**stats**

**Description**

Calculates aggregate statistics, such as average, count, and sum, over the results set. This is similar to SQL aggregation. If the `stats` command is used without a `BY` clause, only one row is returned, which is the aggregation over the entire incoming result set. If a `BY` clause is used, one row is returned for each distinct value specified in the `BY` clause.

The `stats` command can be used for several SQL-like operations. If you are familiar with SQL but new to SPL, see Splunk SPL for SQL users.

**Difference between stats and eval commands**

The `stats` command calculates statistics based on fields in your events. The `eval` command creates new fields in your events by using existing fields and an arbitrary expression.

<table>
<thead>
<tr>
<th>Time</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>7/18/15</td>
<td>182.236.164.11 - - [18/July/2015:18:20:56] * GET /cart.do?action=addtocart&amp;itemId=EST-14...</td>
</tr>
<tr>
<td>7/18/15</td>
<td>182.236.164.11 - - [18/July/2015:18:21:04] * POST /old/link?itemId=EST18&amp;JSESSIONID=SD6Sl8FF10...</td>
</tr>
</tbody>
</table>

Result set before `stats` command

... | stats count(eval(method="GET")) as GET by host

<table>
<thead>
<tr>
<th>host</th>
<th>GET</th>
</tr>
</thead>
<tbody>
<tr>
<td>www1</td>
<td>8413</td>
</tr>
<tr>
<td>www2</td>
<td>4654</td>
</tr>
</tbody>
</table>

Result set after `stats` command
Syntax

Simple: stats (stats-function(field) [AS field])... [BY field-list]

Complete:

stats [partitions=<num>] [allnum=<bool>] [delim=<string>] ( <stats-agg-term>... | <sparkline-agg-term>... ) [by-clause] [dedup_splitvals]

Required arguments

stats-agg-term

Syntax: <stats-func>(<evaled-field> | <wc-field>) [AS <wc-field>]

Description: A statistical aggregation function. See Stats function options. The function can be applied to an eval expression, or to a field or set of fields. Use the AS clause to place the result into a new field with a name that you specify. You can use wild card characters in field names. For more information on eval expressions, see Types of eval expressions in the Search Manual.

sparkline-agg-term

Syntax: <sparkline-agg> [AS <wc-field>]

Description: A sparkline aggregation function. Use the AS clause to place the result into a new field with a name that you specify. You can use wild card characters in the field name.

Optional arguments

allnum

Syntax: allnum=<bool>

Description: If true, computes numerical statistics on each field if and only if all of the values of that field are numerical.

Default: false

by-clause

Syntax: BY <field-list>

Description: The name of one or more fields to group by. You cannot use a wildcard character to specify multiple fields with similar names. You must specify each field separately. The BY clause returns one row for each distinct value in the BY clause fields. If no BY clause is specified, the stats command returns only one row, which is the aggregation over the entire incoming result set.

dedup_splitvals

Syntax: dedup_splitvals=<boolean>

Description: Specifies whether to remove duplicate values in multivalued BY clause fields.

Default: false

delim

Syntax: delim=<string>

Description: Specifies how the values in the list() or values() aggregation are delimited.

Default: a single space

partitions
Syntax: partitions=<num>
Description: If specified, partitions the input data based on the split-by fields for multithreaded reduce. The partitions argument runs the reduce step (in parallel reduce processing) with multiple threads in the same search process on the same machine. Compare that with parallel reduce, using the redistribute command, that runs the reduce step in parallel on multiple machines.
Default: 1

Stats function options

stats-func
Syntax: The syntax depends on the function that you use. Refer to the table below.
Description: Statistical and charting functions that you can use with the stats command. Each time you invoke the stats command, you can use one or more functions. However, you can only use one BY clause. See Usage.

The following table lists the supported functions by type of function. Use the links in the table to see descriptions and examples for each function. For an overview about using functions with commands, see Statistical and charting functions.

<table>
<thead>
<tr>
<th>Type of function</th>
<th>Supported functions and syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aggregate functions</td>
<td>avg()</td>
</tr>
<tr>
<td></td>
<td>count()</td>
</tr>
<tr>
<td></td>
<td>distinct_count()</td>
</tr>
<tr>
<td></td>
<td>estdc()</td>
</tr>
<tr>
<td></td>
<td>estdc_error()</td>
</tr>
<tr>
<td>Event order functions</td>
<td>first()</td>
</tr>
<tr>
<td>Multivalue stats and chart functions</td>
<td>list()</td>
</tr>
<tr>
<td>Time functions</td>
<td>earliest()</td>
</tr>
<tr>
<td></td>
<td>earliest_time()</td>
</tr>
</tbody>
</table>

Sparkline function options

Sparklines are inline charts that appear within table cells in search results to display time-based trends associated with the primary key of each row. Read more about how to "Add sparklines to your search results" in the Search Manual.

sparkline-agg
Syntax: sparkline (count(<wc-field>), <span-length>) | sparkline (<sparkline-func>(<wc-field>), <span-length>)
Description: A sparkline specifier, which takes the first argument of a aggregation function on a field and an optional timespan specifier. If no timespan specifier is used, an appropriate timespan is chosen based on the time range of the search. If the sparkline is not scoped to a field, only the count aggregator is permitted. You can use wildcard characters in the field name. See the Usage section.

sparkline-func
Syntax: c() | count() | dc() | mean() | avg() | stdev() | stdevp() | var() | varp() | sum() | sumsq() | min() | max() | range()
Description: Aggregation function to use to generate sparkline values. Each sparkline value is produced by
applying this aggregation to the events that fall into each particular time bin.

**Usage**

The `stats` command is a **transforming command**. See Command types.

**Eval expressions with statistical functions**

When you use the `stats` command, you must specify either a statistical function or a sparkline function. When you use a statistical function, you can use an eval expression as part of the statistical function. For example:

```
index=* | stats count(eval(status="404")) AS count_status BY sourcetype
```

**Numeric calculations**

During calculations, numbers are treated as double-precision floating-point numbers, subject to all the usual behaviors of floating point numbers. If the calculation results in the floating-point special value NaN, it is represented as "nan" in your results. The special values for positive and negative infinity are represented in your results as "inf" and "-inf" respectively. Division by zero results in a null field.

There are situations where the results of a calculation contain more digits than can be represented by a floating-point number. In those situations precision might be lost on the least significant digits. For an example of how to correct this, see Example 2 of the basic examples for the sigfig(X) function.

**Functions and memory usage**

Some functions are inherently more expensive, from a memory standpoint, than other functions. For example, the `distinct_count` function requires far more memory than the `count` function. The `values` and `list` functions also can consume a lot of memory.

If you are using the `distinct_count` function without a split-by field or with a low-cardinality split-by by field, consider replacing the `distinct_count` function with the the `estdc` function (estimated distinct count). The `estdc` function might result in significantly lower memory usage and run times.

**Memory and maximum results**

In the `limits.conf` file, the `maxresultrows` setting in the `[searchresults]` stanza specifies the maximum number of results to return. The default value is 50,000. Increasing this limit can result in more memory usage.

The `max_mem_usage_mb` setting in the `[default]` stanza is used to limit how much memory the `stats` command uses to keep track of information. If the `stats` command reaches this limit, the command stops adding the requested fields to the search results. You can increase the limit, contingent on the available system memory.

If you are using Splunk Cloud and want to change either of these limits, file a Support ticket.

**Event order functions**

Using the `first` and `last` functions when searching based on time does not produce accurate results.

- To locate the first value based on time order, use the `earliest` function, instead of the `first` function.
- To locate the last value based on time order, use the `latest` function, instead of the `last` function.
For example, consider the following search.

```
index=test sourcetype=testDb | eventstats first(LastPass) as LastPass, last(_time) as mostRecentTestTime BY testCaseId | where startTime==LastPass OR _time==mostRecentTestTime | stats first(startTime) AS startTime, first(status) AS status, first(histID) AS currentHistId, last(histID) AS lastPassHistId BY testCaseId
```

Replace the `first` and `last` functions when you use the `stats` and `eventstats` commands for ordering events based on time. The following search shows the function changes.

```
index=test sourcetype=testDb | eventstats latest(LastPass) AS LastPass, earliest(_time) AS mostRecentTestTime BY testCaseId | where startTime==LastPass OR _time==mostRecentTestTime | stats latest(startTime) AS startTime, latest(status) AS status, latest(histID) AS currentHistId, earliest(histID) AS lastPassHistId BY testCaseId
```

**Wildcards in BY clauses**

The `stats` command does not support wildcard characters in field values in BY clauses.

For example, you cannot specify `| stats count BY source*`.

**Renaming fields**

You cannot rename one field with multiple names. For example if you have field A, you cannot rename A as B, A as C. The following example is not valid.

```
... | stats first(host) AS site, first(host) AS report
```

**Basic examples**

1. **Return the average transfer rate for each host**

```
sourcetype=access* | stats avg(kbps) BY host
```

2. **Search the access logs, and return the total number of hits from the top 100 values of “referer_domain”**

Search the access logs, and return the total number of hits from the top 100 values of “referer_domain”. The “top” command returns a count and percent value for each “referer_domain”.

```
sourcetype=access_combined | top limit=100 referer_domain | stats sum(count) AS total
```

3. **Calculate the average time for each hour for similar fields using wildcard characters**

Return the average, for each hour, of any unique field that ends with the string "lay". For example, delay, xdelay, relay, etc.

```
... | stats avg(*lay) BY date_hour
```

4. **Remove duplicates in the result set and return the total count for the unique results**

Remove duplicates of results with the same "host" value and return the total count of the remaining results.

```
... | stats dc(host)
```
5. In a multivalue BY field, remove duplicate values

For each unique value of \texttt{mvfield}, return the average value of \texttt{field}. Deduplicates the values in the \texttt{mvfield}.

\[ \text{...| stats avg(field) BY mvfield dedup\_splitvals=true} \]

Extended examples

1. Compare the difference between using the \texttt{stats} and \texttt{chart} commands

This example uses the sample data from the Search Tutorial but should work with any format of Apache web access log. To try this example on your own Splunk instance, you must download the sample data and follow the instructions to get the tutorial data into Splunk. Use the time range \texttt{All time} when you run the search.

This search uses the \texttt{stats} command to count the number of events for a combination of HTTP status code values and host:

\[ \text{source\_type=access\_* | stats count BY status, host} \]

The BY clause returns one row for each distinct value in the BY clause fields. In this search, because two fields are specified in the BY clause, every unique combination of status and host is listed on separate row.

The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>status</th>
<th>host</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>www1</td>
<td>11835</td>
</tr>
<tr>
<td>200</td>
<td>www2</td>
<td>11186</td>
</tr>
<tr>
<td>200</td>
<td>www3</td>
<td>11261</td>
</tr>
<tr>
<td>400</td>
<td>www1</td>
<td>233</td>
</tr>
<tr>
<td>400</td>
<td>www2</td>
<td>257</td>
</tr>
<tr>
<td>400</td>
<td>www3</td>
<td>211</td>
</tr>
<tr>
<td>403</td>
<td>www2</td>
<td>228</td>
</tr>
<tr>
<td>404</td>
<td>www1</td>
<td>244</td>
</tr>
<tr>
<td>404</td>
<td>www2</td>
<td>209</td>
</tr>
</tbody>
</table>

If you click the Visualization tab, the \texttt{status} field forms the X-axis and the \texttt{host} and \texttt{count} fields form the data series. The problem with this chart is that the host values (www1, www2, www3) are strings and cannot be measured in a chart.

Substitute the \texttt{chart} command for the \texttt{stats} command in the search.

\[ \text{source\_type=access\_* | chart count BY status, host} \]

With the \texttt{chart} command, the two fields specified after the BY clause change the appearance of the results on the Statistics tab. The BY clause also makes the results suitable for displaying the results in a chart visualization.

- The first field you specify is referred to as the <row-split> field. In the table, the values in this field become the labels for each row. In the chart, this field forms the X-axis.
• The second field you specify is referred to as the <column-split> field. In the table, the values in this field are used as headings for each column. In the chart, this field forms the data series.

The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>status</th>
<th>www1</th>
<th>www2</th>
<th>www3</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>11835</td>
<td>11186</td>
<td>11261</td>
</tr>
<tr>
<td>400</td>
<td>233</td>
<td>257</td>
<td>211</td>
</tr>
<tr>
<td>403</td>
<td>0</td>
<td>288</td>
<td>0</td>
</tr>
<tr>
<td>404</td>
<td>244</td>
<td>209</td>
<td>237</td>
</tr>
<tr>
<td>406</td>
<td>258</td>
<td>228</td>
<td>224</td>
</tr>
<tr>
<td>408</td>
<td>267</td>
<td>243</td>
<td>246</td>
</tr>
<tr>
<td>500</td>
<td>225</td>
<td>262</td>
<td>246</td>
</tr>
<tr>
<td>503</td>
<td>324</td>
<td>299</td>
<td>329</td>
</tr>
<tr>
<td>505</td>
<td>242</td>
<td>0</td>
<td>238</td>
</tr>
</tbody>
</table>

If you click the Visualization tab, the status field forms the X-axis, the values in the host field form the data series, and the Y-axis shows the count.

2. Use eval expressions to count the different types of requests against each Web server

This example uses the sample data from the Search Tutorial but should work with any format of Apache web access log. To try this example on your own Splunk instance, you must download the sample data and follow the instructions to get the tutorial data into Splunk. Use the time range All time when you run the search.

Run the following search to use the stats command to determine the number of different page requests, GET and POST, that occurred for each Web server.

```
sourcetype=access_* | stats count(eval(method="GET")) AS GET, count(eval(method="POST")) AS POST BY host
```

This example uses eval expressions to specify the different field values for the stats command to count.

• The first clause uses the count() function to count the Web access events that contain the method field value GET. Then, using the AS keyword, the field that represents these results is renamed GET.
• The second clause does the same for POST events.
• The counts of both types of events are then separated by the web server, using the BY clause with the host field.

The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>host</th>
<th>GET</th>
<th>POST</th>
</tr>
</thead>
<tbody>
<tr>
<td>www1</td>
<td>8431</td>
<td>5197</td>
</tr>
<tr>
<td>www2</td>
<td>8097</td>
<td>4815</td>
</tr>
<tr>
<td>www3</td>
<td>8338</td>
<td>4654</td>
</tr>
</tbody>
</table>

You can substitute the chart command for the stats command in this search. You can then click the Visualization tab to see a chart of the results.
3. Calculate a wide range of statistics by a specific field

Count the number of earthquakes that occurred for each magnitude range

This search uses recent earthquake data downloaded from the USGS Earthquakes website. The data is a comma separated ASCII text file that contains magnitude (mag), coordinates (latitude, longitude), region (place), etc., for each earthquake recorded.

You can download a current CSV file from the USGS Earthquake Feeds and upload the file to your Splunk instance. This example uses the All Earthquakes data from the past 30 days.

Run the following search to calculate the number of earthquakes that occurred in each magnitude range. This data set is comprised of events over a 30-day period.

```
source=all_month.csv | chart count AS "Number of Earthquakes" BY mag span=1 | rename mag AS "Magnitude Range"
```

- This search uses `span=1` to define each of the ranges for the magnitude field, `mag`.
- The `rename` command is then used to rename the field to "Magnitude Range".

The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>Magnitude Range</th>
<th>Number of Earthquakes</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1-0</td>
<td>18</td>
</tr>
<tr>
<td>0-1</td>
<td>2088</td>
</tr>
<tr>
<td>1-2</td>
<td>3005</td>
</tr>
<tr>
<td>2-3</td>
<td>1026</td>
</tr>
<tr>
<td>3-4</td>
<td>194</td>
</tr>
<tr>
<td>4-5</td>
<td>452</td>
</tr>
<tr>
<td>5-4</td>
<td>109</td>
</tr>
<tr>
<td>6-7</td>
<td>11</td>
</tr>
<tr>
<td>7-8</td>
<td>3</td>
</tr>
</tbody>
</table>

Click the Visualization tab to see the result in a chart.

Calculate aggregate statistics for the magnitudes of earthquakes in an area

Search for earthquakes in and around California. Calculate the number of earthquakes that were recorded. Use statistical functions to calculate the minimum, maximum, range (the difference between the min and max), and average magnitudes of the recent earthquakes. List the values by magnitude type.

```
source=all_month.csv place=*California* | stats count, max(mag), min(mag), range(mag), avg(mag) BY magType
```

The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>magType</th>
<th>count</th>
<th>max(mag)</th>
<th>min(mag)</th>
<th>range(mag)</th>
<th>avg(mag)</th>
</tr>
</thead>
</table>

8
### Find the mean, standard deviation, and variance of the magnitudes of the recent quakes

Search for earthquakes in and around California. Calculate the number of earthquakes that were recorded. Use statistical functions to calculate the mean, standard deviation, and variance of the magnitudes for recent earthquakes. List the values by magnitude type.

```
source=usgs place=*California* | stats count mean(mag), stdev(mag), var(mag) BY magType
```

The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>magType</th>
<th>count</th>
<th>mean(mag)</th>
<th>std(mag)</th>
<th>var(mag)</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>123</td>
<td>0.549593</td>
<td>0.356985</td>
<td>0.127438</td>
</tr>
<tr>
<td>MbLg</td>
<td>1</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
</tr>
<tr>
<td>Md</td>
<td>1565</td>
<td>1.056486</td>
<td>0.580042</td>
<td>0.336449</td>
</tr>
<tr>
<td>Me</td>
<td>2</td>
<td>1.800000</td>
<td>0.346410</td>
<td>0.120000</td>
</tr>
<tr>
<td>Ml</td>
<td>1202</td>
<td>1.226622</td>
<td>0.629664</td>
<td>0.396476</td>
</tr>
<tr>
<td>Mw</td>
<td>6</td>
<td>3.650000</td>
<td>0.716240</td>
<td>0.513000</td>
</tr>
<tr>
<td>ml</td>
<td>10</td>
<td>0.934000</td>
<td>0.560401</td>
<td>0.314049</td>
</tr>
</tbody>
</table>

The mean values should be exactly the same as the values calculated using `avg()`.

### 4. In a table display items sold by ID, type, and name and calculate the revenue for each product

This example uses the sample dataset from the Search Tutorial and a field lookup to add more information to the event data.

```
• Download the data set from Add data tutorial and follow the instructions to load the tutorial data.
• Download the CSV file from Use field lookups tutorial and follow the instructions to set up the lookup definition to add price and productName to the events.
```

After you configure the field lookup, you can run this search using the time range, All time.

Create a table that displays the items sold at the Buttercup Games online store by their ID, type, and name. Also, calculate the revenue for each product.

```
sourcetype=access_* status=200 action=purchase | stats values(categoryId) AS Type, values(productName) AS "Product Name", sum(price) AS "Revenue" by productId | rename productId AS "Product ID" | eval Revenue="$.tostring(Revenue,"commas")
```

9
This example uses the `values()` function to display the corresponding `categoryId` and `productName` values for each `productId`. Then, it uses the `sum()` function to calculate a running total of the values of the `price` field.

Also, this example renames the various fields, for better display. For the `stats` functions, the renames are done inline with an "AS" clause. The `rename` command is used to change the name of the `product_id` field, since the syntax does not let you rename a split-by field.

Finally, the results are piped into an `eval` expression to reformat the `Revenue` field values so that they read as currency, with a dollar sign and commas.

This returns the following table of results:

<table>
<thead>
<tr>
<th>Product ID</th>
<th>Type</th>
<th>Product Name</th>
<th>Revenue</th>
</tr>
</thead>
<tbody>
<tr>
<td>BS-AG-009</td>
<td>ARCADE</td>
<td>Byling Space Decks</td>
<td>$3,148.88</td>
</tr>
<tr>
<td>C1-PK-006</td>
<td>SPORTS</td>
<td>Cutting 204</td>
<td>$2,984.63</td>
</tr>
<tr>
<td>DB-9G-001</td>
<td>STRATEGY</td>
<td>Medieval Inquisitions</td>
<td>$2,947.82</td>
</tr>
<tr>
<td>DC-9G-002</td>
<td>STRATEGY</td>
<td>Dream Conquer</td>
<td>$8,373.94</td>
</tr>
<tr>
<td>FL-AG-008</td>
<td>ARCADE</td>
<td>Oval the Wave</td>
<td>$5,599.50</td>
</tr>
<tr>
<td>FS-9G-003</td>
<td>STRATEGY</td>
<td>Final Sequel</td>
<td>$4,918.00</td>
</tr>
<tr>
<td>MB-AG-007</td>
<td>ARCADE</td>
<td>Mongoviello Bros</td>
<td>$6,335.91</td>
</tr>
</tbody>
</table>

5. Determine how much email comes from each domain

This example uses sample email data. You should be able to run this search on any email data by replacing the `sourcetype=cisco:esa` with the `sourcetype` value and the `mailfrom` field with email address field name in your data. For example, the email might be To, From, or Cc.

Find out how much of the email in your organization comes from .com, .net, .org or other top level domains.

The `eval` command in this search contains two expressions, separated by a comma.

```evalcommand
sourcetype="cisco:esa" mailfrom=* | eval accountname=split(mailfrom,"@"), from_domain=mvindex(accountname,-1) | stats count(eval(match(from_domain, "[\^\n\r\s\+].com"))) AS ".com", count(eval(match(from_domain, "[\^\n\r\s\+].net"))) AS ".net", count(eval(match(from_domain, "[\^\n\r\s\+].org"))) AS ".org", count(eval(NOT match(from_domain, "[\^\n\r\s\+].(com|net|org)"))) AS "other"
```

- The first part of this search uses the `eval` command to break up the email address in the `mailfrom` field. The `from_domain` is defined as the portion of the `mailfrom` field after the `@` symbol.
  - The `split()` function is used to break the `mailfrom` field into a multivalue field called `accountname`. The first value of `accountname` is everything before the `@` symbol, and the second value is everything after.
  - The `mvindex()` function is used to set `from_domain` to the second value in the multivalue field `accountname`.
- The results are then piped into the `stats` command. The `count()` function is used to count the results of the `eval` expression.
  - The `eval` uses the `match()` function to compare the `from_domain` to a regular expression that looks for the different suffixes in the domain. If the value of `from_domain` matches the regular expression, the `count` is updated for each suffix, .com, .net, and .org. Other domain suffixes are counted as `other`.

The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>.com</th>
<th>.net</th>
<th>.org</th>
<th>other</th>
</tr>
</thead>
<tbody>
<tr>
<td>4246</td>
<td>9890</td>
<td>0</td>
<td>3543</td>
</tr>
</tbody>
</table>
6. Search Web access logs for the total number of hits from the top 10 referring domains

This example uses the sample data from the Search Tutorial but should work with any format of Apache web access log. To try this example on your own Splunk instance, you must download the sample data and follow the instructions to get the tutorial data into Splunk. Use the time range Yesterday when you run the search.

This example searches the web access logs and return the total number of hits from the top 10 referring domains.

```
sourcetype=access_* | top limit=10 referer
```

This search uses the `top` command to find the ten most common referer domains, which are values of the `referer` field. Some events might use `referer_domain` instead of `referer`. The `top` command returns a count and percent value for each referer.

<table>
<thead>
<tr>
<th>referer</th>
<th>count</th>
<th>percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>/<a href="http://www.uberogames.com">http://www.uberogames.com</a></td>
<td>209</td>
<td>4.707669</td>
</tr>
<tr>
<td><a href="http://www.google.com">http://www.google.com</a></td>
<td>163</td>
<td>3.710448</td>
</tr>
<tr>
<td>/<a href="http://www.uberogames.com/categories/screen/category-id-NULL">http://www.uberogames.com/categories/screen/category-id-NULL</a></td>
<td>152</td>
<td>3.459150</td>
</tr>
<tr>
<td>/<a href="http://www.uberogames.com/product/screen/productid/SF-BV5-S01">http://www.uberogames.com/product/screen/productid/SF-BV5-S01</a></td>
<td>127</td>
<td>2.849453</td>
</tr>
<tr>
<td>/<a href="http://www.uberogames.com/category/screen/category-id-ARCADE">http://www.uberogames.com/category/screen/category-id-ARCADE</a></td>
<td>122</td>
<td>2.777145</td>
</tr>
<tr>
<td>/<a href="http://www.uberogames.com/category/screen/categoryid/TEE">http://www.uberogames.com/category/screen/categoryid/TEE</a></td>
<td>93</td>
<td>2.07004</td>
</tr>
<tr>
<td>/<a href="http://www.uberogames.com/category/screen/categoryid/SHOOTER">http://www.uberogames.com/category/screen/categoryid/SHOOTER</a></td>
<td>81</td>
<td>1.843842</td>
</tr>
</tbody>
</table>

You can then use the `stats` command to calculate a total for the top 10 referrer accesses.

```
sourcetype=access_* | top limit=10 referer | stats sum(count) AS total
```

The `sum()` function adds the values in the `count` to produce the total number of times the top 10 referrers accessed the web site.

See also

Functions
  Statistical and charting functions

Commands
  eventstats
  rare
Blogs

Getting started with stats, eventstats and streamstats
Search commands > stats, chart, and timechart
Smooth operator | Searching for multiple field values

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the stats command.