```
# 📊 Advanced Span Analytics

> **Series:** SPANS | **Notebook:** 5 of 8 | **Created:** December 2025

## Time-Series Analysis and Complex Aggregations

This notebook covers advanced analytical techniques for span data, including
time-series analysis, trend detection, and complex aggregations for building
dashboards and reports.

---

## Table of Contents

1. Time-Series with makeTimeseries
2. Trend Analysis
3. Complex Aggregations
4. Comparison Queries
5. Dashboard-Ready Queries


## Prerequisites

Before starting this notebook, ensure you have:

- ✅ Completed previous SPANS notebooks (01-04)
- ✅ Understanding of summarize and aggregation functions
- ✅ Familiarity with time-based filtering

## 1. Time-Series with makeTimeseries

Use `makeTimeseries` to create time-series data for visualization and trend
analysis.

![Timeseries Analytics]
```

(data:image/svg+xml;base64,PHN2ZyB4bWxucz0iaHR0cDovL3d3dy53My5vcmcvMjAwMC9zdm
ciIHZpZXdCb3g9IjAgMCA3NTAgMzIwIj4KICA8ZGVmcz4KICAgIDxsaW5lYXJHcmFkaWVudCBpZD0
iY2hhcnRHcmFkIiB4MT0iMCUiIHkxPSIwJSIgeDI9IjAlIiB5Mj0iMTAwJSI+CiAgICAgIDxzdG9w
IG9mZnNldD0iMCUiIHN0eWxlPSJzdG9wLWNvbG9yOiMxNDk2ZmY7c3RvcC1vcGFjaXR5OjAuNCIgL
z4KICAgICAgPHN0b3Agb2Zmc2V0PSIxMDAlIiBzdHlsZT0ic3RvcC1jb2xvcjojMTQ5NmZmO3N0b3
Atb3BhY2l0eTowLjA1IiAvPgogICAgPC9saW5lYXJHcmFkaWVudD4KICAgIDxmaWx0ZXIgaWQ9ImN
oYXJ0U2hhZG93Ij4KICAgICAgPGZlRHJvcFNoYWRvdyBkeD0iMSIgZHk9IjEiIHN0ZERldmlhdGlv
bj0iMiIgZmxvb2Qtb3BhY2l0eT0iMC4xIi8+CiAgICA8L2ZpbHRlcj4KICA8L2RlZnM+CgogIDwhL
S0gQmFja2dyb3VuZCAtLT4KICA8cmVjdCB3aWR0aD0iNzUwIiBoZWlnaHQ9IjMyMCIgZmlsbD0iI2
Y4ZjlmYSIgcng9IjEwIi8+CgogIDwhLS0gVGl0bGUgLS0+CiAgPHRleHQgeD0iMzc1IiB5PSIyOCI
gZm9udC1mYW1pbHk9IkFyaWFsLCBzYW5zLXNlcmlmIiBmb250LXNpemU9IjE4IiBmb250LXdlaWdo
dD0iYm9sZCIgZmlsbD0iIzMzMyIgdGV4dC1hbmNob3I9Im1pZGRsZSI+VGltZS1TZXJpZXMgQW5hb

```
Hl0aWNzIHdpdGggbWFrZVRpbWVzZXJpZXM8L3RleHQ+CgogIDwhLS0gQ2hhcnQgQXJlYSAtLT4KIC
A8cmVjdCB4PSI2MCIgeT0iNTAiIHdpZHRoPSI0NTAiIGhlaWdodD0iMjAwIiByeD0iNiIgZmlsbD0
iI2ZmZiIgc3Ryb2tlPSIjZTJlOGYwIiBzdHJva2Utd2lkdGg9IjIiIGZpbHRlcj0idXJsKCNjaGFy
dFNoYWRvdykiLz4KCiAgPCEtLSBZLWF4aXMgbGFiZWxzIC0tPgogIDx0ZXh0IHg9IjUwIiB5PSI3M
CIgZm9udC1mYW1pbHk9IkFyaWFsLCBzYW5zLXNlcmlmIiBmb250LXNpemU9IjEwIiBmaWxsPSIjNj
Q3NDhiIiB0ZXh0LWFuY2hvcj0iZW5kIj41MDA8L3RleHQ+CiAgPHRleHQgeD0iNTAiIHk9IjEyMCI
gZm9udC1mYW1pbHk9IkFyaWFsLCBzYW5zLXNlcmlmIiBmb250LXNpemU9IjEwIiBmaWxsPSIjNjQ3
NDhiIiB0ZXh0LWFuY2hvcj0iZW5kIj4zMDA8L3RleHQ+CiAgPHRleHQgeD0iNTAiIHk9IjE3MCIgZ
m9udC1mYW1pbHk9IkFyaWFsLCBzYW5zLXNlcmlmIiBmb250LXNpemU9IjEwIiBmaWxsPSIjNjQ3ND
hiIiB0ZXh0LWFuY2hvcj0iZW5kIj4xMDA8L3RleHQ+CiAgPHRleHQgeD0iNTAiIHk9IjI0MCIgZm9
udC1mYW1pbHk9IkFyaWFsLCBzYW5zLXNlcmlmIiBmb250LXNpemU9IjEwIiBmaWxsPSIjNjQ3NDhi
IiB0ZXh0LWFuY2hvcj0iZW5kIj4wPC90ZXh0PgoKICA8IS0tIFktYXhpcyB0aXRsZSAtLT4KICA8d
GV4dCB4PSIyNSIgeT0iMTUwIiBmb250LWZhbWlseT0iQXJpYWwsIHNhbnMtc2VyaWYiIGZvbnQtc2
l6ZT0iMTEiIGZpbGw9IiM2NDc0OGIiIHRleHQtYW5jaG9yPSJtaWRkbGUiIHRyYW5zZm9ybT0icm9
0YXRlKC05MCwgMjUsIDE1MCkiPlJlcXVlc3RzL21pbjwvdGV4dD4KCiAgPCEtLSBHcmlkIGxpbmVz
IC0tPgogIDxsaW5lIHgxPSI2MCIgeTE9IjcwIiB4Mj0iNTEwIiB5Mj0iNzAiIHN0cm9rZT0iI2YxZ
jVmOSIgc3Ryb2tlLXdpZHRoPSIxIi8+CiAgPGxpbmUgeDE9IjYwIiB5MT0iMTIwIiB4Mj0iNTEwIi
B5Mj0iMTIwIiBzdHJva2U9IiNmMWY1ZjkiIHN0cm9rZS13aWR0aD0iMSIvPgogIDxsaW5lIHgxPSI
2MCIgeTE9IjE3MCIgeDI9IjUxMCIgeTI9IjE3MCIgc3Ryb2tlPSIjZjFmNWY5IiBzdHJva2Utd2lk
dGg9IjEiLz4KCiAgPCEtLSBBcmVhIGZpbGwgLS0+CiAgPHBhdGggZD0iTTcwLDIwMCBMMTIwLDE4M
CBMMTcwLDE1MCBMMjIwLDEyMCBMMjcwLDEwMCBMMzIwLDg1IEwzNzAsOTAgTDQyMCwxMTAgTDQ3MC
wxMzAgTDUwMCwxNDAgTDUwMCwyNDAgTDcwLDI0MCBaIiBmaWxsPSJ1cmwoI2NoYXJ0R3JhZCkiLz4
KCiAgPCEtLSBMaW5lIGNoYXJ0IC0tPgogIDxwb2x5bGluZSBwb2ludHM9IjcwLDIwMCAxMjAsMTgw
IDE3MCwxNTAgMjIwLDEyMCAyNzAsMTAwIDMyMCw4NSAzNzAsOTAgNDIwLDExMCA0NzAsMTMwIDUwM
CwxNDAiCiAgICAgICAgICAgIGZpbGw9Im5vbmUiIHN0cm9rZT0iIzE0OTZmZiIgc3Ryb2tlLXdpZH
RoPSIzIiBzdHJva2UtbGluZWpvaW49InJvdW5kIi8+CgogIDwhLS0gRGF0YSBwb2ludHMgLS0+CiA
gPGNpcmNsZSBjeD0iNzAiIGN5PSIyMDAiIHI9IjUiIGZpbGw9IiMxNDk2ZmYiLz4KICA8Y2lyY2xl
IGN4PSIxMjAiIGN5PSIxODAiIHI9IjUiIGZpbGw9IiMxNDk2ZmYiLz4KICA8Y2lyY2xlIGN4PSIxN
zAiIGN5PSIxNTAiIHI9IjUiIGZpbGw9IiMxNDk2ZmYiLz4KICA8Y2lyY2xlIGN4PSIyMjAiIGN5PS
IxMjAiIHI9IjUiIGZpbGw9IiMxNDk2ZmYiLz4KICA8Y2lyY2xlIGN4PSIyNzAiIGN5PSIxMDAiIHI
9IjUiIGZpbGw9IiMxNDk2ZmYiLz4KICA8Y2lyY2xlIGN4PSIzMjAiIGN5PSI4NSIgcj0iNSIgZmls
bD0iI2VmNDQ0NCIvPgogIDxjaXJjbGUgY3g9IjM3MCIgY3k9IjkwIiByPSI1IiBmaWxsPSIjMTQ5N
mZmIi8+CiAgPGNpcmNsZSBjeD0iNDIwIiBjeT0iMTEwIiByPSI1IiBmaWxsPSIjMTQ5NmZmIi8+Ci
AgPGNpcmNsZSBjeD0iNDcwIiBjeT0iMTMwIiByPSI1IiBmaWxsPSIjMTQ5NmZmIi8+CiAgPGNpcmN
sZSBjeD0iNTAwIiBjeT0iMTQwIiByPSI1IiBmaWxsPSIjMTQ5NmZmIi8+CgogIDwhLS0gUGVhayBh
bm5vdGF0aW9uIC0tPgogIDxsaW5lIHgxPSIzMjAiIHkxPSI4NSIgeDI9IjMyMCIgeTI9IjU1IiBzd
HJva2U9IiNlZjQ0NDQiIHN0cm9rZS13aWR0aD0iMSIgc3Ryb2tlLWRhc2hhcnJheT0iMywyIi8+Ci
AgPHRleHQgeD0iMzIwIiB5PSI1MCIgZm9udC1mYW1pbHk9IkFyaWFsLCBzYW5zLXNlcmlmIiBmb25
0LXNpemU9IjEwIiBmaWxsPSIjZWY0NDQ0IiB0ZXh0LWFuY2hvcj0ibWlkZGxlIj5QZWFrOiA0NTAv
bWluPC90ZXh0PgoKICA8IS0tIFgtYXhpcyBsYWJlbHMgLS0+CiAgPHRleHQgeD0iNzAiIHk9IjI2M
CIgZm9udC1mYW1pbHk9IkFyaWFsLCBzYW5zLXNlcmlmIiBmb250LXNpemU9IjEwIiBmaWxsPSIjNj
Q3NDhiIiB0ZXh0LWFuY2hvcj0ibWlkZGxlIj4wMDowMDwvdGV4dD4KICA8dGV4dCB4PSIxNzAiIHk
9IjI2MCIgZm9udC1mYW1pbHk9IkFyaWFsLCBzYW5zLXNlcmlmIiBmb250LXNpemU9IjEwIiBmaWxs
PSIjNjQ3NDhiIiB0ZXh0LWFuY2hvcj0ibWlkZGxlIj4wNDowMDwvdGV4dD4KICA8dGV4dCB4PSIyN
zAiIHk9IjI2MCIgZm9udC1mYW1pbHk9IkFyaWFsLCBzYW5zLXNlcmlmIiBmb250LXNpemU9IjEwIi
BmaWxsPSIjNjQ3NDhiIiB0ZXh0LWFuY2hvcj0ibWlkZGxlIj4wODowMDwvdGV4dD4KICA8dGV4dCB
4PSIzNzAiIHk9IjI2MCIgZm9udC1mYW1pbHk9IkFyaWFsLCBzYW5zLXNlcmlmIiBmb250LXNpemU9
IjEwIiBmaWxsPSIjNjQ3NDhiIiB0ZXh0LWFuY2hvcj0ibWlkZGxlIj4xMjowMDwvdGV4dD4KICA8d
```

```
GV4dCB4PSI0NzAiIHk9IjI2MCIgZm9udC1mYW1pbHk9IkFyaWFsLCBzYW5zLXNlcmlmIiBmb250LX
NpemU9IjEwIiBmaWxsPSIjNjQ3NDhiIiB0ZXh0LWFuY2hvcj0ibWlkZGxlIj4xNjowMDwvdGV4dD4
KCiAgPCEtLSBEUUwgUXVlcnkgYm94IC0tPgogIDxyZWN0IHg9IjUzMCIgeT0iNTAiIHdpZHRoPSIy
MDAiIGhlaWdodD0iMjAwIiByeD0iNiIgZmlsbD0iIzFlMjkzYiIvPgogIDx0ZXh0IHg9IjU0NSIge
T0iNzUiIGZvbnQtZmFtaWx5PSJBcmlhbCwgc2Fucy1zZXJpZiIgZm9udC1zaXplPSIxMSIgZm9udC
13ZWlnaHQ9ImJvbGQiIGZpbGw9IiM5NGEzYjgiPm1ha2VUaW1lc2VyaWVzIFF1ZXJ5PC90ZXh0Pgo
KICA8dGV4dCB4PSI1NDUiIHk9IjEwMCIgZm9udC1mYW1pbHk9Im1vbm9zcGFjZSIgZm9udC1zaXpl
PSIxMCIgZmlsbD0iIzIyYzU1ZSI+ZmV0Y2g8L3RleHQ+CiAgPHRleHQgeD0iNTgwIiB5PSIxMDAiI
GZvbnQtZmFtaWx5PSJtb25vc3BhY2UiIGZvbnQtc2l6ZT0iMTAiIGZpbGw9IiNmOGZhZmMiPnNwYW
5zPC90ZXh0PgoKICA8dGV4dCB4PSI1NDUiIHk9IjExOCIgZm9udC1mYW1pbHk9Im1vbm9zcGFjZSI
gZm9udC1zaXplPSIxMCIgZmlsbD0iIzk0YTNiOCI+fDwvdGV4dD4KICA8dGV4dCB4PSI1NTUiIHk9
IjExOCIgZm9udC1mYW1pbHk9Im1vbm9zcGFjZSIgZm9udC1zaXplPSIxMCIgZmlsbD0iI2Y5NzMxN
iI+ZmlsdGVyPC90ZXh0PgogIDx0ZXh0IHg9IjYwMCIgeT0iMTE4IiBmb250LWZhbWlseT0ibW9ub3
NwYWNlIiBmb250LXNpemU9IjEwIiBmaWxsPSIjZjhmYWZjIj5zcGFuLmtpbmQgPT08L3RleHQ+CiA
gPHRleHQgeD0iNTU1IiB5PSIxMzMiIGZvbnQtZmFtaWx5PSJtb25vc3BhY2UiIGZvbnQtc2l6ZT0i
MTAiIGZpbGw9IiNmYmJmMjQiPiJzZXJ2ZXIiPC90ZXh0PgoKICA8dGV4dCB4PSI1NDUiIHk9IjE1N
SIgZm9udC1mYW1pbHk9Im1vbm9zcGFjZSIgZm9udC1zaXplPSIxMCIgZmlsbD0iIzk0YTNiOCI+fD
wvdGV4dD4KICA8dGV4dCB4PSI1NTUiIHk9IjE1NSIgZm9udC1mYW1pbHk9Im1vbm9zcGFjZSIgZm9
udC1zaXplPSIxMCIgZmlsbD0iIzE0OTZmZiI+bWFrZVRpbWVzZXJpZXM8L3RleHQ+CgogIDx0ZXh0
IHg9IjU2MCIgeT0iMTczIiBmb250LWZhbWlseT0ibW9ub3NwYWNlIiBmb250LXNpemU9IjEwIiBma
WxsPSIjZjhmYWZjIj5yZXF1ZXN0cyA9IGNvdW50KCksPC90ZXh0PgogIDx0ZXh0IHg9IjU2MCIgeT
0iMTg4IiBmb250LWZhbWlseT0ibW9ub3NwYWNlIiBmb250LXNpemU9IjEwIiBmaWxsPSIjMTQ5NmZ
mIj5pbnRlcnZhbDo8L3RleHQ+CiAgPHRleHQgeD0iNjE1IiB5PSIxODgiIGZvbnQtZmFtaWx5PSJt
b25vc3BhY2UiIGZvbnQtc2l6ZT0iMTAiIGZpbGw9IiM2MGE1ZmEiPjVtPC90ZXh0PgoKICA8dGV4d
CB4PSI1NDUiIHk9IjIxMCIgZm9udC1mYW1pbHk9Im1vbm9zcGFjZSIgZm9udC1zaXplPSIxMCIgZm
lsbD0iIzk0YTNiOCI+Ly8gQWxzbyBzdXBwb3J0czo8L3RleHQ+CiAgPHRleHQgeD0iNTQ1IiB5PSI
yMjUiIGZvbnQtZmFtaWx5PSJtb25vc3BhY2UiIGZvbnQtc2l6ZT0iMTAiIGZpbGw9IiM5NGEzYjgi
Pi8vIGF2ZygpLCBzdW0oKSw8L3RleHQ+CiAgPHRleHQgeD0iNTQ1IiB5PSIyNDAiIGZvbnQtZmFta
Wx5PSJtb25vc3BhY2UiIGZvbnQtc2l6ZT0iMTAiIGZpbGw9IiM5NGEzYjgiPi8vIHBlcmNlbnRpbG
UoKTwvdGV4dD4KCiAgPCEtLSBMZWdlbmQgLS0+CiAgPHJlY3QgeD0iNjAiIHk9IjI3NSIgd2lkdGg
9IjY3MCIgaGVpZ2h0PSIzNSIgcng9IjYiIGZpbGw9IiNmMGY5ZmYiIHN0cm9rZT0iI2JhZTZmZCIg
c3Ryb2tlLXdpZHRoPSIxIi8+CiAgPGNpcmNsZSBjeD0iODAiIGN5PSIyOTIiIHI9IjYiIGZpbGw9I
iMxNDk2ZmYiLz4KICA8dGV4dCB4PSI5NSIgeT0iMjk3IiBmb250LWZhbWlseT0iQXJpYWwsIHNhbn
Mtc2VyaWYiIGZvbnQtc2l6ZT0iMTEiIGZpbGw9IiMwMzY5YTEiPlJlcXVlc3QgY291bnQgcGVyIGl
udGVydmFsPC90ZXh0PgogIDxjaXJjbGUgY3g9IjI4MCIgY3k9IjI5MiIgcj0iNiIgZmlsbD0iI2Vm
NDQ0NCIvPgogIDx0ZXh0IHg9IjI5NSIgeT0iMjk3IiBmb250LWZhbWlseT0iQXJpYWwsIHNhbnMtc
2VyaWYiIGZvbnQtc2l6ZT0iMTEiIGZpbGw9IiNiOTFjMWMiPkFub21hbHkvUGVhayBkZXRlY3RlZD
wvdGV4dD4KICA8dGV4dCB4PSI1MDAiIHk9IjI5NyIgZm9udC1mYW1pbHk9IkFyaWFsLCBzYW5zLXN
lcmlmIiBmb250LXNpemU9IjExIiBmaWxsPSIjMDM2OWExIj5QZXJmZWN0IGZvciBkYXNoYm9hcmRz
ICZhbXA7IGFsZXJ0aW5nPC90ZXh0Pgo8L3N2Zz4K)
```

~~~
> ⚠️ **Important:** `makeTimeseries` does NOT support `percentile()` or
arithmetic expressions in aggregations. Use time-bucketed `summarize` for
those.

```dql
~~~

~~~
// Request volume over time by service
fetch spans
| filter span.kind == "server"
| makeTimeseries {
    request_count = count(),
    error_count = countIf(span.status_code == "error")
  }, by:{service.name}, interval: 5m
```

```dql
// Average duration over time (without grouping)
fetch spans
| filter span.kind == "server"
| makeTimeseries {
    request_count = count(),
    avg_duration_ns = avg(duration)
  }, interval: 5m
```

```dql
// For percentile trends, use time-bucketed summarize instead
fetch spans
| filter span.kind == "server"
| fieldsAdd time_bucket = bin(start_time, 10m)
| summarize {
    request_count = count(),
    p95_duration_ms = percentile(duration, 95) / 1000000,
    error_count = countIf(span.status_code == "error")
  }, by:{time_bucket, service.name}
| sort time_bucket asc
| limit 200
```

---

## 2. Trend Analysis

Identify trends and patterns in your span data over time.

```dql
// Hourly error rate trend
fetch spans
| filter span.kind == "server"
| fieldsAdd hour_bucket = bin(start_time, 1h)
| summarize {
    total_requests = count(),
    errors = countIf(span.status_code == "error")
  }, by:{hour_bucket}
~~~

~~~
| fieldsAdd error_rate_pct = (errors * 100.0) / total_requests
| sort hour_bucket asc
| limit 48
```

```dql
// Latency trend by service (10-minute buckets)
fetch spans
| filter span.kind == "server"
| fieldsAdd time_bucket = bin(start_time, 10m)
| summarize {
    request_count = count(),
    avg_duration_ms = avg(duration) / 1000000,
    p90_duration_ms = percentile(duration, 90) / 1000000
  }, by:{time_bucket, service.name}
| sort time_bucket asc, service.name
| limit 300
```

```dql
// Identify services with degrading performance
fetch spans
| filter span.kind == "server"
| fieldsAdd time_bucket = bin(start_time, 30m)
| summarize {
    request_count = count(),
    p95_ms = percentile(duration, 95) / 1000000
  }, by:{time_bucket, service.name}
| filter p95_ms > 500
| sort time_bucket desc, p95_ms desc
| limit 50
```

---

## 3. Complex Aggregations

Perform multi-dimensional analysis with complex aggregation patterns.

```dql
// Service health scorecard
fetch spans
| filter span.kind == "server"
| summarize {
    total_requests = count(),
    error_count = countIf(span.status_code == "error"),
    avg_duration_ms = avg(duration) / 1000000,
    p50_duration_ms = percentile(duration, 50) / 1000000,
~~~

~~~
    p95_duration_ms = percentile(duration, 95) / 1000000,
    p99_duration_ms = percentile(duration, 99) / 1000000,
    max_duration_ms = max(duration) / 1000000
  }, by:{service.name}
| fieldsAdd error_rate_pct = (error_count * 100.0) / total_requests
| fieldsAdd health_score = if(error_rate_pct > 5, "Critical",
                            else: if(error_rate_pct > 1, "Warning",
                            else: "Healthy"))
| sort error_rate_pct desc
| limit 30
```

```dql
// Endpoint-level analysis with multiple metrics
fetch spans
| filter span.kind == "server"
| summarize {
    request_count = count(),
    error_count = countIf(span.status_code == "error"),
    slow_count = countIf(duration > 1000000000),  // > 1 second
    avg_duration_ms = avg(duration) / 1000000,
    p95_duration_ms = percentile(duration, 95) / 1000000
  }, by:{service.name, span.name}
| fieldsAdd error_rate_pct = (error_count * 100.0) / request_count
| fieldsAdd slow_rate_pct = (slow_count * 100.0) / request_count
| filter request_count > 10
| sort error_rate_pct desc
| limit 50
```

```dql
// HTTP method distribution with performance metrics
fetch spans
| filter isNotNull(http.request.method)
| summarize {
    request_count = count(),
    error_count = countIf(span.status_code == "error"),
    avg_duration_ms = avg(duration) / 1000000
  }, by:{http.request.method, service.name}
| sort request_count desc
| limit 30
```

---

## 4. Comparison Queries

Compare metrics across different dimensions to identify outliers and
~~~

~~~
patterns.

```dql
// Compare span kinds: server vs client performance
fetch spans
| filter in(span.kind, {"server", "client"})
| summarize {
    span_count = count(),
    error_count = countIf(span.status_code == "error"),
    avg_duration_ms = avg(duration) / 1000000,
    p95_duration_ms = percentile(duration, 95) / 1000000
  }, by:{span.kind}
| fieldsAdd error_rate_pct = (error_count * 100.0) / span_count
```

```dql
// Compare success vs error span characteristics
fetch spans
| filter span.kind == "server"
| summarize {
    span_count = count(),
    avg_duration_ms = avg(duration) / 1000000,
    p95_duration_ms = percentile(duration, 95) / 1000000
  }, by:{span.status_code, service.name}
| sort service.name, span.status_code
| limit 50
```

```dql
// HTTP status code distribution by service
fetch spans
| filter isNotNull(http.response.status_code)
| fieldsAdd status_class = if(http.response.status_code >= 500, "5xx",
                            else: if(http.response.status_code >= 400, "4xx",
                            else: if(http.response.status_code >= 300, "3xx",
                            else: "2xx")))
| summarize {count = count()}, by:{status_class, service.name}
| sort service.name, status_class
```

---

## 5. Dashboard-Ready Queries

Queries optimized for use in Dynatrace dashboards and reports.

> 💡 **Tip:** These queries are designed to produce clean output suitable for
dashboard tiles.
~~~

~~~
```dql
// Dashboard: Request rate and error rate over time
fetch spans
| filter span.kind == "server"
| makeTimeseries {
    requests = count(),
    errors = countIf(span.status_code == "error")
  }, interval: 5m
```

```dql
// Dashboard: Service health summary (single value tiles)
fetch spans
| filter span.kind == "server"
| summarize {
    total_requests = count(),
    total_errors = countIf(span.status_code == "error"),
    unique_services = countDistinct(service.name),
    avg_latency_ms = avg(duration) / 1000000
  }
| fieldsAdd overall_error_rate_pct = (total_errors * 100.0) / total_requests
```

```dql
// Dashboard: Top 10 services by request volume
fetch spans
| filter span.kind == "server"
| summarize {
    requests = count(),
    errors = countIf(span.status_code == "error"),
    p95_ms = percentile(duration, 95) / 1000000
  }, by:{service.name}
| fieldsAdd error_rate = (errors * 100.0) / requests
| sort requests desc
| limit 10
```

```dql
// Dashboard: Recent errors list
fetch spans
| filter span.status_code == "error"
| fields start_time, service.name, span.name, span.status_message
| sort start_time desc
| limit 20
```

---
~~~

```
## Summary

In this notebook, you learned:

✅ **Time-series analysis** with makeTimeseries and its limitations
✅ **Time-bucketed summarize** for percentile trends
✅ **Trend analysis** to identify patterns over time
✅ **Complex aggregations** for multi-dimensional analysis
✅ **Comparison queries** to identify outliers
✅ **Dashboard-ready queries** for visualization

---

## Next Steps

Continue to **SPANS-06: Security Analysis with Spans** to learn:
- Detecting security-relevant patterns in traces
- Analyzing authentication and authorization flows
- Finding anomalous behavior
- Security audit queries
```