Computing statistical values

This page lists a number of common statistical computations and how to perform them, often making use of the statistical support libraries in `com.google.common.math`.

In the following examples, a variable with a name like `intArray` or `collectionOfDouble` is of the type implied by that name. The identifier `values` can represent an `int[]`, `long[]`, `double[]`, `Collection<? extends Number>`, or can be replaced with primitive varargs. (In some cases, even more variations may be accepted; check Javadoc for full details.)

Links to named classes are given at the bottom of the page.

Mean (only) of existing values

```
double mean = Stats.meanOf(values);

double mean = doubleStream.average().getAsDouble();
```

Maximum (only) of existing values (ditto for minimum)

```
double max = doubleStream.max().getAsDouble();

double max = Double.max(doubleA, doubleB);  // Java 8+ only

double max = Doubles.max(doubleA, doubleB, doubleC);

double max = Doubles.max(doubleArray);

double max = immutableDoubleArray.stream().max().getAsDouble();

double max = Collections.max(collectionOfDouble);

double max = Ordering.natural().max(iterableOfDouble);
```

Sum (only) of existing values

```
double sum = doubleStream.sum();

double sum = Arrays.stream(doubleArray).sum();

double sum = Stats.of(values).sum();
```

Both mean and maximum of existing values

```
DoubleSummaryStatistics stats = doubleStream.summaryStatistics();
double mean = stats.getAverage();
double max = stats.getMax();

Stats stats = Stats.of(values);
double mean = stats.mean();
double max = stats.max();
```

Standard deviation of existing values

Choose between `populationStandardDeviation` and `sampleStandardDeviation`; see the Javadoc of these methods to understand the difference. You can get other statistics, such as mean, min, and max, from the same `Stats` instance.

```
double stddev = Stats.of(values).populationStandardDeviation();

double stddev = primitiveStream.collect(toStats()).populationStandardDeviation();
```

(The `toStats()` method is statically imported from `Stats`.)

Mean and sample standard deviation of incoming values

This approach is useful when you don't want to store up all the values in advance. Instead, create an "acccumulator", and as you get the values you can feed them in and then discard them.

```
StatsAccumulator accum = new StatsAccumulator();
...

// any number of times, over time
accum.add(value); // or addAll
...

double mean = accum.mean();
double stddev = accum.sampleStandardDeviation();

// or use accum.snapshot() to get an immutable Stats instance
```

Median (only) of existing values

```
double median = Quantiles.median().compute(values);
```

95th percentile of existing values

```
double percentile95 = Quantiles.percentiles().index(95).compute(values);
```

Find the 90th, 99th, and 99.9th percentile

```
Map<Integer, Double> largeValues =
    Quantiles.scale(1000).indexes(900, 990, 999).compute(values);
double p99 = largeValues.get(990); // for example
```

Find the statistical correlation between two sets of values

```
PairedStatsAccumulator accum = new PairedStatsAccumulator();

for (...) {
  ...
  accum.add(x, y);
}

double correl = accum.pearsonsCorrelationCoefficient();
```

Find a linear approximation for a set of ordered pairs

```
PairedStatsAccumulator accum = new PairedStatsAccumulator();

for (...) {
  ...
  accum.add(x, y);
}

LinearTransformation bestFit = accum.leastSquaresFit();
double slope = bestFit.slope();
double yIntercept = bestFit.transform(0);
double estimateXWhenYEquals5 = bestFit.inverse().transform(5);
```

Links to classes used in these examples

- `Arrays`
- `Collections`
- `Doubles`
- `DoubleStream`
- `DoubleSummaryStatistics`
- `ImmutableIntArray`
- `LinearTransformation`
- `Longs`
- `Ordering`
- `PairedStatsAccumulator`
- `Quantiles`
- `Stats`

- `StatsAccumulator`
- `Streams`