

Telemetry Onboarding

Firefox Data Platform Team

Structure

1. Privacy & Policies
2. What is Telemetry?
3. Summary dashboards
4. Telemetry dashboards
5. Adding a probe
6. What is a ping?
7. Data Pipeline
8. Experiments
9. Offline Processing
10. Stats

Before we Begin

<https://analysis.telemetry.mozilla.org/>

Privacy Policy

1. No surprises: use and share information in a way that is transparent and benefits the user
2. User control: develop products and advocate for best practices that put users in control of their data and online experiences
3. Limited data: collect what we need, de-identify where we can and delete when no longer necessary
4. Sensible settings: design for a thoughtful balance of safety and user experience
5. Defense in depths: maintain multi-layered security controls and practices, many of which are publicly verifiable

<https://www.mozilla.org/privacy/>

Data Collection Policy

When proposing a new measurement or data system, consider the requirements and the necessary data properties, e.g:

- Is it necessary to take a measurement from all users? Or is it sufficient to measure only prerelease users?
- Is it desirable to track data changes over time? With what frequency? With what latency?

For every new measurement, even a trivial measurement, please request approval by setting the feedback flag for the module owner or a peer.

Owner: Benjamin Smedberg

Questions?

What is Telemetry?

Telemetry is a system that:

- measures how Firefox behaves in the real world
- collects non-personal information about performance, hardware, and other stuff,
- allows us to identify bugs, issues, and regressions
- allows us to conduct longitudinal studies and experiments

Data Sets

- Base Telemetry (formerly FHR) contains critical, representative data that supports longitudinal studies
 - Enabled by default on all channels
- Extended Telemetry send richer performance and usage data
 - Disabled by default on release
 - Enabled by default on prerelease

- General
- Search
- Content
- Applications
- Privacy
- Security
- Sync
- Advanced

Advanced



General

Data Choices

Network

Update

Certificates



Enable Firefox Health Report

Helps you understand your browser performance and shares data with Mozilla about your browser health

[Learn More](#)



Share additional data (i.e., Telemetry)

Shares performance, usage, hardware and customization data about your browser with Mozilla to help us make Firefox better

[Learn More](#)



Enable Crash Reporter

Firefox submits crash reports to help Mozilla make your browser more stable and secure

[Learn More](#)

```
export MOZ_TELEMETRY_REPORTING=1
#export MOZILLA_OFFICIAL=1
```

about:telemetry

mozilla

Telemetry Data

This page shows the information about performance, hardware, usage and customizations collected by Telemetry. This information is submitted to Mozilla to help improve Mozilla Firefox.

FHR data upload is **enabled**. [Change](#)
Extended Telemetry recording is **enabled**. [Change](#)

Ping data source:

- ☒ Current ping data
☐ Archived ping data

Ping data display:

- ☒ Structured
☐ Raw JSON

☐ Show subsession data

Payload

Parent Payload ▾

General Data [Click to toggle section](#)

Environment Data [Click to toggle section](#)

Session Information [Click to toggle section](#)

Telemetry Log [Click to toggle section](#)

Slow SQL Statements [Click to toggle section](#)

Browser Hangs [Click to toggle section](#)

Thread Hangs [Click to toggle section](#)

Histograms [Click to toggle section](#)

Keyed Histograms [Click to toggle section](#)

Simple Measurements [Click to toggle section](#)

References

<https://wiki.mozilla.org/Telemetry>

https://wiki.mozilla.org/Firefox_Health_Report

Questions?

Summary Dashboards

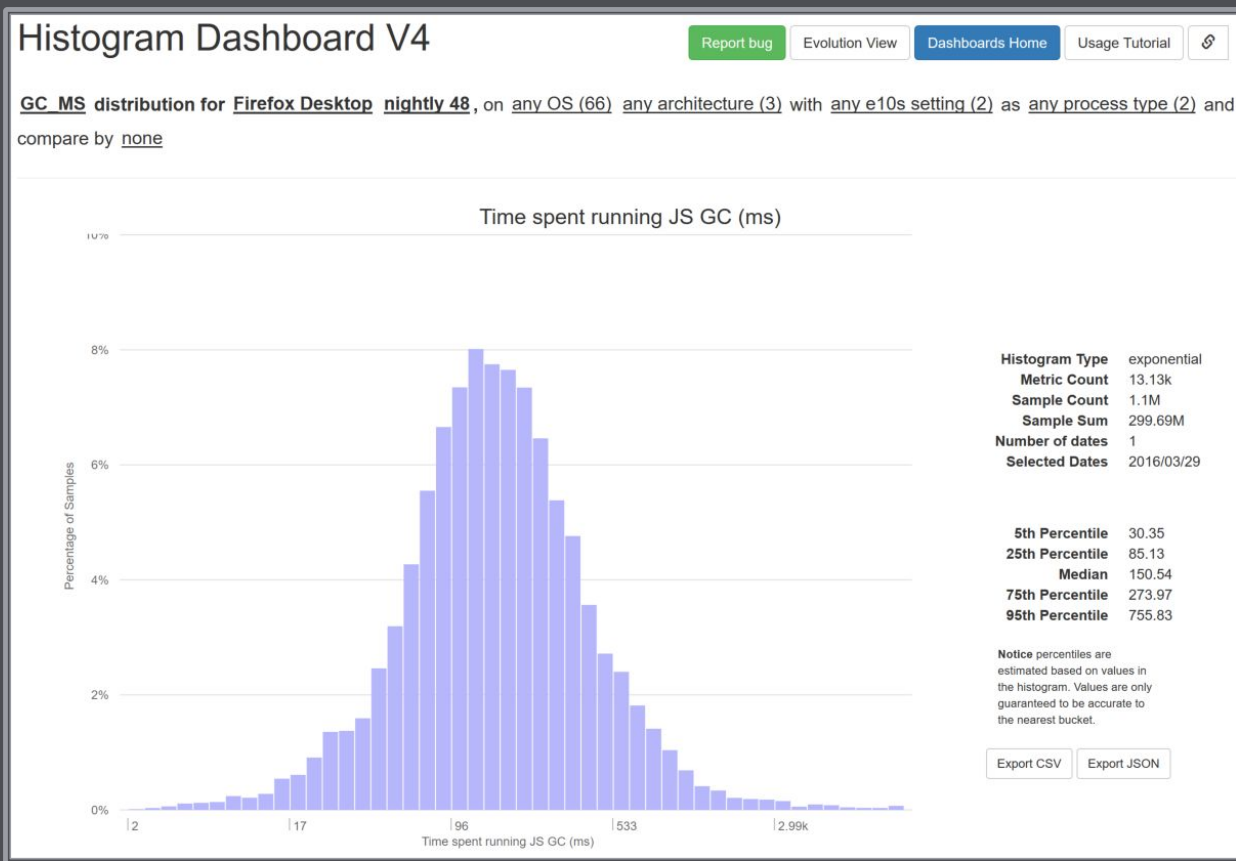
CONFIDENTIAL — Only for staff and contributors under NDA — Do not share

Mozilla Summary Dashboards

<https://metrics.services.mozilla.com/>

Questions?

Telemetry Dashboards



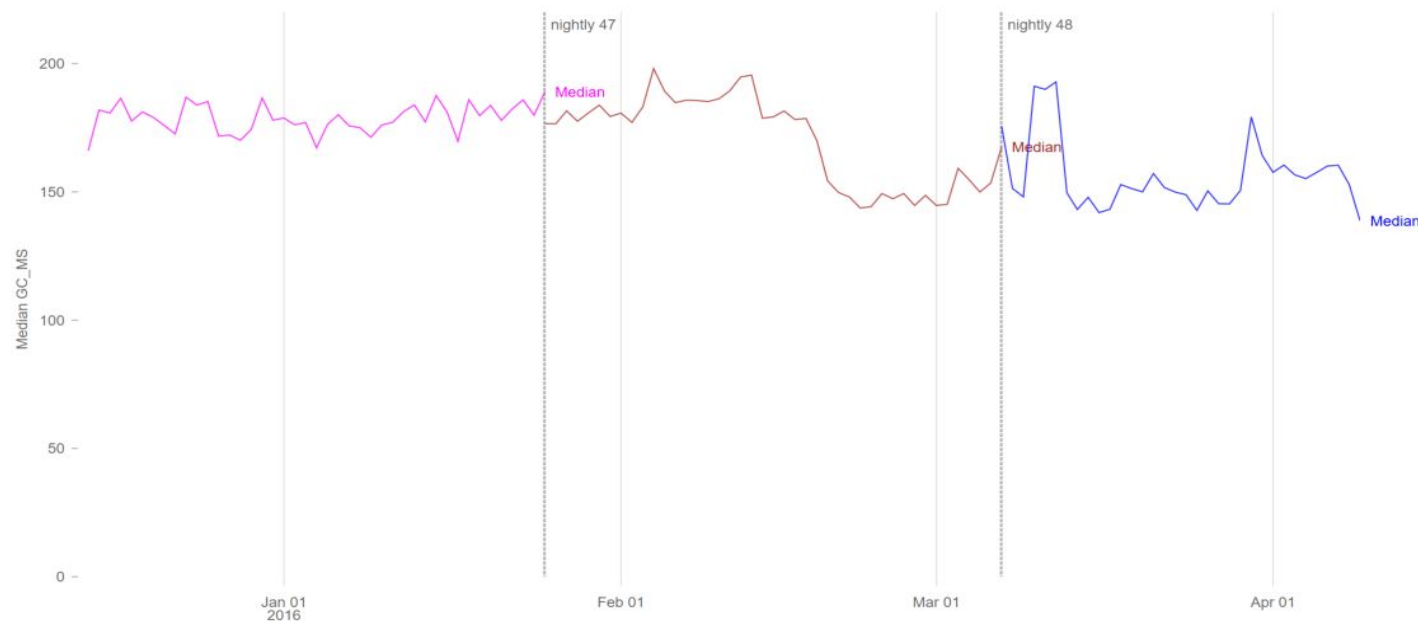
<https://telemetry.mozilla.org/>

Evolution Dashboard V4

[Report bug](#)[Distribution View](#)[Dashboards Home](#)[Usage Tutorial](#)

Median GC_MS from **nightly 46** to **nightly 48** for **Firefox Desktop** on **any OS (77)** **any architecture (3)** with **any e10s setting (2)** as **any process type (2)**

Time spent running JS GC (ms)



[Build ID \(click to use Submission Date\)](#)

<https://telemetry.mozilla.org/new-pipeline/evo.html>

Welcome to the Telemetry Dashboard Generator!

To construct your dashboard: add individually-tailored histogram and evolution plots one at a time, then generate your dashboard.

This will generate a codepen populated with your dashboard for instant preview and customization. Be sure to click 'Save' on it to show it to others or export for self-hosting.

For an Introduction, check the [Dashboard Generator Introduction Blog Post](#)

Channel:

Version:

Metric:

⌵ Filters

Histogram

Evolution

Trim: ☒

Remove buckets that contain < 0.0001% of the population on both sides of the histogram. Makes for a more condensed view.

Compare:

Split the histogram population by their value of this filter, and plot them comparatively.

Sensible Compare: ☒

Some Compare choices have many usually-irrelevant values. Use this to limit them to the most relevant.

Sanitize: ☒

Remove invalid and low-population values from the results.

Add to Dashboard

Channel	Version	Metric	Use Submission Date	Sanitize	Trim	Compare	Sensible Compare	Versions for Evolution	Filters	-
---------	---------	--------	---------------------	----------	------	---------	------------------	------------------------	---------	---

Generate Dashboard

Aggregates by JS

telemetry.js v2:

```
Telemetry.init(function() {  
  var versions = Telemetry.getVersions("nightly/40", "nightly/42");  
  console.log("Versions between nightly 40 to nightly 42 (inclusive):\n" + versions.join("\n"));  
});
```

<https://github.com/mozilla/telemetry-dashboard/blob/master/v2/doc.md>

telemetry-wrapper.js:

```
window.TelemetryWrapper.go(params, parentEl);
```

<https://github.com/mozilla/telemetry-dashboard/tree/gh-pages/wrapper>

Aggregates by HTTPS

Get available channels:

```
curl -X GET http://SERVICE/aggregates_by/build_id/channels/  
["nightly", "beta", "aurora"]
```

Get a list of options for the available dimensions on a given channel and version:

```
curl -X GET "http://SERVICE/filters/?channel=nightly&version=42"  
{  
  "metric": ["A11Y_CONSUMERS", "A11Y_IATABLE_USAGE_FLAG", ...],  
  "application": ["Fennec", "Firefox"],  
  ...  
}
```

Get a list of available build-ids for a given channel:

```
curl -X GET "http://SERVICE/aggregates_by/build_id/channels/nightly/dates/"  
[{"date": "20150630", "version": "42"}, {"date": "20150629", "version": "42"}]
```

Given a set of build-ids, retrieve for each of build-id the aggregated histogram that complies with the requested filters:

```
curl -X GET "http://SERVICE/aggregates_by/build_id/channels/nightly/?version=41&dates=20150615,20150616&metric=GC_MS&filters="<div>{"buckets": [0, ..., 10000],  
  "data": [  
    {"date": "20150615",  
      "count": 239459,  
      "sum": 412346123,  
      "histogram": [309, ..., 5047],  
      "label": ""},  
    {"date": "20150616",  
      "count": 233688,  
      "sum": 402241121,  
      "histogram": [306, ..., 7875],  
      "label": ""}],  
  "kind": "exponential",  
  "description": "Time spent running JS GC (ms)"}>
```

Telemetry alerts

[Login](#)

Histogram Regression
Detector

Telemetry Main-thread IO

Unpacked Add-on File Scan

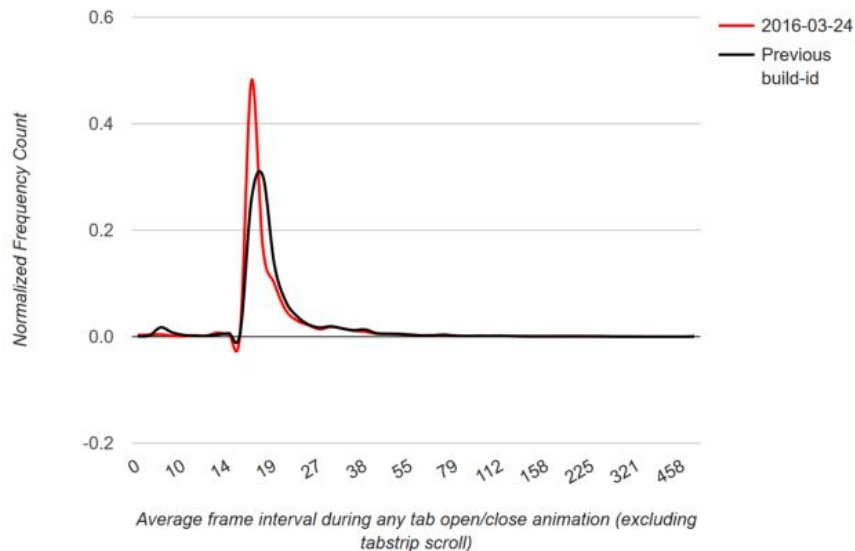
Filter Metrics By:

Select Metric:

From:

To:

FX_TAB_ANIM_ANY_FRAME_INTERVAL_MS



References

<http://robertovitillo.com/2015/07/02/telemetry-metrics-roll-ups/>

<https://anthony-zhang.me/blog/telemetry-demystified/>

<http://robertovitillo.com/2014/07/28/regression-detection-for-telemetry-histograms/>

<https://telemetry.mozilla.org/>

<https://metrics.services.mozilla.com/>

<https://chuttenblog.wordpress.com/tag/telemetry/>

Questions?

Adding a Histogram

Telemetry histograms are the preferred way to track numeric measurements. There are six types:

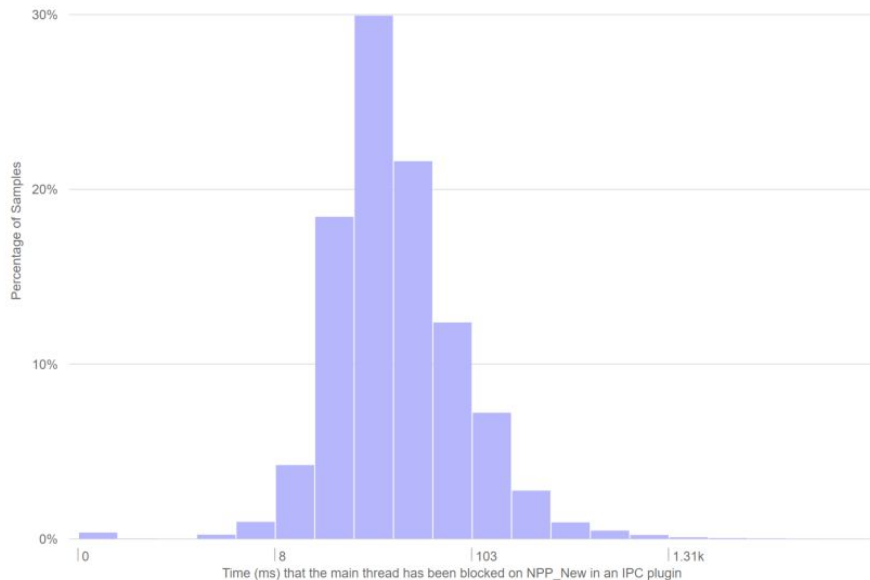
- flag, e.g. FXA_CONFIGURED
- boolean, e.g. E10S_WINDOW
- count, e.g. CONTENT_DOCUMENTS_DESTROYED
- enumerated, e.g. DEVICE_RESET_REASON
- linear, e.g. GC_MAX_PAUSE_MS
- exponential, e.g. GC_MARK_MS

Keyed Histograms

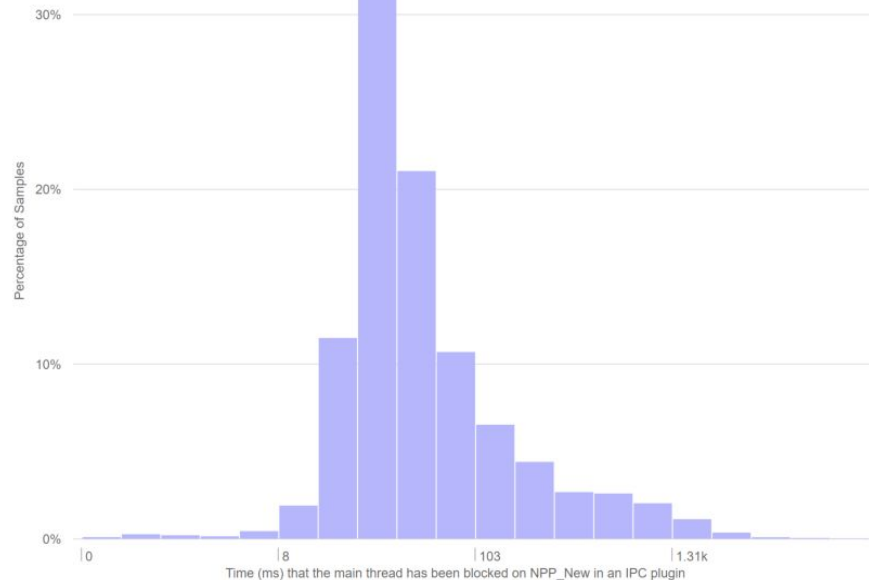
BLOCKED_ON_PLUGIN_INSTANCE_INIT_MS distribution for **Firefox Desktop nightly 48**, on **any OS (66)** **any architecture (3)** with **any e10s setting (2)** as **any process type (2)** and compare by **none**

Time (ms) that the main thread has been blocked on NPP_New in an IPC plugin

Shockwave Flash21.0.0.197



Silverlight Plug-In5.1.41212.0



Declaring a Histogram

toolkit/components/telemetry/Histograms.json

```
"TELEMETRY_TUTORIAL_PROBE": {  
  "alert_emails": ["rvitillo@mozilla.com"],  
  "bug_numbers": [1242013],  
  "expires_in_version": "55",  
  "kind": "exponential",  
  "high": 1000,  
  "n_buckets": 50,  
  "description": "Telemetry tutorial probe (ms)"  
},
```

`./mach build toolkit/components/telemetry`
`./mach run`

Accumulating Data

Go to `about:telemetry` and bring up a console with `Ctrl+Shift+K`

```
Telemetry.isOfficialTelemetry  
<- false  
  
h = Services.telemetry.getHistogramById("TELEMETRY_TUTORIAL_PROBE")  
<- JSHistogram { , 4 more... }  
  
h.add(42)  
<- function add()
```

References

https://developer.mozilla.org/en-US/docs/Mozilla/Performance/Adding_a_new_Telemetry_probe

Questions?

What is a Ping?

A Telemetry ping is the data that we send to Telemetry servers.

On the client it is stored as JSON

Pings follow a common data format.
Different types of pings have different types of data payloads.

Ping Types

“main”: contains most of the measurements that track the performance and health of Firefox instances in the wild. Its “reason” field documents what triggered the ping (e.g. shutdown)

“crash”: captured after the parent Firefox process crashes

others: there are others of more specific utility, and you can also define and submit custom ones if you'd like.

Common Ping Format

```
{
  type: <string>, // "main", "activation", "deletion", "saved-session", ...
  id: <UUID>, // a UUID that identifies this ping
  createDate: <ISO date>, // the date the ping was generated
  version: <number>, // the version of the ping format, currently 4

  application: {
    architecture: <string>, // build architecture, e.g. x86
    buildId: <string>, // "20141126041045"
    name: <string>, // "Firefox"
    version: <string>, // "35.0"
    displayVersion: <string>, // "35.0b3"
    vendor: <string>, // "Mozilla"
    platformVersion: <string>, // "35.0"
    xpcomAbi: <string>, // e.g. "x86-msvc"
    channel: <string>, // "beta"
  },

  clientId: <UUID>, // optional
  environment: { ... }, // optional, not all pings contain the environment
  payload: { ... }, // the actual payload data for this ping type
}
```

Main Ping Payload

The screenshot shows the Mozilla DevTools interface. The top bar includes tabs for Inspector, Console, Debugger, Style Editor, Performance, and Network. The Console tab is active, displaying the following code and output:

```
>> Cu.import("resource://gre/modules/TelemetrySession.jsm")  
← BackstagePass { gLastMemoryPoll: Date 2016-04-11T20:31:47.472Z, gWasDebuggerAttached: false, getLocale: function (), generateUUID: function (), getMsSinceProcessStart: function (), Policy: Object, getPingType: function (), toLocalTimeISOString: function (), annotateCrashReport: function (), processInfo: Object, 144 more... }  
>> TelemetrySession.getPayload()  
← Object { ver: 4, simpleMeasurements: Object, histograms: Object, keyedHistograms: Object, chromeHangs: Object, threadHangStats: Array[6], log: Array[1], webrtc: Object, info: Object, slowSQL: Object, 5 more... }
```

On the right, the 'Object' properties of the returned object are listed:

- UIMeasurements: Array[0]
- addonDetails: Object
- childPayloads: Array[2]
- chromeHangs: Object
- fileIOReports: Object
- histograms: Object
- info: Object
- keyedHistograms: Object
- lateWrites: Object
- log: Array[1]
- simpleMeasurements: Object
- slowSQL: Object
- threadHangStats: Array[6]
- ver: 4
- webrtc: Object
- __proto__: Object

```
Cu.import("resource://gre/modules/TelemetrySession.jsm")
```


E10s Caveat

The screenshot shows the Mozilla DevTools interface. The top bar includes tabs for Inspector, Console, Debugger, Style Editor, Performance, and Network. The Console tab is active, displaying a series of log messages. The first message is a log statement: `Cu.import("resource://gre/modules/TelemetrySession.jsm")`. The second message is a log statement: `BackstagePass { gLastMemoryPoll: Date 2016-04-11T20:31:47.472Z, gWasDebuggerAttached: false, getLocale: function (), generateUUID: function (), getMsSinceProcessStart: function (), Policy: Object, getPingType: function (), toLocalTimeISOString: function (), annotateCrashReport: function (), processInfo: Object, 144 more... }`. The third message is a log statement: `TelemetrySession.getPayload()`. The fourth message is a log statement: `Object { ver: 4, simpleMeasurements: Object, histograms: Object, keyedHistograms: Object, chromeHangs: Object, threadHangStats: Array[6], log: Array[1], webrtc: Object, info: Object, slowSQL: Object, 5 more... }`. The fifth message is a log statement: `TelemetrySession.requestChildPayloads()`. The sixth message is a log statement: `undefined`. The seventh message is a log statement: `TelemetrySession.getPayload()`. The eighth message is a log statement: `Object { ver: 4, simpleMeasurements: Object, histograms: Object, keyedHistograms: Object, chromeHangs: Object, threadHangStats: Array[6], log: Array[1], webrtc: Object, info: Object, slowSQL: Object, 5 more... }`. The Properties panel on the right shows the selected object, which is an `Object`. The properties listed are: `UIMeasurements: Array[0]`, `addonDetails: Object`, `childPayloads: Array[3]`, `0: Object`, `1: Object`, `2: Object`, `length: 3`, `__proto__: Array[0]`, `chromeHangs: Object`, `fileIOReports: Object`, `histograms: Object`, `info: Object`, `keyedHistograms: Object`, `lateWrites: Object`, `log: Array[1]`, `simpleMeasurements: Object`, `slowSQL: Object`, `threadHangStats: Array[6]`, and `ver: 4`.

```
>> Cu.import("resource://gre/modules/TelemetrySession.jsm")
< BackstagePass { gLastMemoryPoll: Date 2016-04-11T20:31:47.472Z, gWasDebuggerAttached: false,
getLocale: function (), generateUUID: function (), getMsSinceProcessStart: function (), Policy:
Object, getPingType: function (), toLocalTimeISOString: function (), annotateCrashReport: function (),
processInfo: Object, 144 more... }

>> TelemetrySession.getPayload()
< Object { ver: 4, simpleMeasurements: Object, histograms: Object, keyedHistograms: Object, chromeHangs:
Object, threadHangStats: Array[6], log: Array[1], webrtc: Object, info: Object, slowSQL: Object, 5
more... }

>> TelemetrySession.requestChildPayloads()
< undefined

>> TelemetrySession.getPayload()
< Object { ver: 4, simpleMeasurements: Object, histograms: Object, keyedHistograms: Object, chromeHangs:
Object, threadHangStats: Array[6], log: Array[1], webrtc: Object, info: Object, slowSQL: Object, 5
more... }
```

Object

- UIMeasurements: Array[0]
- addonDetails: Object
- childPayloads: Array[3]
 - 0: Object
 - 1: Object
 - 2: Object
 - length: 3
- __proto__: Array[0]
- chromeHangs: Object
- fileIOReports: Object
- histograms: Object
- info: Object
- keyedHistograms: Object
- lateWrites: Object
- log: Array[1]
- simpleMeasurements: Object
- slowSQL: Object
- threadHangStats: Array[6]
- ver: 4

Environment

Data that is expected to be characteristic of performance and other behaviour.

Not expected to change too often.

Changes to many of these fields is detected and leads to a session split in the “main” ping.

References

<http://gecko.readthedocs.org/latest/toolkit/components/telemetry/telemetry/index.html>

Questions?



Break Time

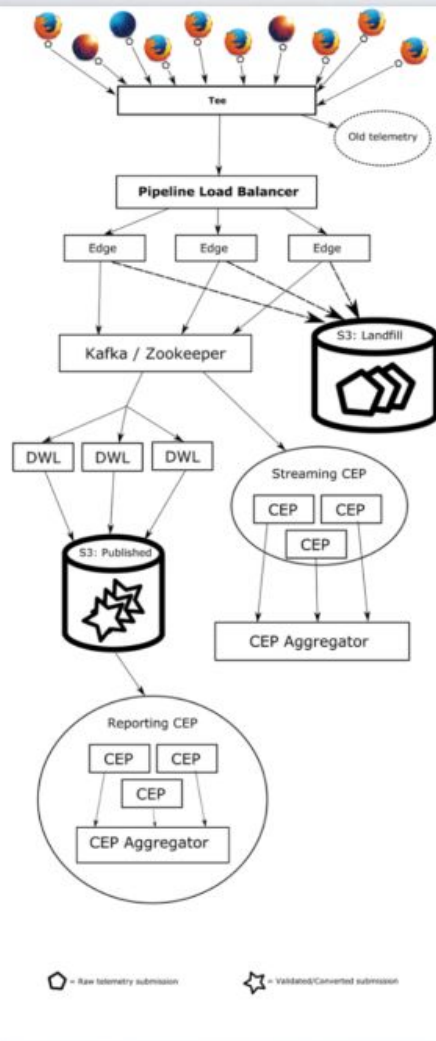
Data Pipeline

How we ingest, transform, store and analyse incoming data

The endpoint is an HTTP server that

- listens for POST/PUT from Firefox
- does some decoding/preprocessing
- sends data off for streaming analyses (Heka)
- archives data to S3 for offline analyses (Spark)

<https://wiki.mozilla.org/CloudServices/DataPipeline>



CEP = Complex Event Processor (basically streaming analysis or reporting)

DWL = Data Warehouse Loader

Landfill = Shorthand for "write-mostly store for backup / recovery purposes"

References

<https://wiki.mozilla.org/CloudServices/DataPipeline>

<https://github.com/mozilla-services/data-pipeline>

<https://github.com/mozilla-services/heka>

<https://github.com/apache/spark/>

Questions?

Offline Processing

IPython, Pandas, and Spark

<https://github.com/vitillo/telemetry-onboarding/tree/master/notebooks>

Offline Processing

Using SQL

<https://sql.telemetry.mozilla.org/>

Questions?

Experiments

Targeted, restartless addons

Currently only available on desktop Firefox

Must undergo Data Collection Review; may require privacy/security reviews as well

Product approval required

<https://wiki.mozilla.org/Telemetry/Experiments>

Flags for Testing

```
experiments.force-sample-value = "0.0"
```

```
experiments.logging.level = 0
```

```
experiments.manifest.cert.checkAttributes =  
false
```

```
experiments.manifest.uri = "http://localhost:  
8000/firefox-manifest.json"
```

```
xpinstall.signatures.required = false
```

```
1 let {classes: Cc, interfaces: Ci, utils: Cu} = Components;
2
3 Cu.import("resource://modules/experiments/Experiments.jsm");
4 Cu.import("resource://gre/modules/Task.jsm");
5 Cu.import("resource://gre/modules/Preferences.jsm");
6
7 var gStarted = false;
8
9 const kSELF_ID = "flash-protectedmode-beta35@experiments.mozilla.org";
10
11 function startup() {
12   // Seems startup() function is launched twice after install, we're
13   // unsure why so far. We only want it to run once.
14   if (gStarted) {
15     return;
16   }
17   gStarted = true;
18
19   Task.spawn(function*() {
20     let branch = yield Experiments.instance().getExperimentBranch(kSELF_ID);
21     switch (branch) {
22       case "control":
23         return;
24       case null:
25         let r = (Math.random() >= 0.5);
26         if (!r) {
27           yield Experiments.instance().setExperimentBranch(kSELF_ID, "control");
28           return;
29         }
30         yield Experiments.instance().setExperimentBranch(kSELF_ID, "experiment");
31         // FALL THROUGH
32       case "experiment":
33         let defaultPrefs = new Preferences({defaultBranch: true});
34         defaultPrefs.set("dom.ipc.plugins.flash.disable-protected-mode", true);
35         return;
36       default:
37         throw new Error("Unexpected experiment branch: " + branch);
38     }
39   }).then(
40     function() {
41     },
42     function(e) {
43       Cu.reportError("Got error during bootstrap startup: " + e);
44     });
45 }
46
47 function shutdown() {
48   let defaultPrefs = new Preferences({defaultBranch: true});
49   defaultPrefs.set("dom.ipc.plugins.flash.disable-protected-mode", false);
50 }
```

```
1 {
2   "publish"      : true,
3   "priority"    : 2,
4   "name"        : "Flash Protected-Mode Testing",
5   "description"  : "Measuring the effect of Flash protected mode on crashes, hangs, and other browser jank.",
6   "info"        : "<p><a href=\"https://bugzilla.mozilla.org/show_bug.cgi?id=1110215\">Related bug</a></p>",
7   "manifest"    : {
8     "id"         : "flash-protectedmode-beta35@experiments.mozilla.org",
9     "startTime"  : 1418601600,
10    "endTime"    : 1421280000,
11    "maxActiveSeconds" : 2764800,
12    "appName"    : ["Firefox"],
13    "channel"    : ["beta"],
14    "os"         : ["WINNT"],
15    "minVersion" : "35.0",
16    "minBuildID" : "20141215000000",
17    "maxVersion" : "37.*",
18    "sample"     : 0.1,
19    "disabled"   : true
20  }
21 }
```


Bug 1111791

“Telemetry report: effect of the Flash protected-mode experiment”

https://bugzilla.mozilla.org/show_bug.cgi?id=1111791

References

<https://wiki.mozilla.org/Telemetry/Experiments>

https://developer.mozilla.org/en-US/Add-ons/Bootstrapped_extensions

https://wiki.mozilla.org/QA/Telemetry#Telemetry_Experiments.2FFHR_Documentation

<http://codefirefox.com/video/install-telemetry-experiment>

<http://hg.mozilla.org/webtools/telemetry-experiment-server/file/tip/experiments>

https://bugzilla.mozilla.org/show_bug.cgi?id=1110215

https://bugzilla.mozilla.org/show_bug.cgi?id=1111791

Questions?

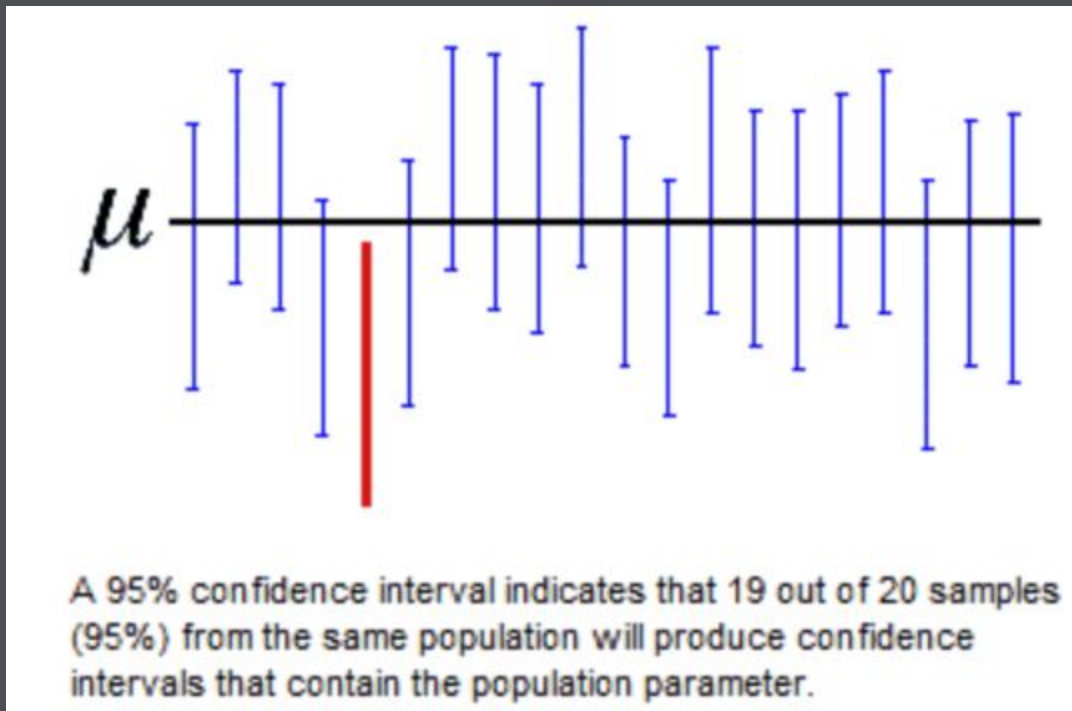


Stats

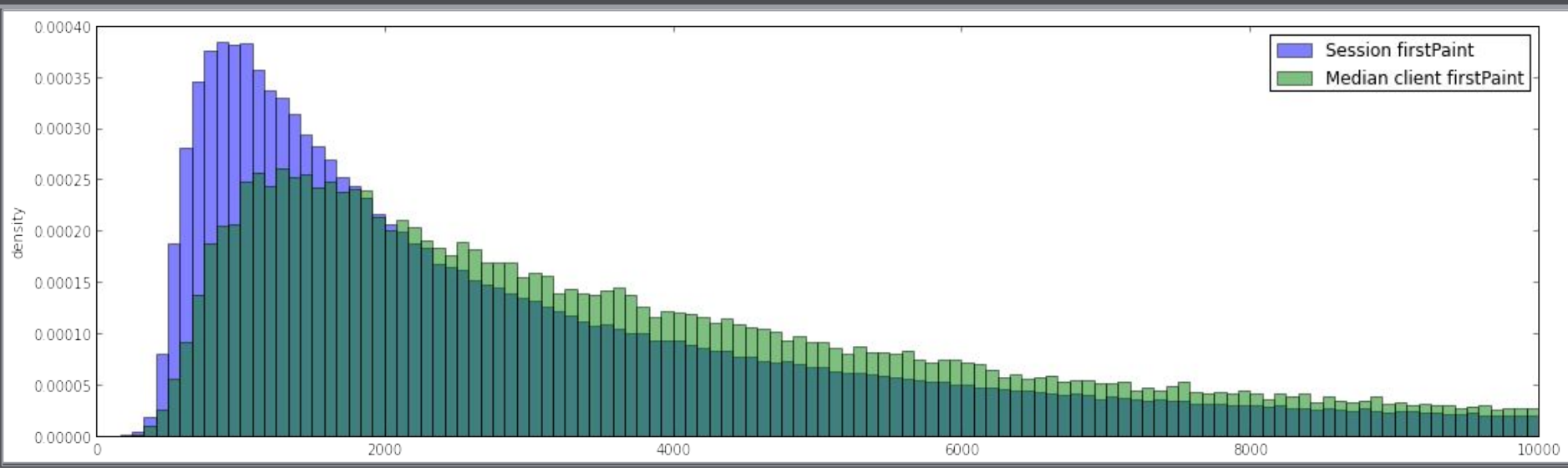
Use Representative Samples



Use Sufficient Data

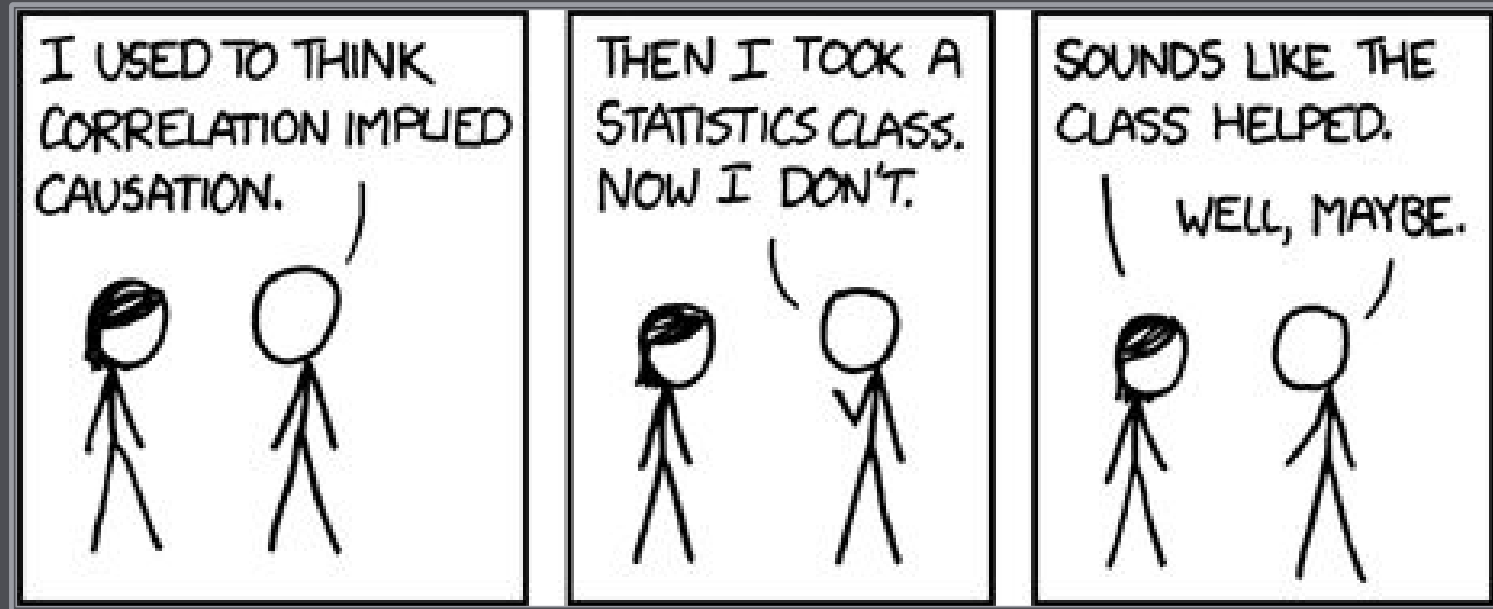


Beware of Pseudoreplication



<http://robertovitillo.com/2014/12/19/clientid-in-telemetry-submissions/>

Correlation is not Causation



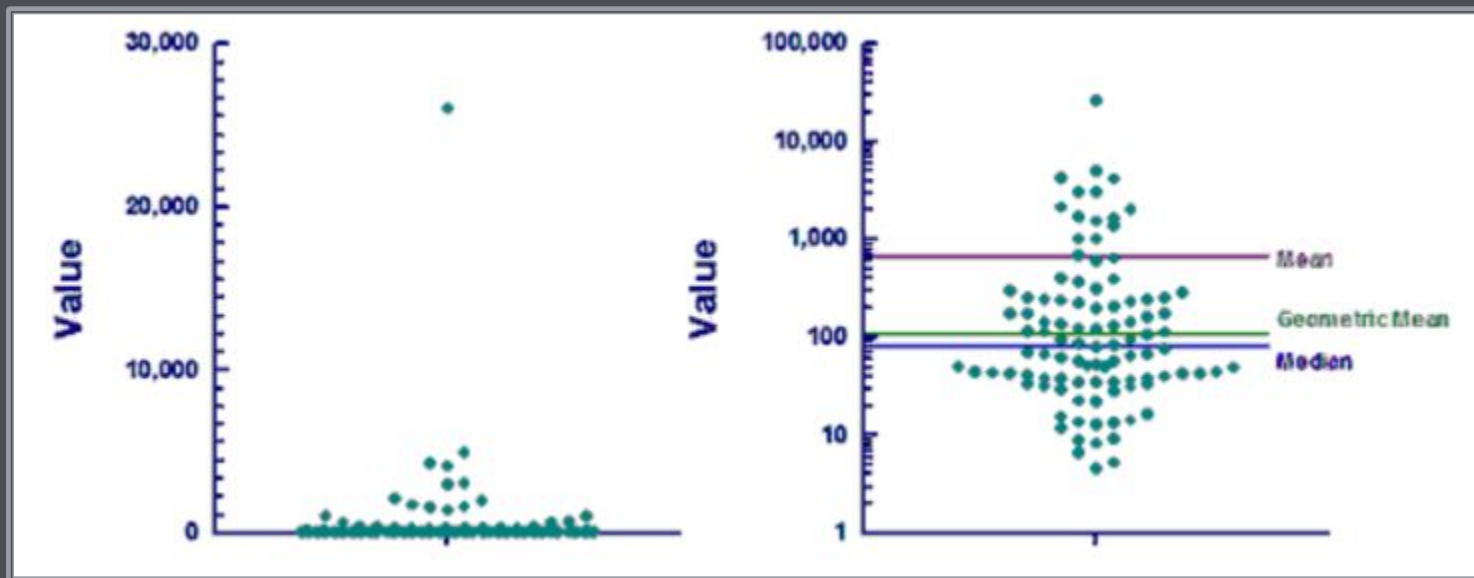
Use the Correct Average

Arithmetic Mean: add up N values, divide by N

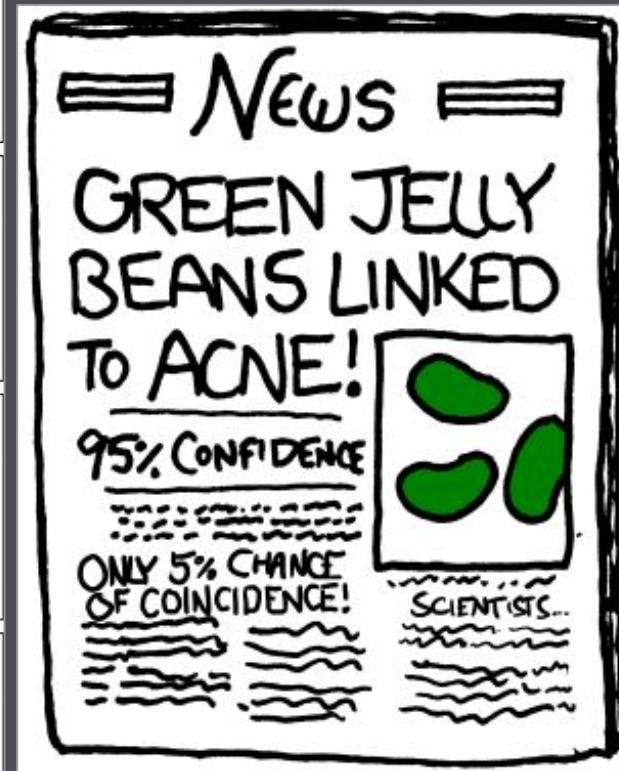
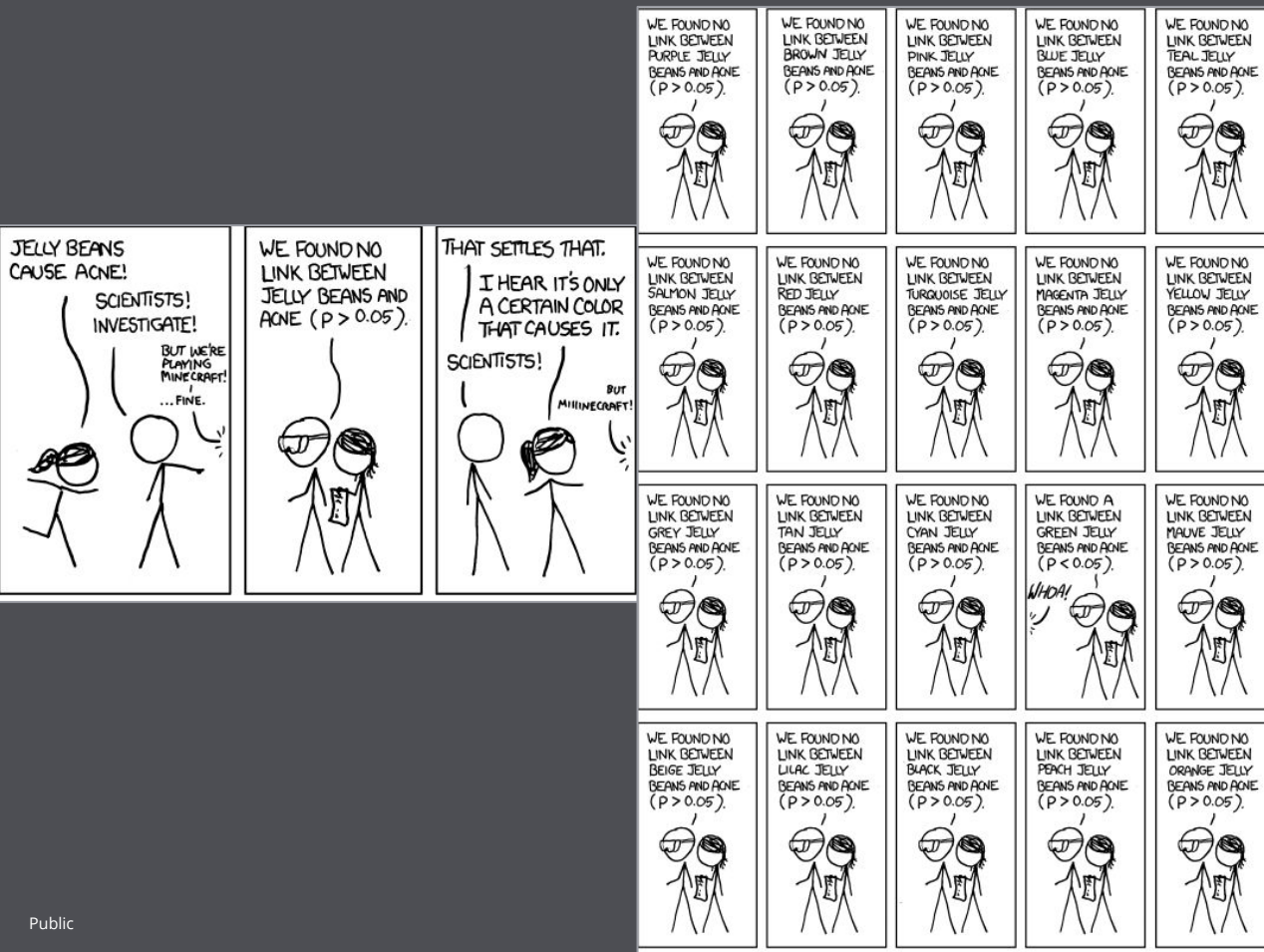
Median: sort the values and pick the middle one

Use the Correct Average

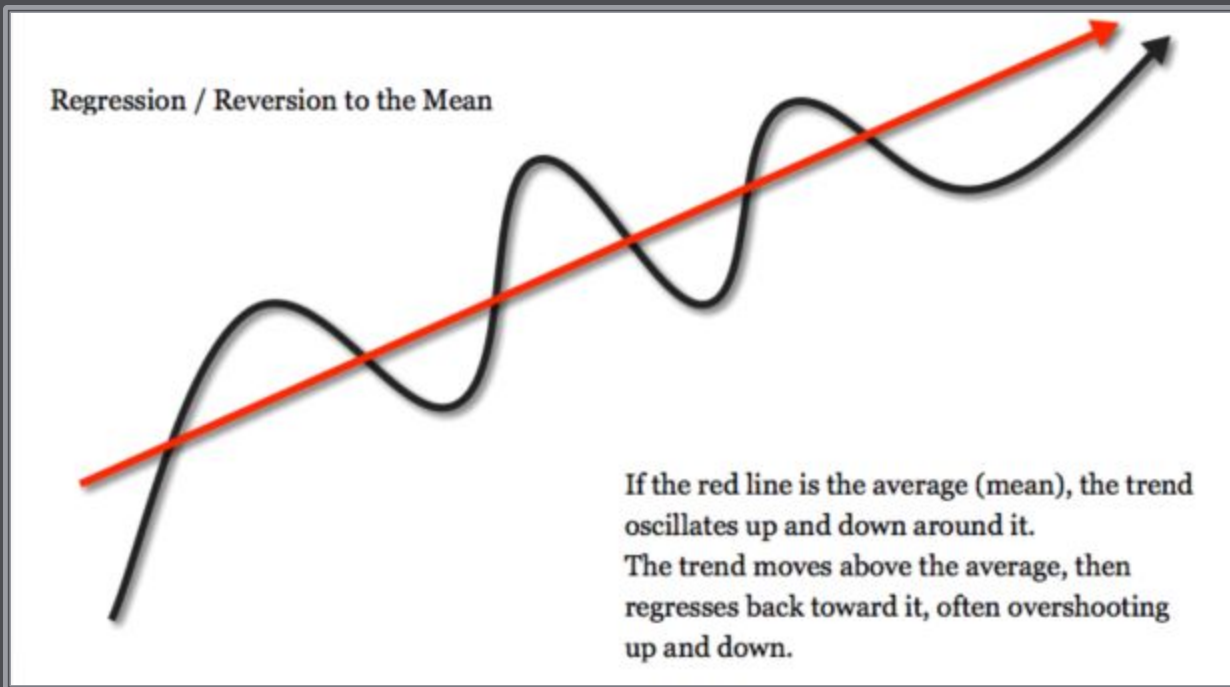
Geometric Mean: take the Nth root of the product of the N values



Control the False Discovery Rate



Regression Towards the Mean



Reproducible Analyses

```
In [1]: import ujson as json
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import plotly.plotly as py

from moztelemetry import get_pings, get_pings_properties, get_one_ping_per_client, get_clients_history

%pylab inline
```

Populating the interactive namespace from numpy and matplotlib

```
In [7]: pings = get_pings(sc, app="Firefox", channel="release", submission_date="20150928", build_id="20150917150946")
```

```
In [8]: def telemetry_enabled(ping):
return ping.get("environment", {}).get("settings", {}).get("telemetryEnabled", False)
```

```
In [*]: pings.count()
```

```
Out[9]: 1242836
```

References

<http://www.statisticsonewrong.com/data-analysis.html>

<http://www.amazon.com/How-Lie-Statistics-Darrell-Huff/dp/0393310728>

<http://www.slideshare.net/RobertoAgostinoViti/all-you-need-to-know-about-statistics>

Questions?

Contact

Don't get frustrated, get help.

IRC: #telemetry #datapipeline

Teams: Measurement, Metrics, Data Engineering

FHR-dev: <https://mail.mozilla.org/listinfo/fhr-dev>

FX-Data-Platform: <https://groups.google.com/a/mozilla.com/forum/#!forum/fx-data-platform>



Fin
