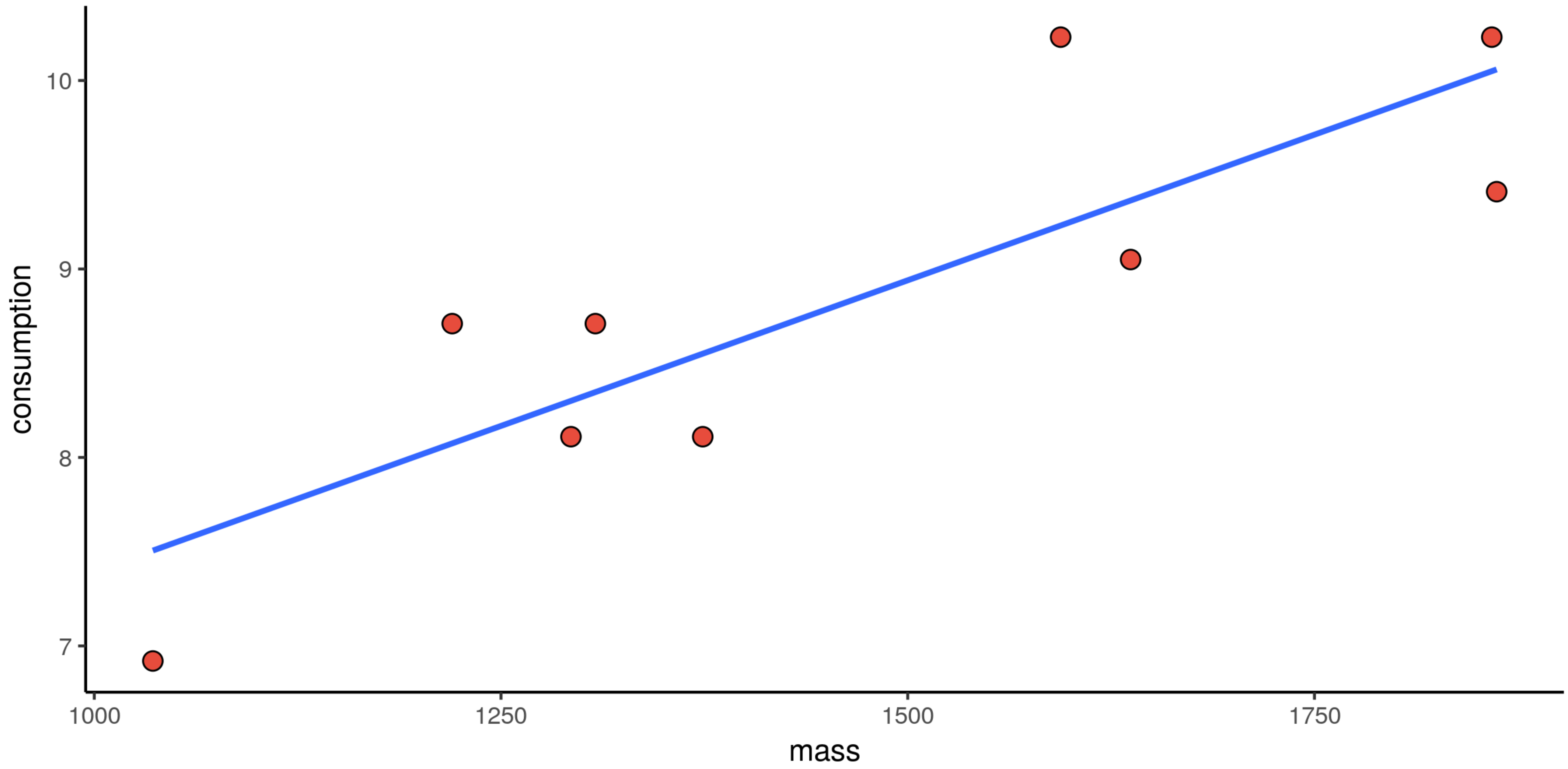
MACHINE LEARNING WITH PYSPARK

Andrew Collier
Data Scientist, Fathom Data

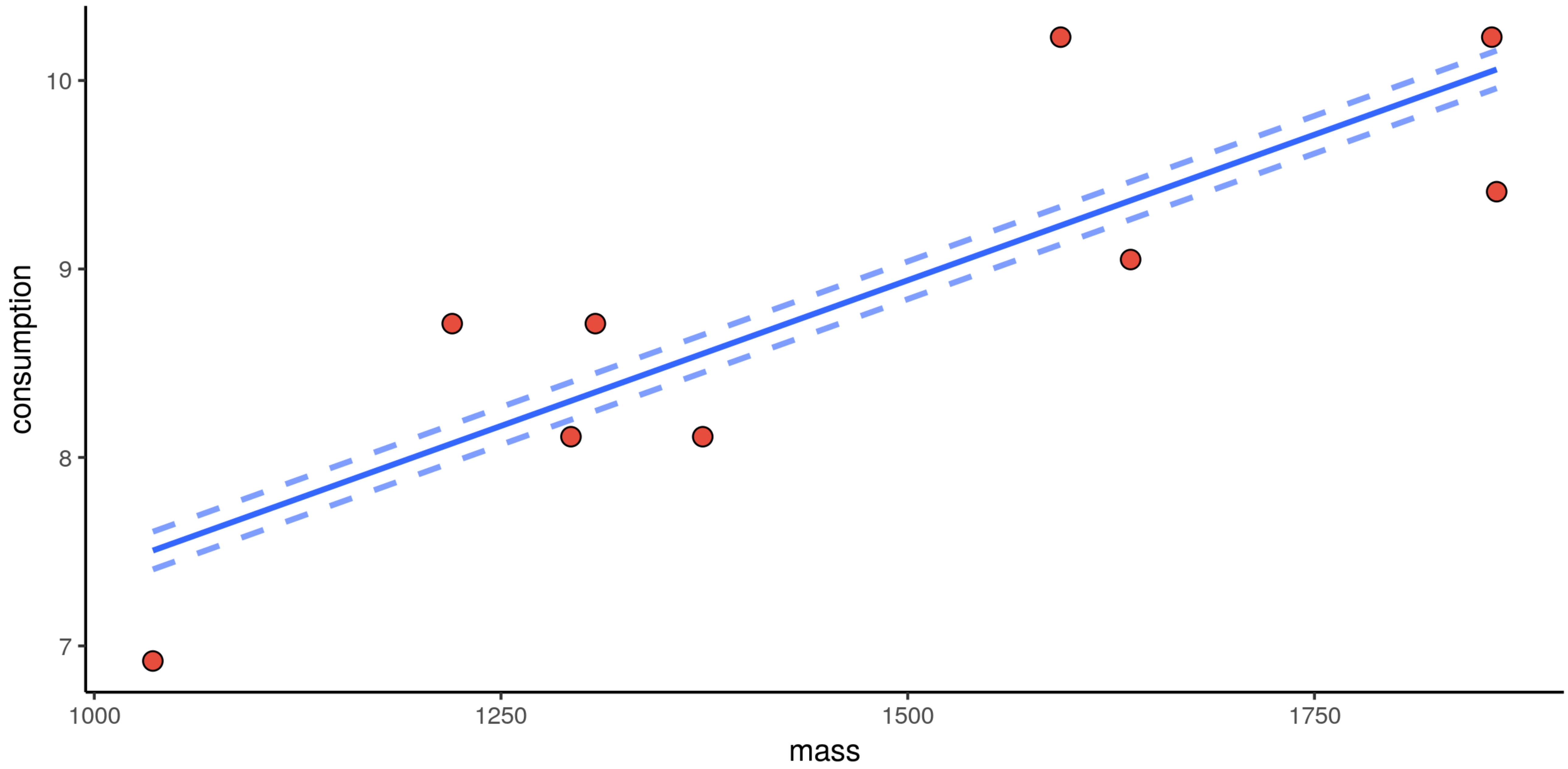
Consumption versus mass: scatter



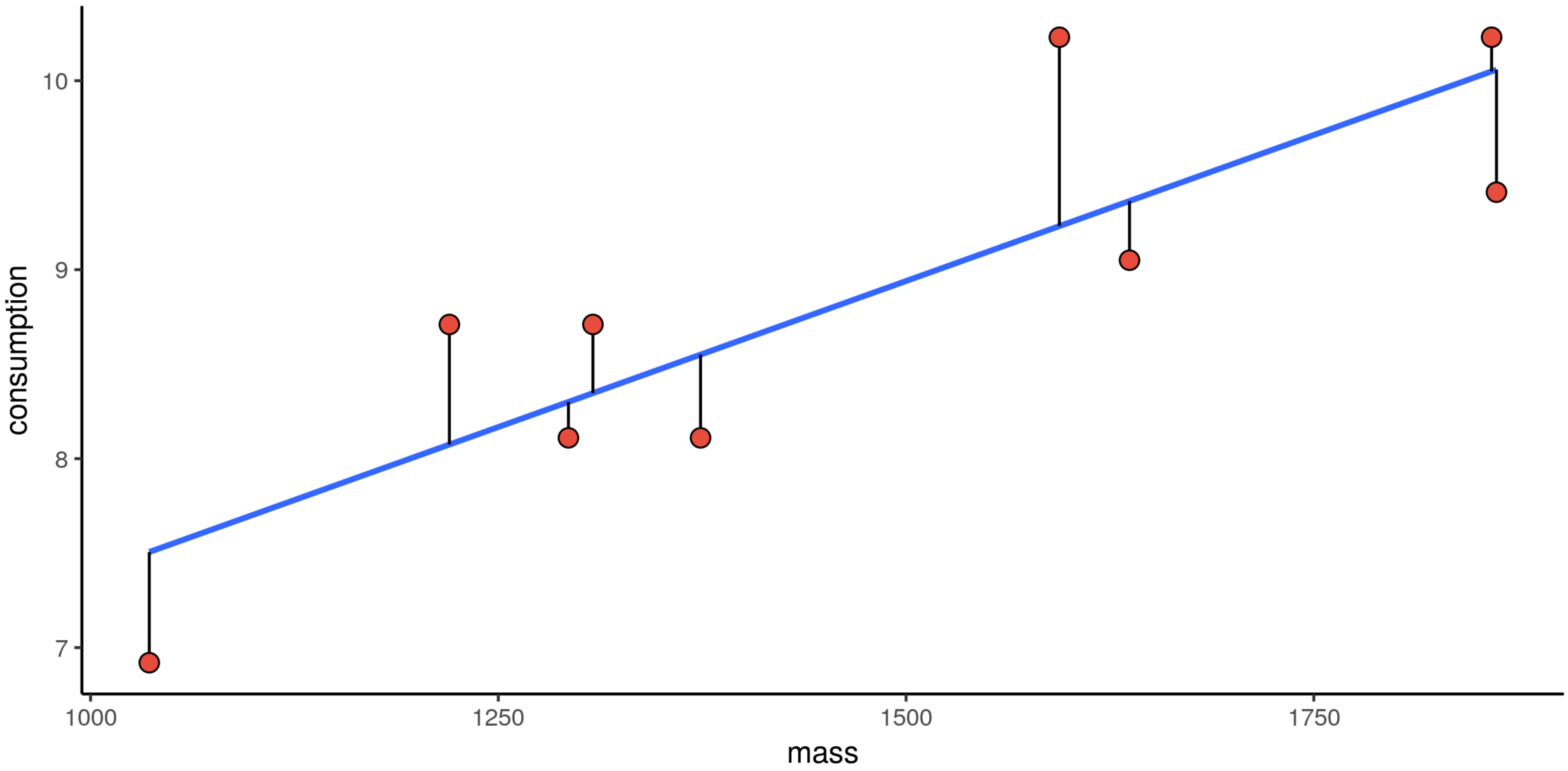
Consumption versus mass: fit



Consumption versus mass: alternative fits



Consumption versus mass: residuals



Loss function

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y_i})^2$$

MSE = "Mean Squared Error"

Loss function: Observed values

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

y_i — observed values

Loss function: Model values

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

y_i — observed values

$\hat{y}_i$ — model values

Loss function: Mean

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

y_i — observed values

$\hat{y}_i$ — model values

Assemble predictors

Predict `consumption` using `mass`, `cyl` and `type_dummy`.

Consolidate predictors into a single column.

```
+------+---+-------------+-----------------------------+-----------+
|mass  |cyl|type_dummy   |features                     |consumption|
+------+---+-------------+-----------------------------+-----------+
|1451.0|6  |(5,[0],[1.0])|(7,[0,1,2],[1451.0,6.0,1.0])|9.05       |
|1129.0|4  |(5,[2],[1.0])|(7,[0,1,4],[1129.0,4.0,1.0])|6.53       |
|1399.0|4  |(5,[2],[1.0])|(7,[0,1,4],[1399.0,4.0,1.0])|7.84       |
|1147.0|4  |(5,[1],[1.0])|(7,[0,1,3],[1147.0,4.0,1.0])|7.84       |
|1111.0|4  |(5,[3],[1.0])|(7,[0,1,5],[1111.0,4.0,1.0])|9.05       |
+------+---+-------------+-----------------------------+-----------+
```

Build regression model

```
from pyspark.ml.regression import LinearRegression

regression = LinearRegression(labelCol='consumption')
```

Fit to `cars_train` (training data).

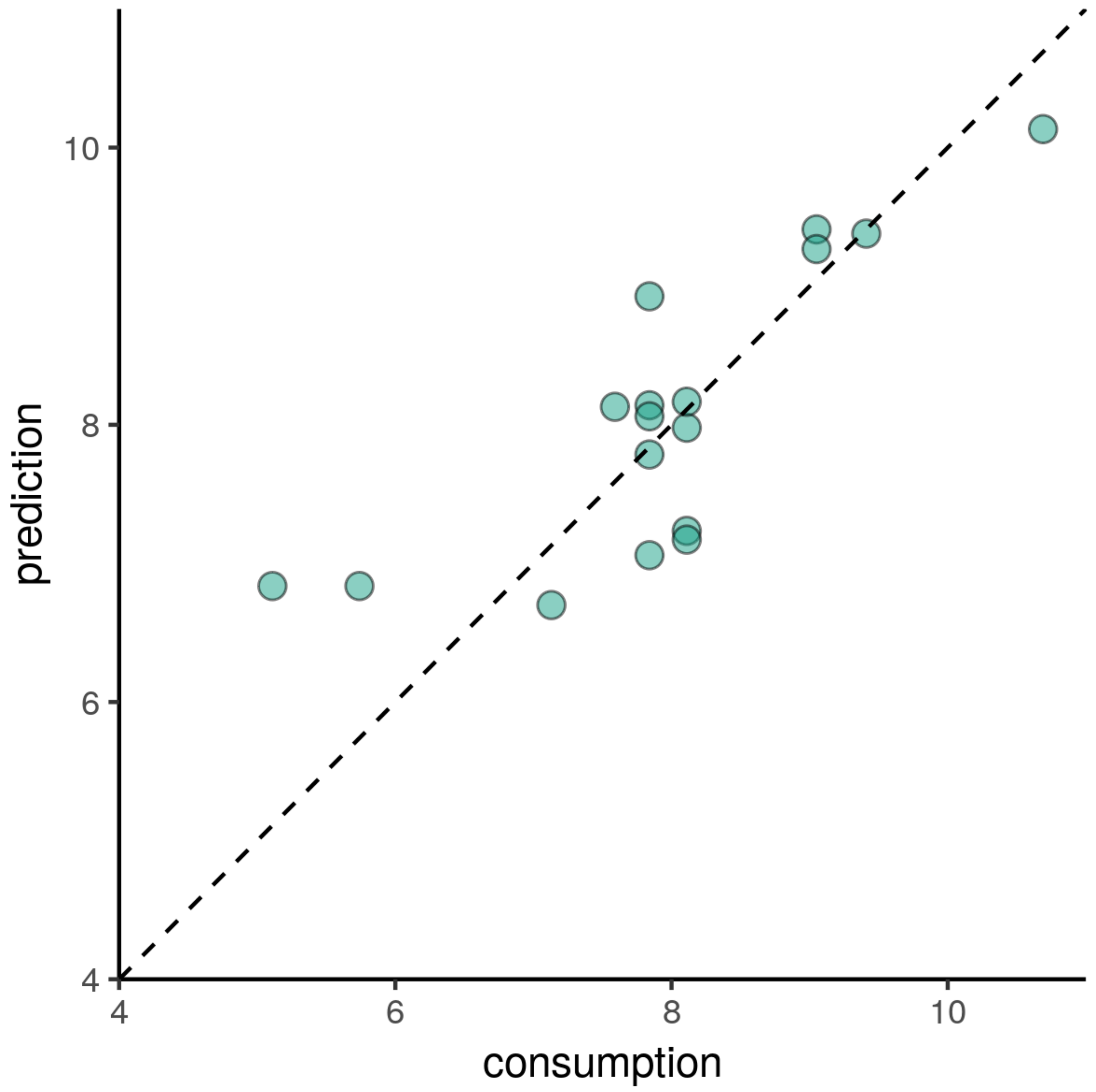
```
regression = regression.fit(cars_train)
```

Predict on `cars_test` (testing data).

```
predictions = regression.transform(cars_test)
```

Examine predictions

```
+-----------+-----------------+
|consumption|prediction       |
+-----------+-----------------+
|7.84       |8.92699470743403 |
|9.41       |9.379295891451353|
|8.11       |7.23487264538364 |
|9.05       |9.409860194333735|
|7.84       |7.059190923328711|
|7.84       |7.785909738591766|
|7.59       |8.129959405168547|
|5.11       |6.836843743852942|
|8.11       |7.1717370265201 5|
+-----------+-----------------+
```



Calculate RMSE

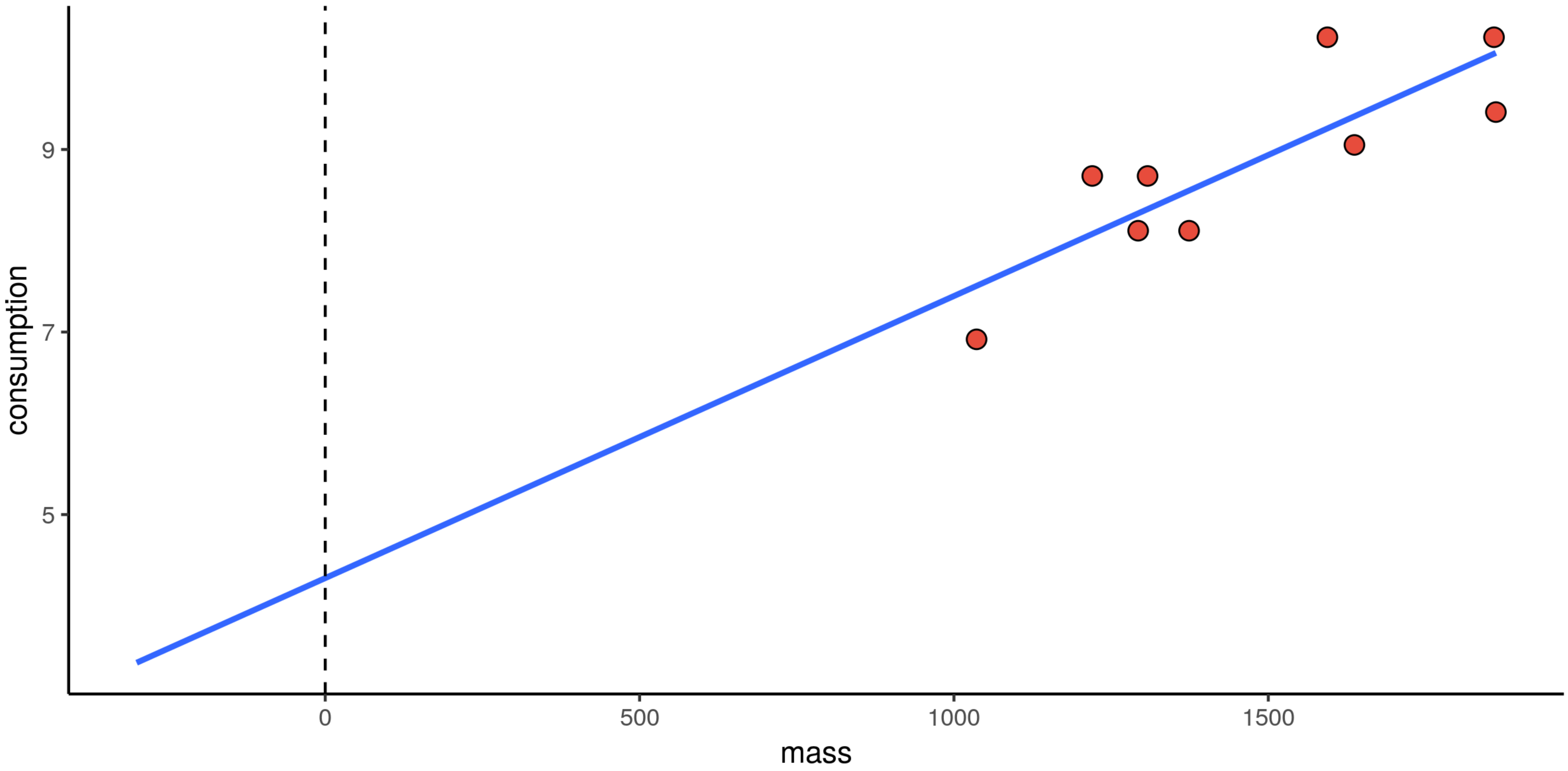
```
from pyspark.ml.evaluation import RegressionEvaluator


# Find RMSE (Root Mean Squared Error)
RegressionEvaluator(labelCol='consumption').evaluate(predictions)
```

```
0.708699086182001
```

A `RegressionEvaluator` can also calculate the following metrics:

- `mae` (Mean Absolute Error)
- `r2` (R^2)
- `mse` (Mean Squared Error).

Consumption versus mass: intercept



Examine intercept

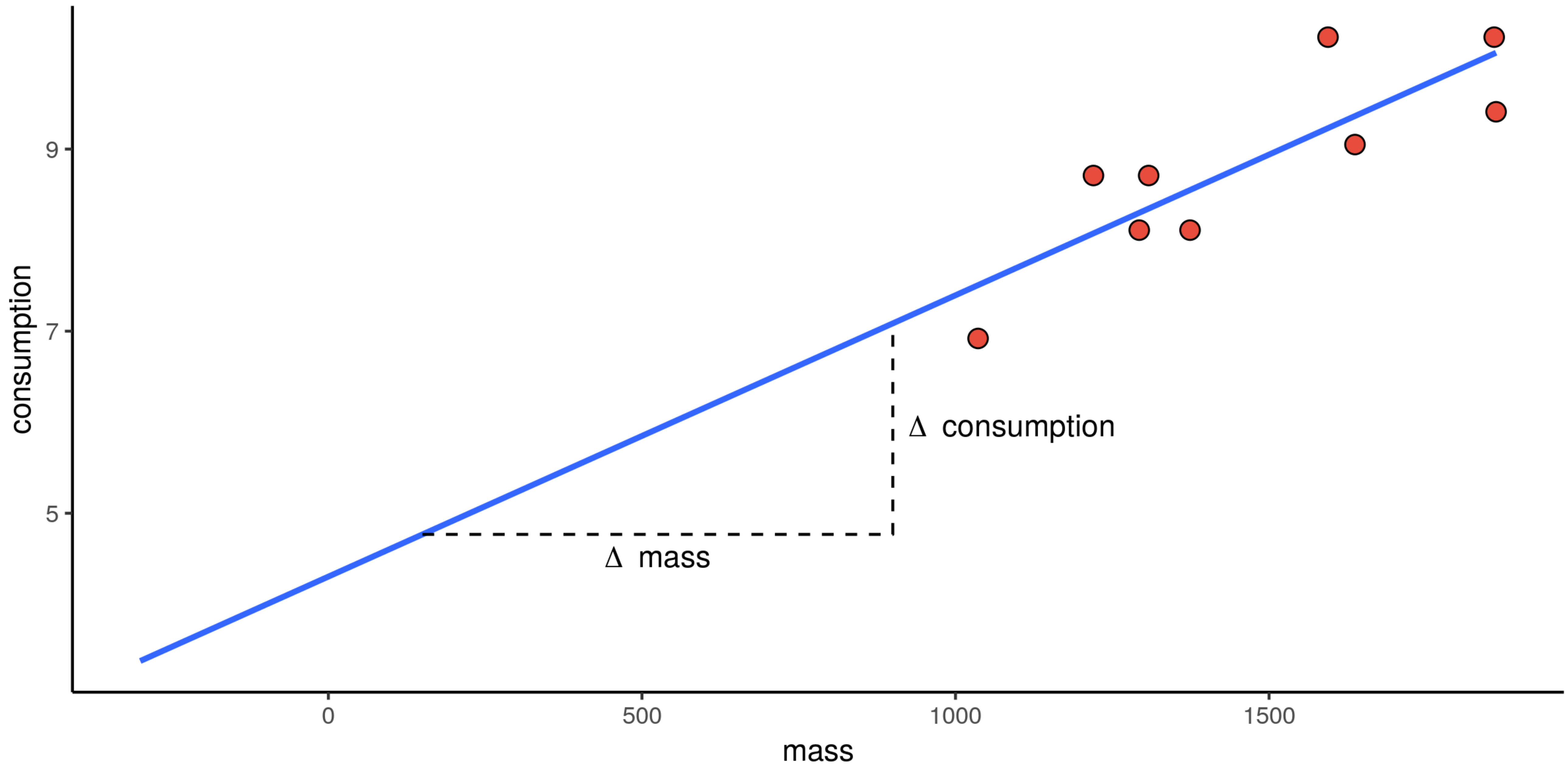
```
regression.intercept
```

```
4.9450616833727095
```

This is the fuel consumption in the (hypothetical) case that:

- `mass` = 0
- `cyl` = 0 and
- vehicle type is 'Van'.

Consumption versus mass: slope



Examine Coefficients

```
regression.coefficients
```

```
DenseVector([0.0027, 0.1897, -1.309, -1.7933, -1.3594, -1.2917, -1.9693])
```

```
mass         0.0027
cyl          0.1897

Midsize     -1.3090
Small       -1.7933
Compact     -1.3594
Sporty      -1.2917
Large       -1.9693
```

Regression for numeric predictions

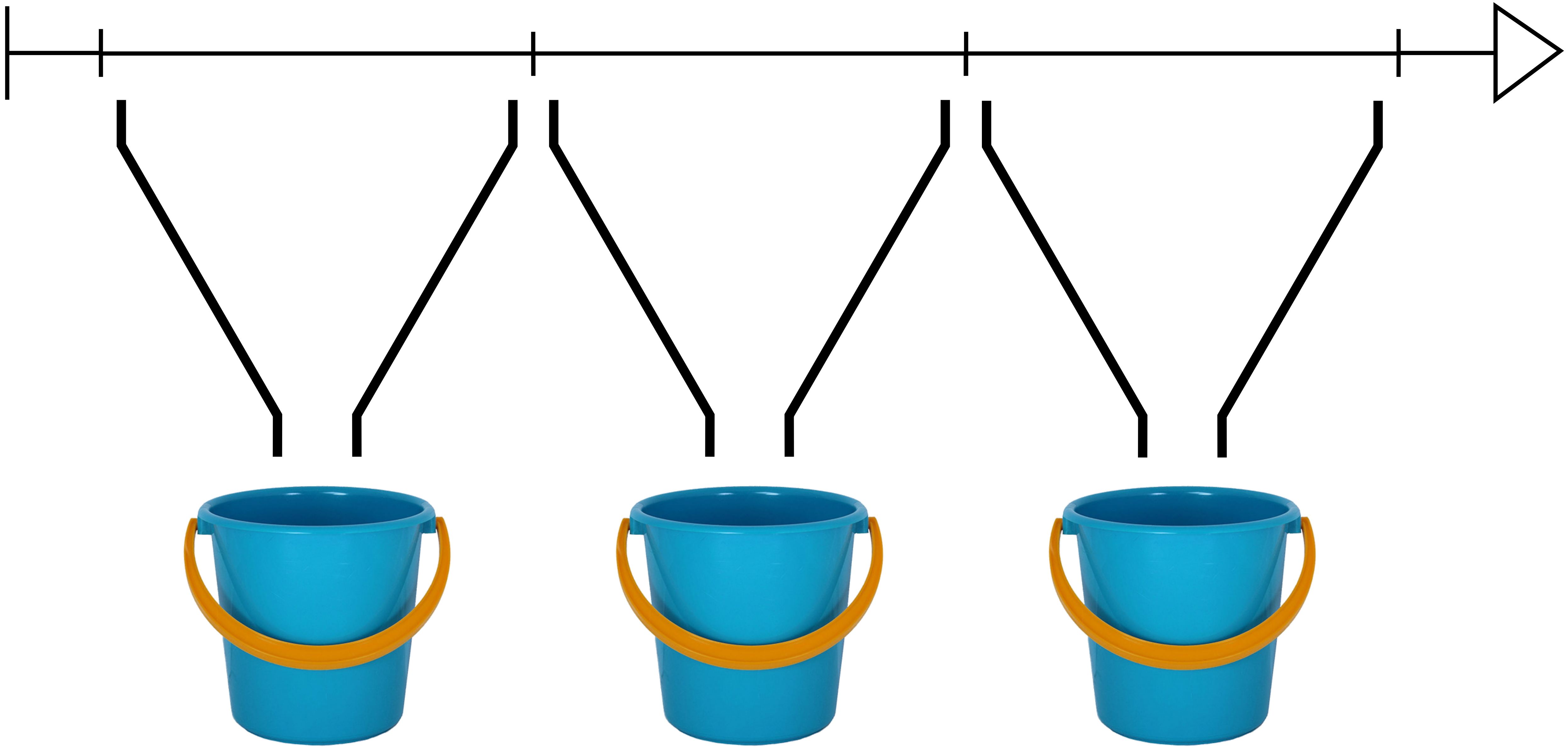
MACHINE LEARNING WITH PYSPARK

Bucketing & Engineering

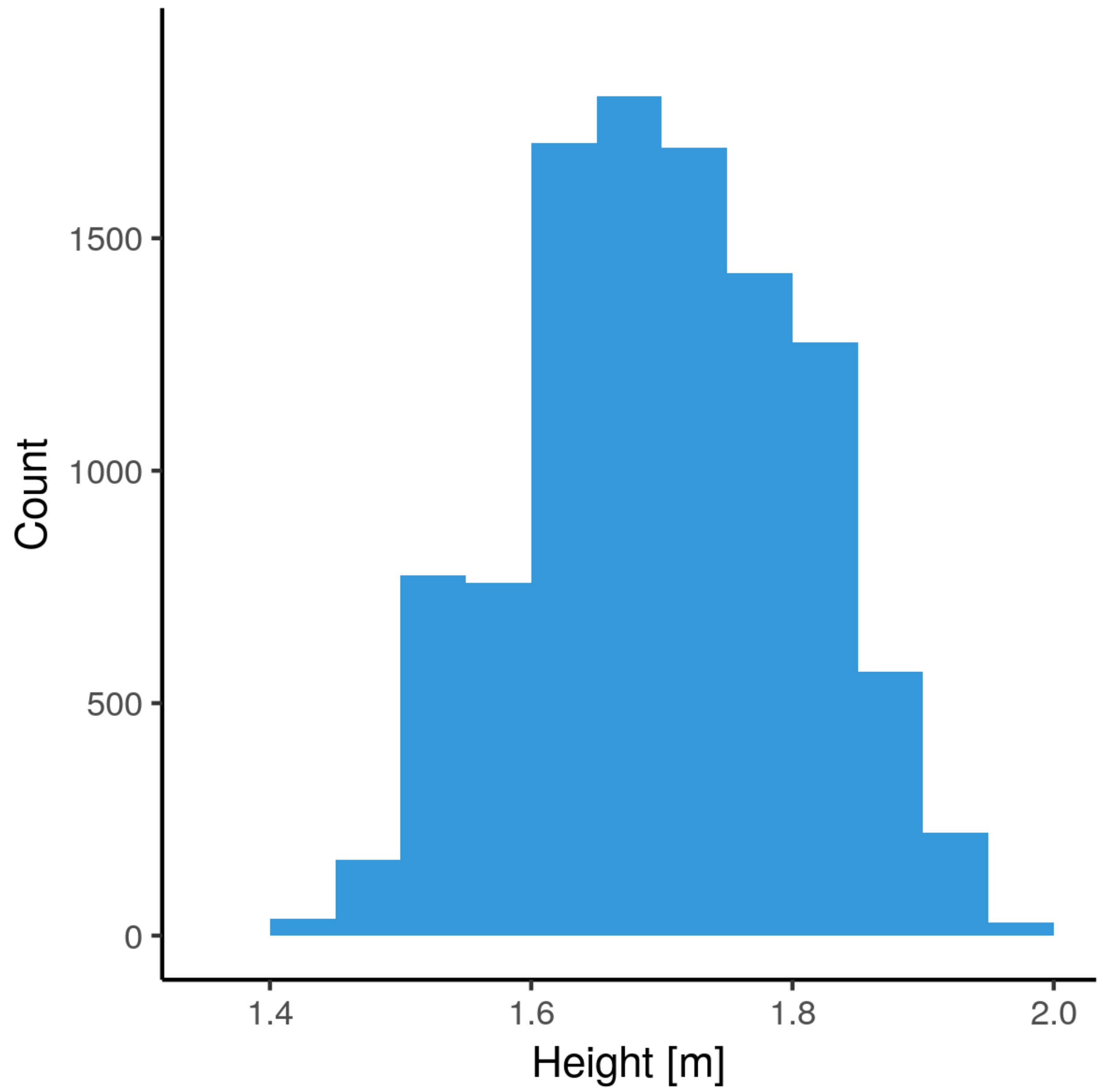
MACHINE LEARNING WITH PYSPARK

Andrew Collier
Data Scientist, Fathom Data

Bucketing

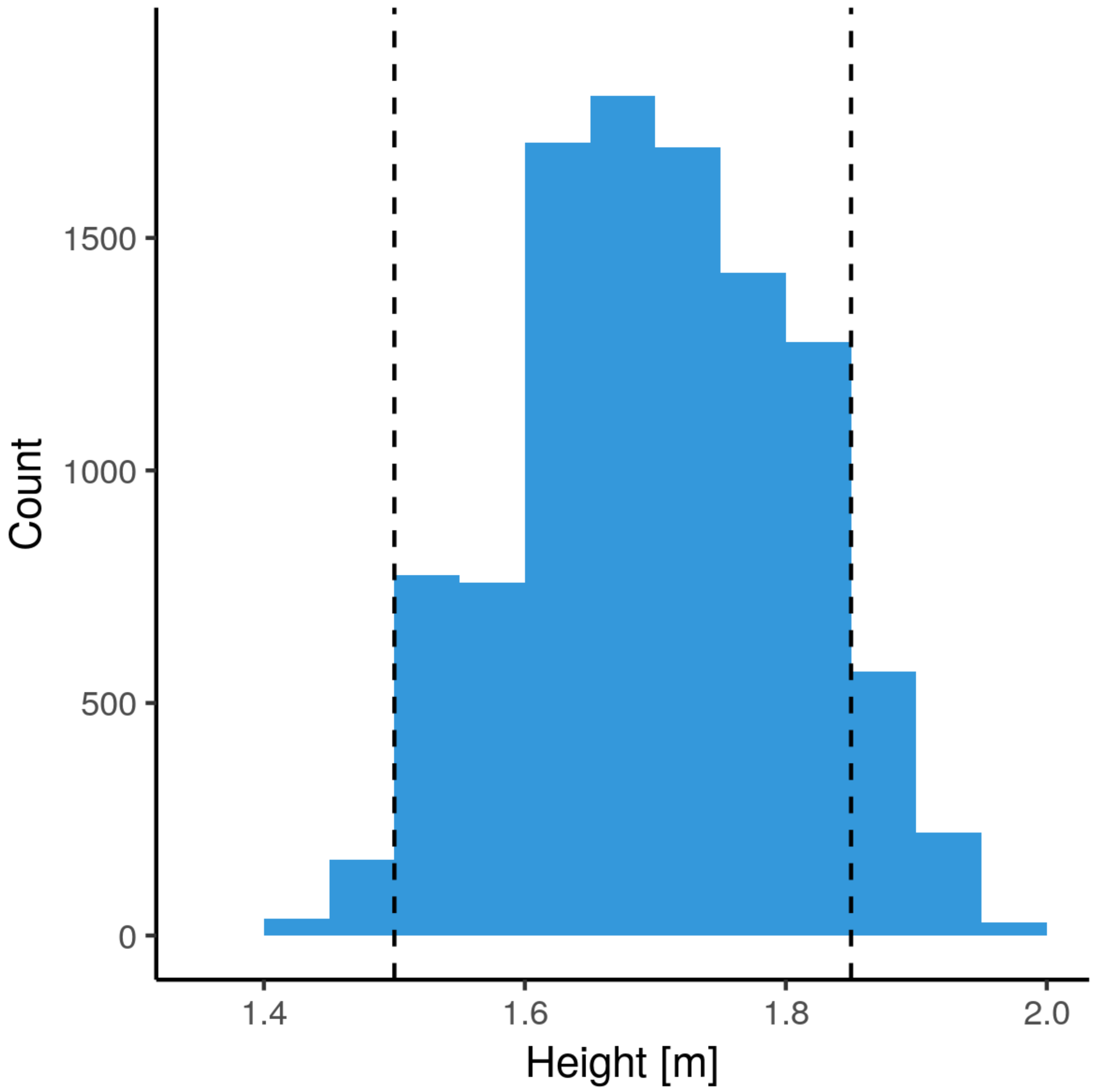


Bucketing heights



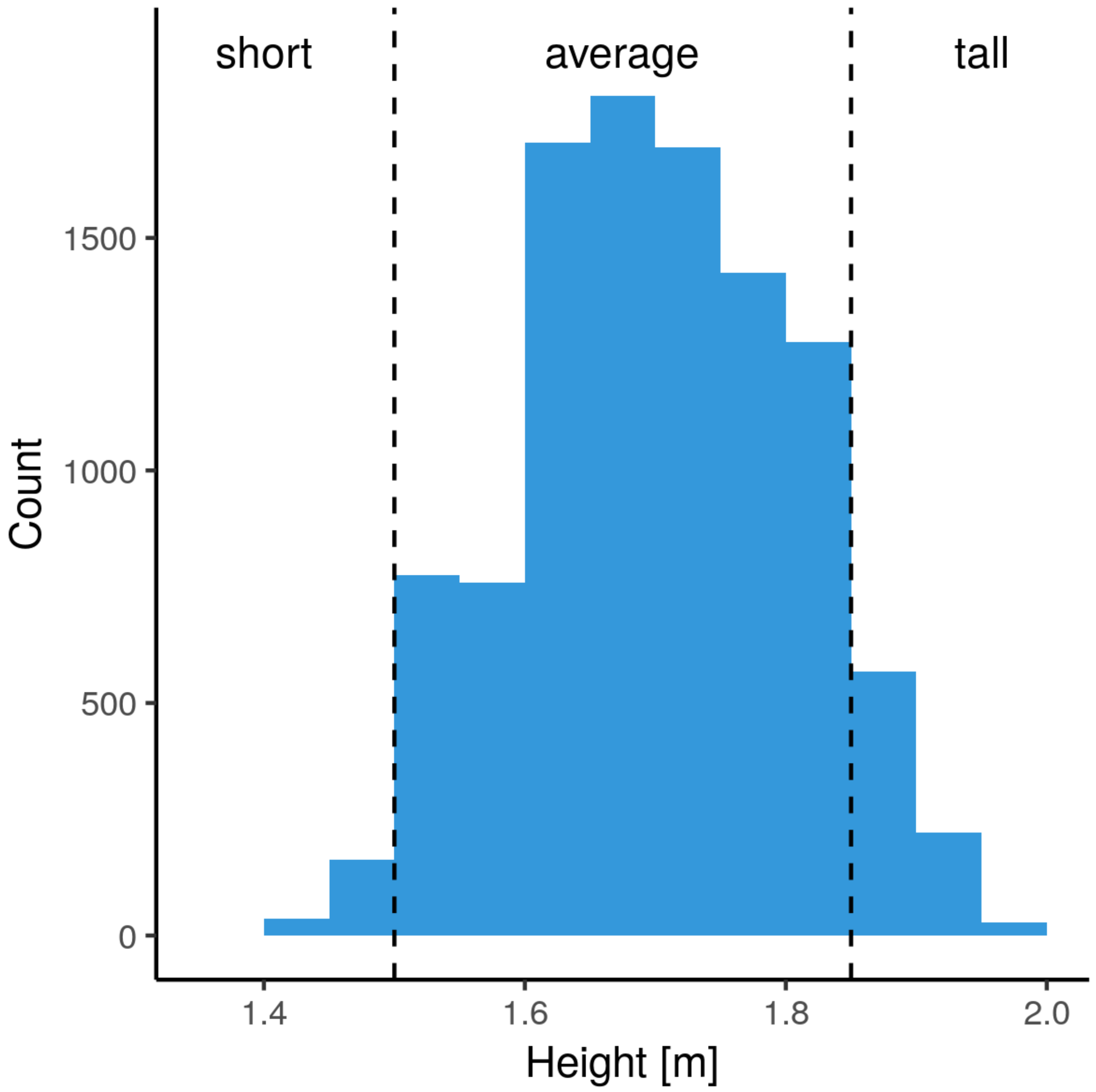

```
+------+
|height|
+------+
|  1.42|
|  1.45|
|  1.47|
|  1.50|
|  1.52|
|  1.57|
|  1.60|
|  1.75|
|  1.85|
|  1.88|
+------+
```

Bucketing heights



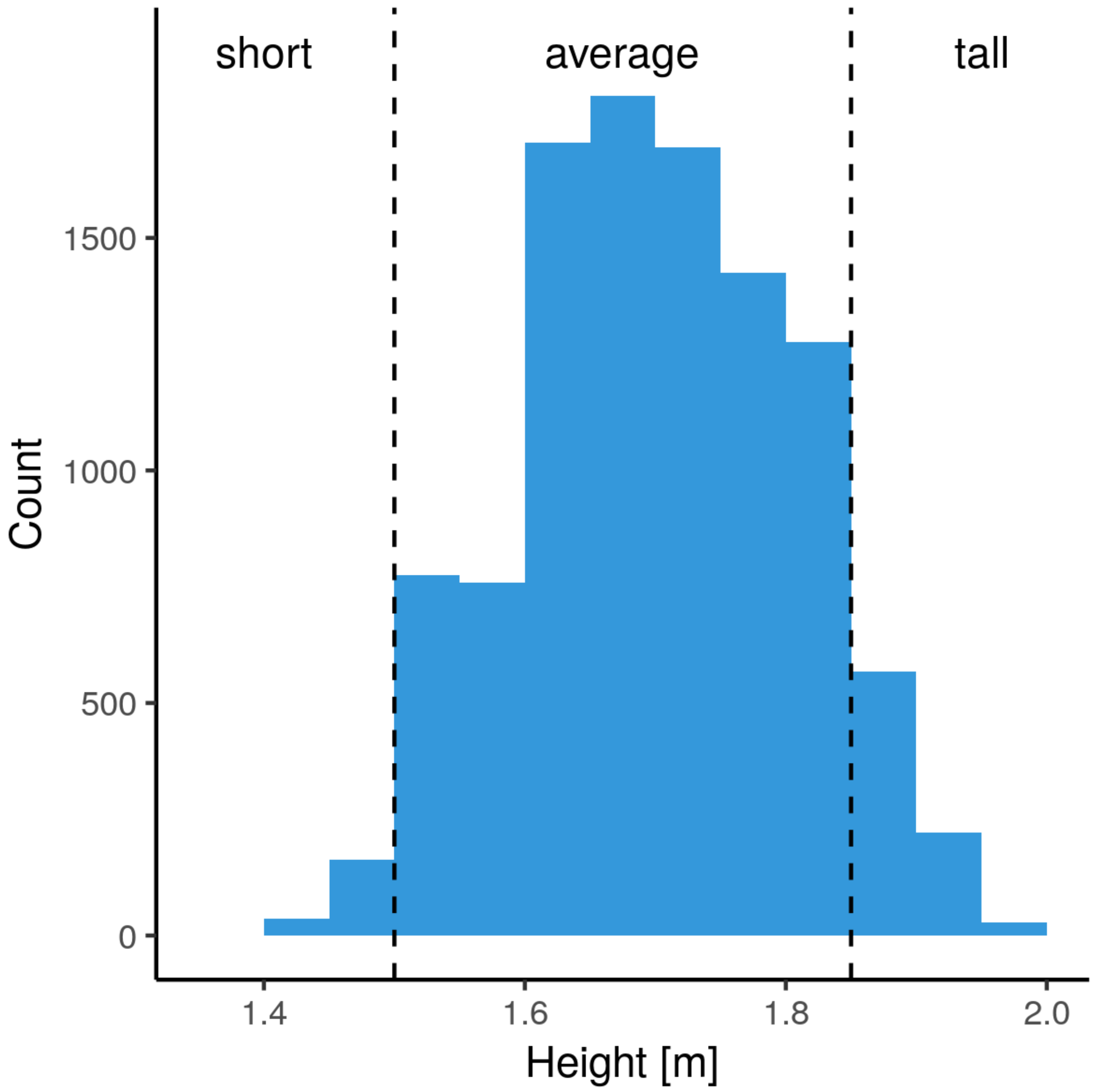

```
+-------+
|height|
+-------+
|   1.42|
|   1.45|
|   1.47|
|   1.50|
|   1.52|
|   1.57|
|   1.60|
|   1.75|
|   1.85|
|   1.88|
+-------+
```

Bucketing heights




```
+------+
|height|
+------+
|  1.42|
|  1.45|
|  1.47|
|  1.50|
|  1.52|
|  1.57|
|  1.60|
|  1.75|
|  1.85|
|  1.88|
+------+
```

Bucketing heights

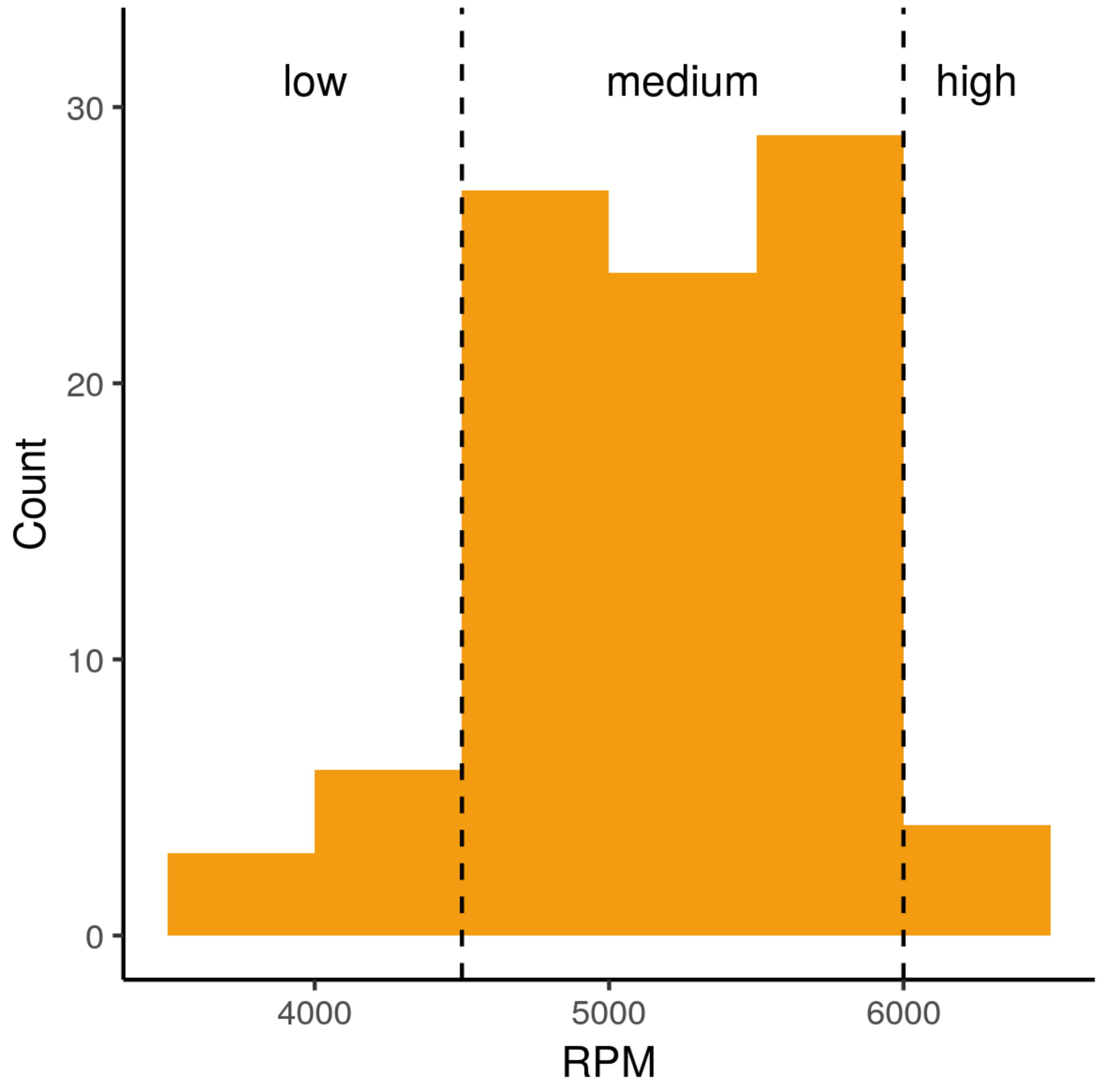



```
+------+----------+
|height|height_bin|
+------+----------+
|  1.42|     short|
|  1.45|     short|
|  1.47|     short|
|  1.50|     short|
|  1.52|   average|
|  1.57|   average|
|  1.60|   average|
|  1.75|   average|
|  1.85|      tall|
|  1.88|      tall|
+------+----------+
```

RPM histogram

Car RPM has "natural" breaks:

- $\mathrm{RPM} < 4500$ — low
- $\mathrm{RPM} > 6000$ — high
- otherwise — medium.



RPM buckets

```
from pyspark.ml.feature import Bucketizer

bucketizer = Bucketizer(splits=[3500, 4500, 6000, 6500],
                        inputCol="rpm",
                        outputCol="rpm_bin")
```

Apply buckets to `rpm` column.

```
bucketed = bucketizer.transform(cars)
```

RPM buckets

```
bucketed.select('rpm', 'rpm_bin').show(5)
```

```
+----+-------+
| rpm|rpm_bin|
+----+-------+
|3800|    0.0|
|4500|    1.0|
|5750|    1.0|
|5300|    1.0|
|6200|    2.0|
+----+-------+
```

```
bucketed.groupBy('rpm_bin').count().show()
```

```
+-------+-----+
|rpm_bin|count|
+-------+-----+
|    0.0|    8| <- low
|    1.0|   67| <- medium
|    2.0|   17| <- high
+-------+-----+
```

One-hot encoded RPM buckets

The RPM buckets are one-hot encoded to dummy variables.

```
+-------+-------------+
|rpm_bin|    rpm_dummy|
+-------+-------------+
|    0.0|(2,[0],[1.0])| <- low
|    1.0|(2,[1],[1.0])| <- medium
|    2.0|    (2,[],[])| <- high
+-------+-------------+
```

The 'high' RPM bucket is the reference level and doesn't get a dummy variable.

Model with bucketed RPM

```
regression.coefficients
```

```
DenseVector([1.3814, 0.1433])
```

```
+-------+-------------+
|rpm_bin|    rpm_dummy|
+-------+-------------+
|    0.0|(2,[0],[1.0])| <- low
|    1.0|(2,[1],[1.0])| <- medium
|    2.0|    (2,[],[])| <- high
+-------+-------------+
```

```
regression.intercept
```

```
8.1835
```

Consumption for 'low' RPM:

```
8.1835 + 1.3814 = 9.5649
```

Consumption for 'medium' RPM:

```
8.1835 + 0.1433 = 8.3268
```

More feature engineering

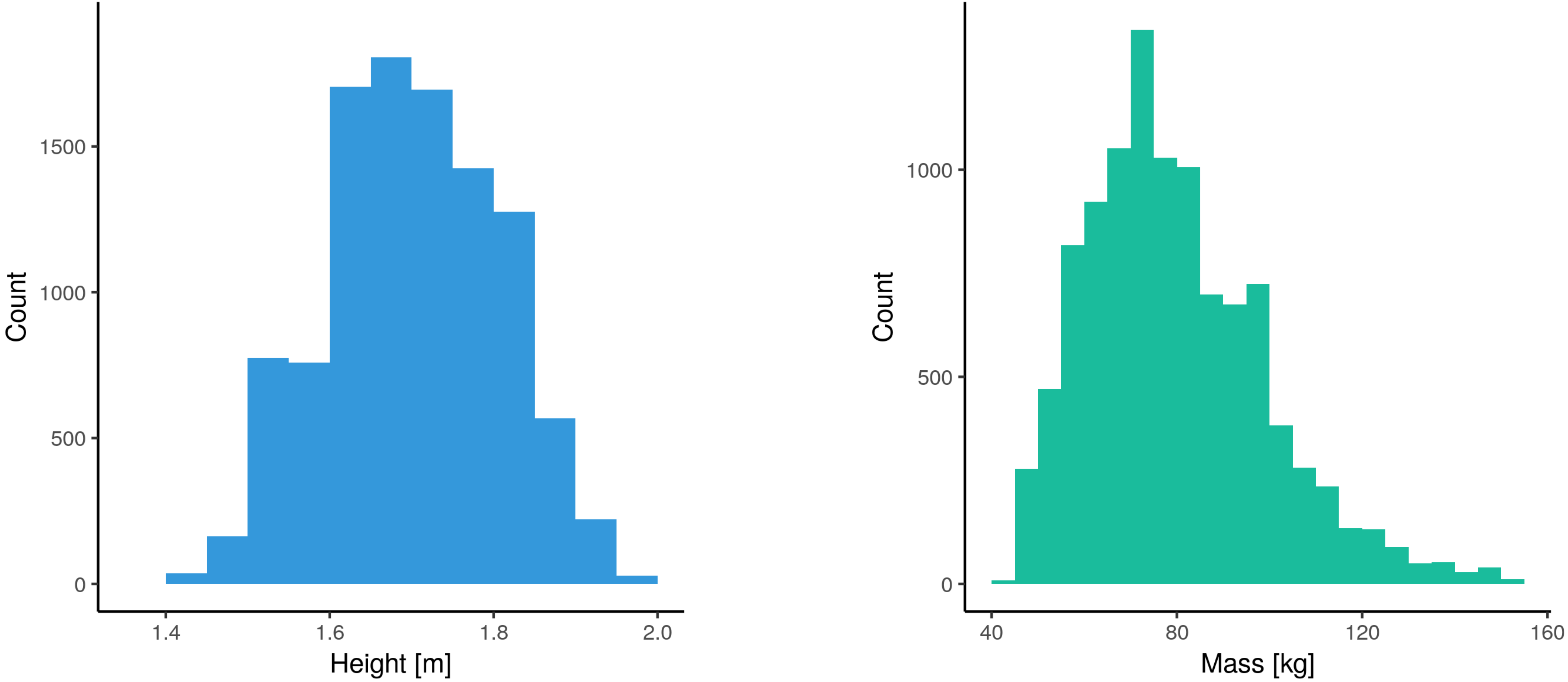
Operations on a single column:

- `log()`
- `sqrt()`
- `pow()`

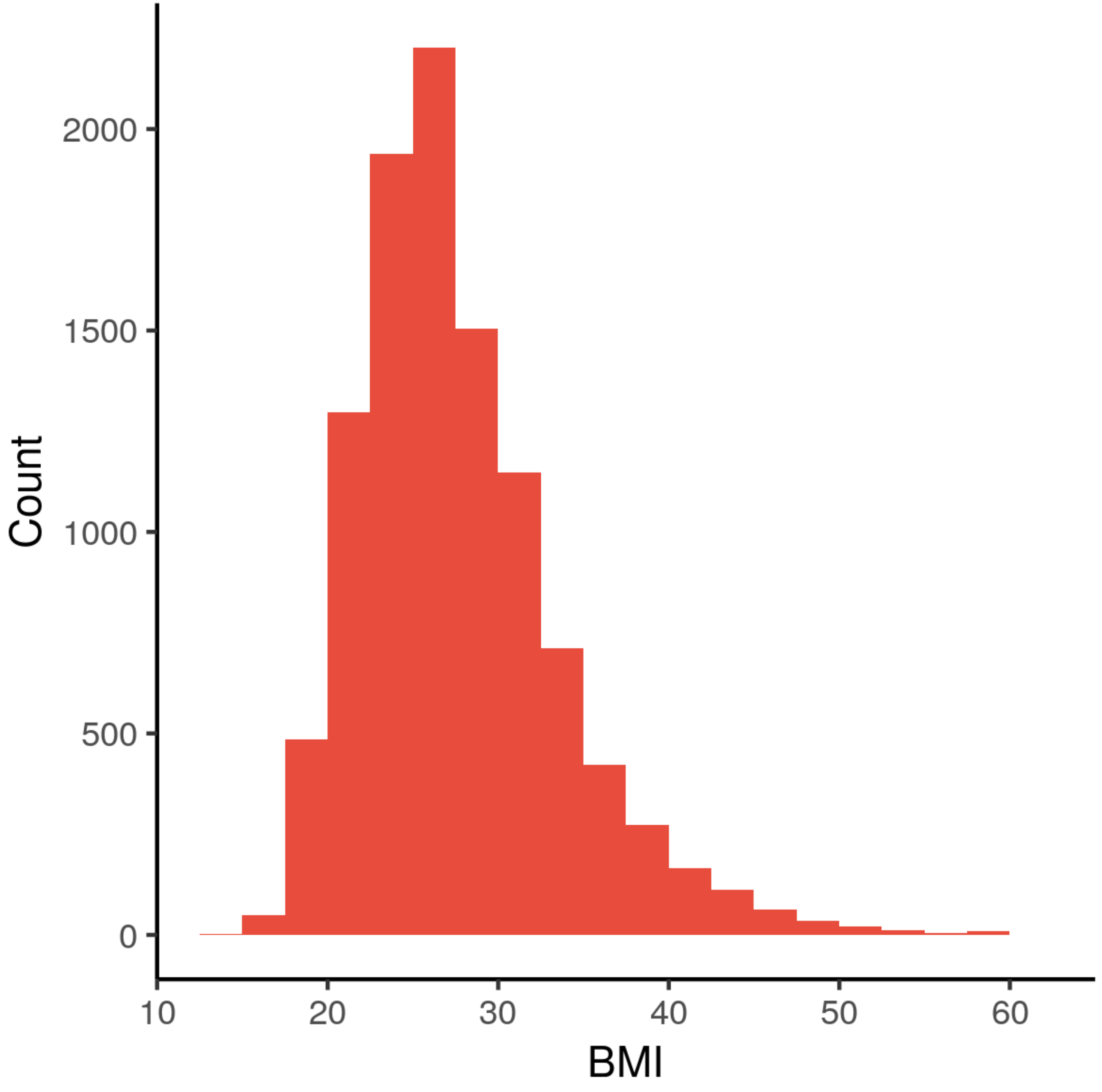
Operations on two columns:

- product
- ratio.

Mass & Height to BMI



Mass & Height to BMI




```
+------+-----+----+
|height| mass| bmi|     bmi = mass / height^2
+------+-----+----+
|  1.52| 77.1|33.2|
|  1.60| 58.1|22.7|
|  1.57|122.0|49.4|
|  1.75| 95.3|31.0|
|  1.80| 99.8|30.7|
|  1.65| 90.7|33.3|
|  1.60| 70.3|27.5|
|  1.78| 81.6|25.8|
|  1.65| 77.1|28.3|
|  1.78|128.0|40.5|
+------+-----+----+
```

Engineering density

```
cars = cars.withColumn('density_line', cars.mass / cars.length)       # Linear density
cars = cars.withColumn('density_quad', cars.mass / cars.length**2)    # Area density
cars = cars.withColumn('density_cube', cars.mass / cars.length**3)    # Volume density
```

```
+------+------+------------+------------+------------+
|  mass|length|density_line|density_quad|density_cube|
+------+------+------------+------------+------------+
|1451.0| 4.775|303.87434554|63.638606397|13.327456837|
|1129.0| 4.623|244.21371403|52.825808790|11.426737787|
|1399.0| 4.547|307.67539036|67.665579583|14.881367843|
+------+------+------------+------------+------------+
```

Let's engineer some features!

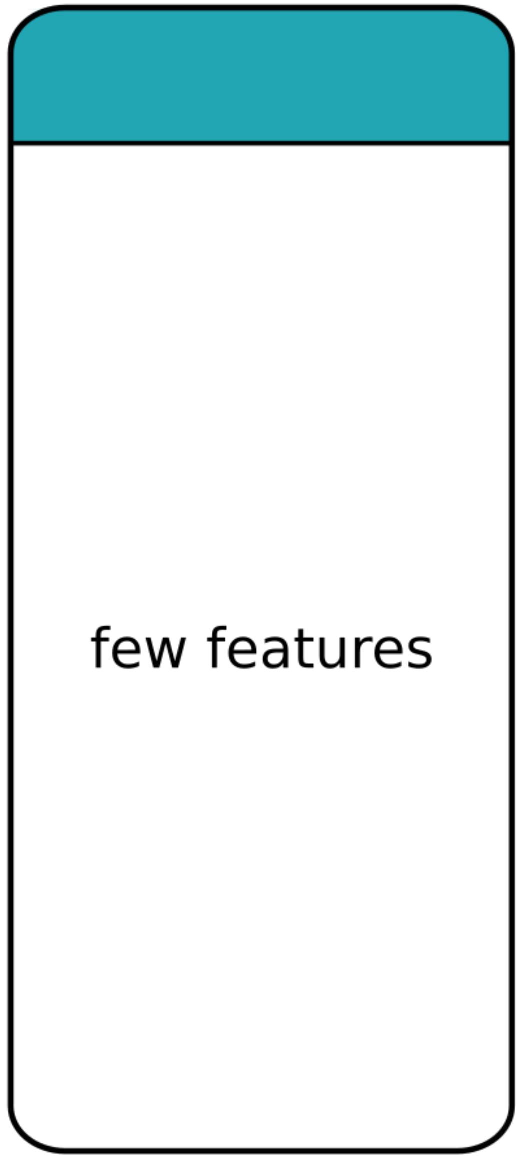
MACHINE LEARNING WITH PYSPARK

Regularization

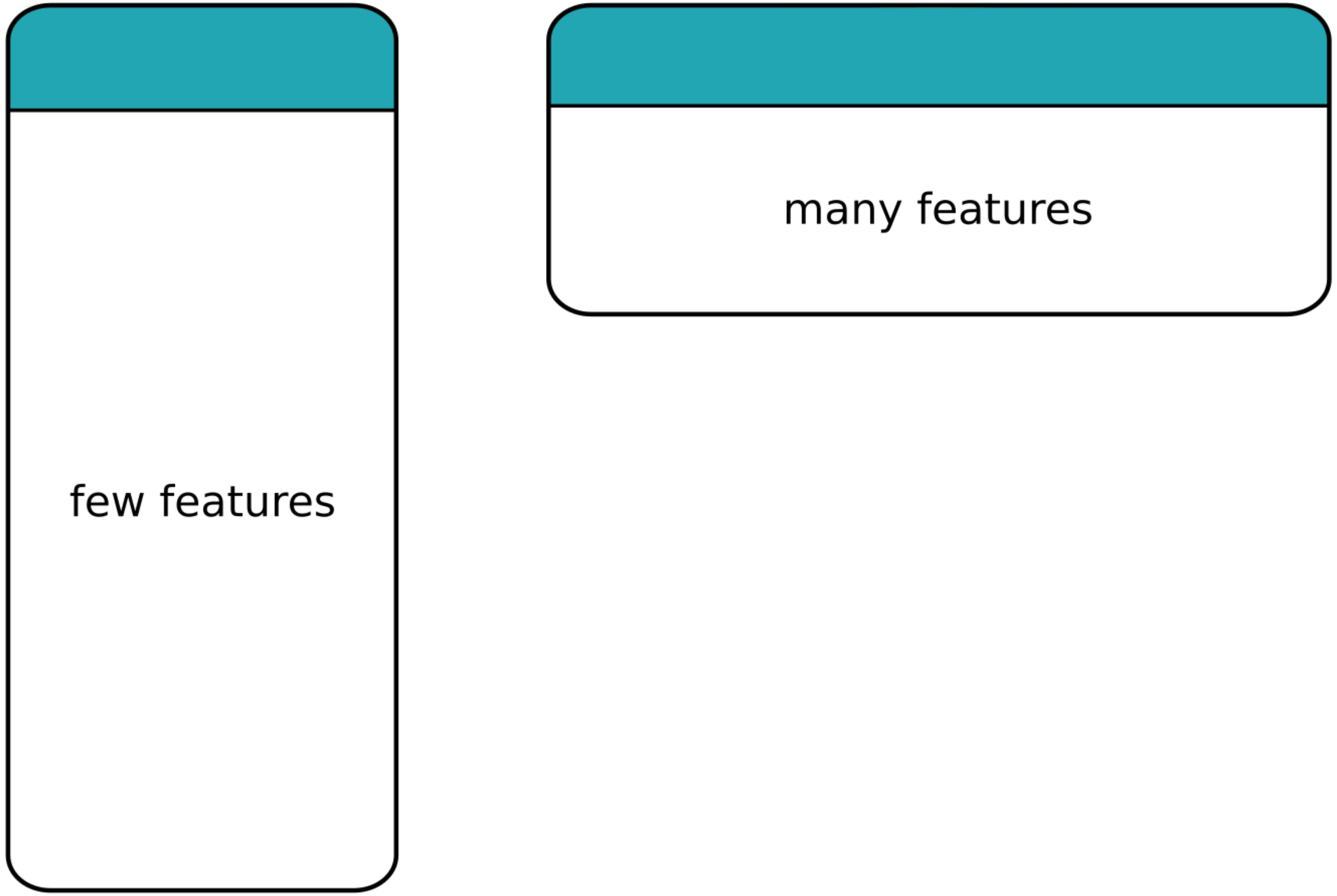
MACHINE LEARNING WITH PYSPARK

Andrew Collier
Data Scientist, Fathom Data

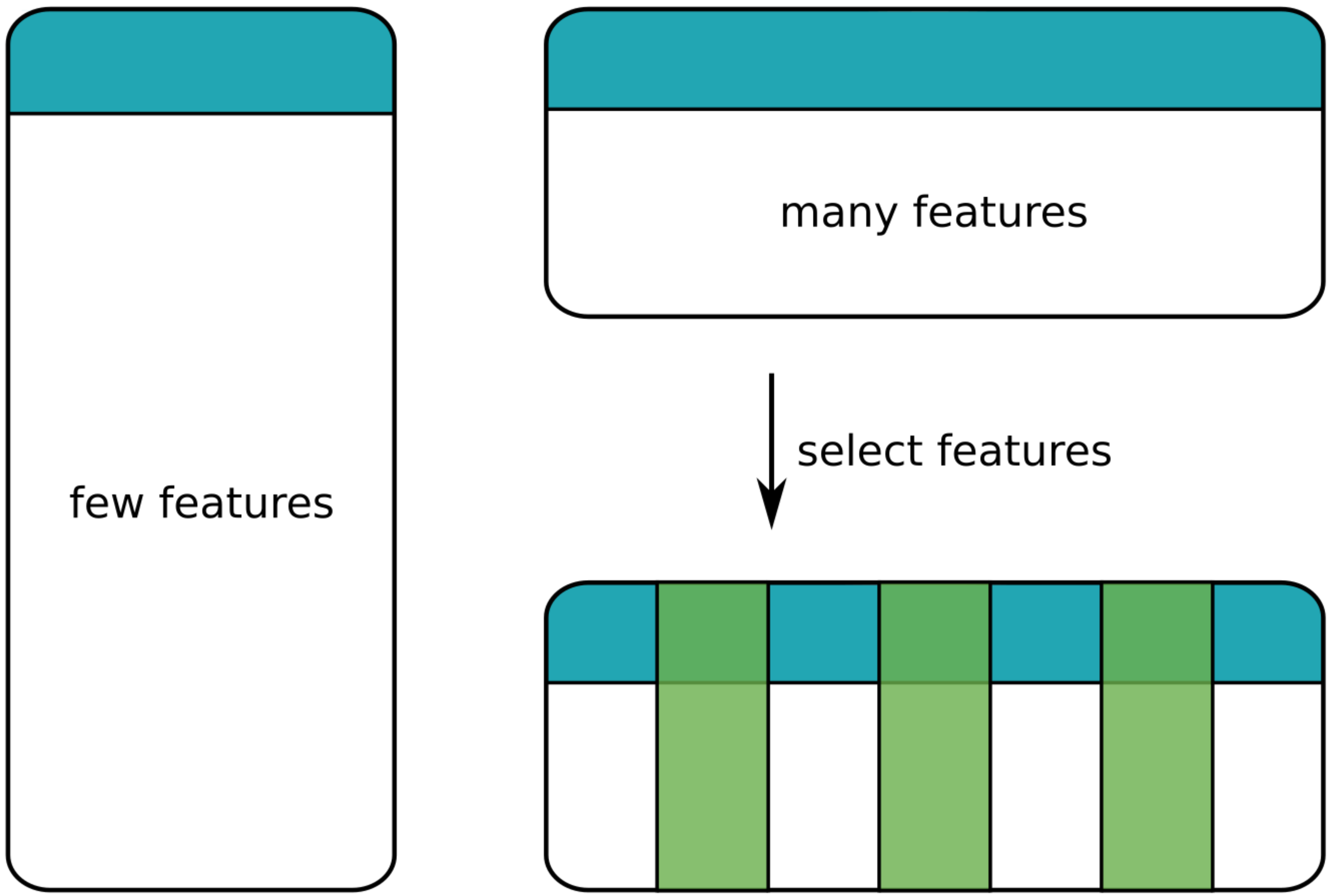
Features: Only a few



Features: Too many



Features: Selected



Loss function (revisited)

Linear regression aims to minimise the MSE.

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y_i})^2$$

Loss function with regularization

Linear regression aims to minimise the MSE.

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 + \boxed{\lambda f(\beta)}$$

Add a *regularization* term which depends on coefficients.

Regularization term

An extra *regularization* term is added to the loss function.

The regularization term can be either

- *Lasso* — absolute value of the coefficients
- *Ridge* — square of the coefficients

It's also possible to have a blend of Lasso and Ridge regression.

Strength of regularization determined by parameter λ:

- $\lambda = 0$ — no regularization (standard regression)
- $\lambda = \infty$ — complete regularization (all coefficients zero)

Cars again

```
assembler = VectorAssembler(inputCols=[
    'mass', 'cyl', 'type_dummy', 'density_line', 'density_quad', 'density_cube'
], outputCol='features')
cars = assembler.transform(cars)
```

```
+-----------------------------------------------------------------------------+-----------+
|features                                                                     |consumption|
+-----------------------------------------------------------------------------+-----------+
|[1451.0,6.0,1.0,0.0,0.0,0.0,0.0,303.8743455497,63.63860639785,13.32745683724]|9.05       |
|[1129.0,4.0,0.0,0.0,1.0,0.0,0.0,244.2137140385,52.82580879050,11.42673778726]|6.53       |
|[1399.0,4.0,0.0,0.0,1.0,0.0,0.0,307.6753903672,67.66557958374,14.88136784335]|7.84       |
|[1147.0,4.0,0.0,1.0,0.0,0.0,0.0,264.1031545014,60.81122599620,14.00212433714]|7.84       |
+-----------------------------------------------------------------------------+-----------+
```

Cars: Linear regression

Fit a (standard) Linear Regression model to the training data.

```
regression = LinearRegression(labelCol='consumption').fit(cars_train)
```

```
# RMSE on testing data
0.708699086182001
```

Examine the coefficients:

```
regression.coefficients
```

```
DenseVector([-0.012, 0.174, -0.897, -1.445, -0.985, -1.071, -1.335, 0.189, -0.780, 1.160])
```

Cars: Ridge regression

```
# alpha = 0 | lambda = 0.1 -> Ridge
ridge = LinearRegression(labelCol='consumption', elasticNetParam=0, regParam=0.1)
ridge.fit(cars_train)
```

```
# RMSE
0.724535609745491
```

```
# Ridge coefficients
DenseVector([ 0.001, 0.137, -0.395, -0.822, -0.450, -0.582, -0.806, 0.008,  0.029, 0.001])
# Linear Regression coefficients
DenseVector([-0.012, 0.174, -0.897, -1.445, -0.985, -1.071, -1.335, 0.189, -0.780, 1.160])
```

Cars: Lasso regression

```
# alpha = 1 | lambda = 0.1 -> Lasso
lasso = LinearRegression(labelCol='consumption', elasticNetParam=1, regParam=0.1)
lasso.fit(cars_train)
```

```
# RMSE
0.771988667026998
```

```
# Lasso coefficients
DenseVector([   0.0,    0.0,    0.0, -0.056,    0.0,    0.0,    0.0, 0.026,    0.0,   0.0])
# Ridge coefficients
DenseVector([ 0.001, 0.137, -0.395, -0.822, -0.450, -0.582, -0.806, 0.008,  0.029, 0.001])
# Linear Regression coefficients
DenseVector([-0.012, 0.174, -0.897, -1.445, -0.985, -1.071, -1.335, 0.189, -0.780, 1.160])
```

Regularization → simple model

MACHINE LEARNING WITH PYSPARK