

DESK READY · LIVE BUILD

Building a rates trading dashboard, from an empty file

Python, Dash and AG Grid. Every decision explained as we go.

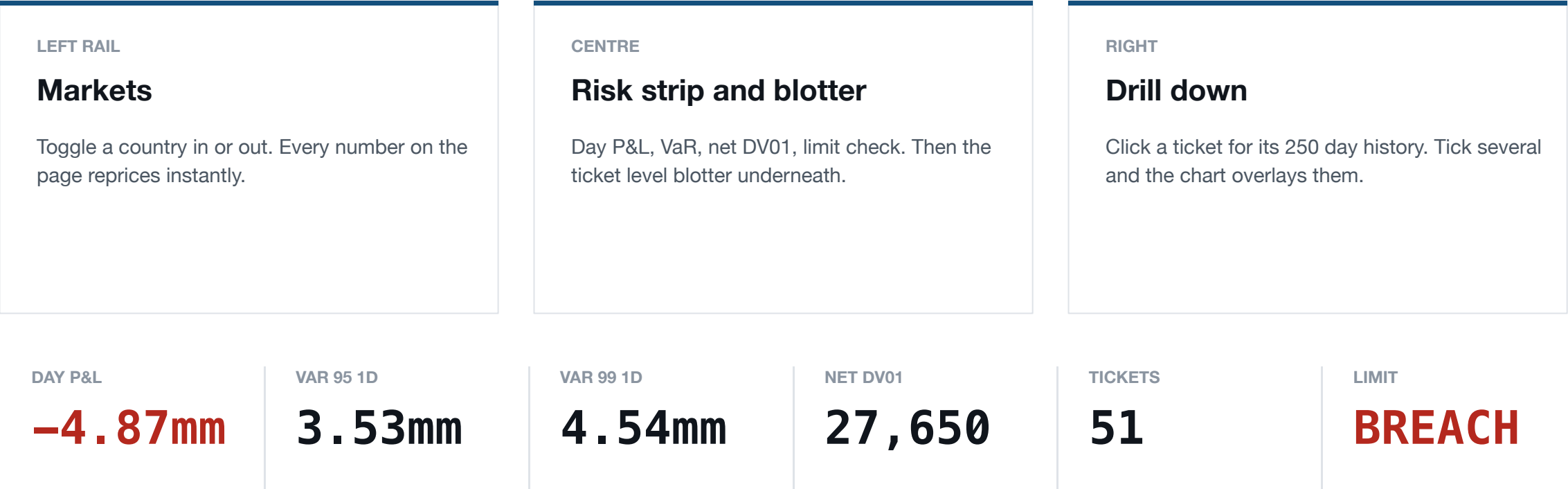
Sunday 20 September, 4:30pm
staydeskreedy.com



THE END STATE

What we are building

A three column risk screen, the way a real rates desk lays one out.



The book loses more than its 95% VaR on the day. That is a breach, and it is the first thing we talk about.



The stack, and why each piece is there

pandas

Every dataset is a DataFrame. Filtering, grouping, totals.

Dash

Turns Python into a web app. You write no JavaScript.

Dash AG Grid

The blotter. Sorting, filtering, grouping, editable cells.

Plotly

The charts, which Dash renders natively.

NumPy

The random walks and the percentile maths behind VaR.

TEN MINUTES, THEN NEVER AGAIN

Setup, once

A virtual environment keeps this project's packages separate from everything else on your machine.

terminal

```
# make a folder and step into it
mkdir rates-dashboard && cd rates-dashboard

# create the environment
python3 -m venv .venv

# switch it on (Mac and Linux)
source .venv/bin/activate

# Windows instead:
.venv\Scripts\activate
```

terminal

```
# install what we need
pip install dash dash-ag-grid \
    pandas numpy plotly

# build the sample data
python generate_data.py

# run the dashboard
python app.py

# then open http://127.0.0.1:8050
```

In VS Code, press Cmd+Shift+P, choose Python: Select Interpreter, and pick the one inside .venv.
Get that wrong and VS Code underlines every import in red while the code still runs fine.



FOUR FILES, ONE SCRIPT

The data model

book.csv	The live book. One row per ticket: side, notional, entry, last, DV01, P&L.
history.csv	250 business days of yields for every instrument. This is what the drill down charts read.
var_history.csv	250 days of 1 day VaR at 95% and 99%, per desk.
pnl_intraday.csv	Today's P&L path in 15 minute buckets.

One seeded random number generator, so every screen in the room shows identical numbers.

Desk Ready - Rates Dashboard Session

DV01, and the one minus sign that matters

DV01 is what you make or lose on a 1 basis point move. It scales with tenor, which is why a 30 year bond carries roughly ten times the risk of a 2 year for the same notional.

generate_data.py

```
dv01 = dv01_per_million * notional_mm

# Long a bond and yields FALL, the price rises
# and you make money. So a negative yield move
# is a positive P&L for a long. Hence the minus.
direction = 1 if side == 'Buy' else -1
pnl = -direction * move_bp * dv01
```

WORKED EXAMPLE

Long 50mm UST 10Y
DV01 860 per million
Position DV01 43,000

Yields fall 2bp
P&L = +86,000

If you take one idea from this session, take that one. Everything else on the screen is bookkeeping around it.

VaR, in three steps

Historical simulation, which is the method most desks actually report. No normal distribution assumed anywhere.

- 01 Take the daily yield changes from the last 250 days.
- 02 Apply today's DV01 to each of those days. That gives the P&L you would have had if that day repeated.
- 03 Sort that distribution. Read off the 5th percentile for 95% VaR, the 1st for 99%.

```
changes_bp = yields.diff().dropna() * 100          # daily moves, in bp
daily_pnl   = -(window[names] * dv01[names]).sum(axis=1) # P&L if that day repeated
var95       = -np.percentile(daily_pnl, 5)          # the 5th percentile loss
```

Dash, in one idea

A layout describes the page. A callback says: when these inputs change, recompute these outputs. Dash watches the browser, posts the values back to Python, runs your function, patches the page.

app.py

```
@app.callback(
    Output('blotter', 'rowData'),      # what gets updated
    Output('s-pnl', 'children'),
    Input('active-countries', 'data'),  # what triggers it
    Input('status-filter', 'value'),
)
def refresh(countries, statuses):
    df = book[book['Country'].isin(countries) & book['Status'].isin(statuses)]
    return df.to_dict('records'), money(df['PnL'].sum())
```

The return tuple must match the Output order exactly. That is the single most common thing to get wrong.

AG Grid: one dict per column

These are AG Grid options rather than Dash ones, which is why they are camelCase.

```
{
  'field': 'PnL',
  'headerName': 'P&L',
  'type': 'numericColumn',
  'aggFunc': 'sum',
  'valueFormatter': NUM,
  'cellStyle': {'styleConditions': [
    {'condition': 'params.value > 0',
     'style': {'color': '#0a7a48'}},
    {'condition': 'params.value < 0',
     'style': {'color': '#b4281e'}},
  ]},
}
```

field

which DataFrame column to show

type

numericColumn right aligns it

aggFunc

how to total it when grouped

valueFormatter

changes what is displayed, never the underlying value, so sorting still runs on the real number

cellStyle

conditional formatting. This is how P&L goes red and green

One callback, six buttons

The country rail could have been six near identical callbacks. Pattern matching ids collapse it into one. Give a component a dict id instead of a string, then listen to all of them at once with ALL.

app.py

```
html.Div(id={'type': 'ctry', 'index': 'Japan'}, n_clicks=0)    # in the layout

@app.callback(
    Output('active-countries', 'data'),
    Input({'type': 'ctry', 'index': ALL}, 'n_clicks'),          # every ctry at once
    State('active-countries', 'data'),                          # read without firing
)
def toggle_country(_clicks, active):
    country = ctx.triggered_id['index']    # which one was actually clicked
    ...
```

Note the guard in the real code: never let the user switch every market off. An empty screen mid session looks like a crash.

Comparing markets properly

Tick more than one row and the yield chart overlays them, rebased to zero at the start of the window.

Comparing raw yield levels across markets tells you nothing. A JGB at 1.6% and a Gilt at 4.6% sit far apart on the page even on a day they moved identically. Rebasing puts both in basis points moved.

app.py

```
names = list(dict.fromkeys(r['Instrument'] for r in selected))
if len(names) > 1:
    for nm in names[:5]:
        h = history[history['Instrument'] == nm].sort_values('Date')
        rebased = (h['Yield'] - h['Yield'].iloc[0]) * 100    # bp from the start
        cfig.add_trace(go.Scatter(x=h['Date'], y=rebased, name=nm))
```

Worth knowing: `cellClicked` looks like the obvious input for a drill down, but it only reports the value, the column and the row index. `selectedRows` hands back the whole record, which is what you actually need.

Make a cell editable, reprice the book

Set editable on the Last column. Type a new yield and the move, the P&L and the colour all follow.

app.py

```
@app.callback(
    Output('blotter', 'rowData', allow_duplicate=True),
    Input('blotter', 'cellValueChanged'),
    State('blotter', 'rowData'),
    prevent_initial_call=True,
)
def reprice(changed, rows):
    move = (float(r['Last']) - float(r['Entry'])) * 100
    r['MoveBp'] = round(move, 2)
    # DV01 already carries the sign of the position, so no direction term.
    r['PnL'] = round(-move * float(r['DV01']), 0)
```

allow_duplicate is needed because two callbacks now write to rowData. prevent_initial_call is required alongside it.

Three things that will cost you an hour

The grid renders bare

AG Grid v33 and up ship a new theming system. If you style with the classic CSS theme you must pass theme: 'legacy' in dashGridOptions, and load the stylesheets yourself.

Enterprise features look broken

Set filters, row grouping and totals need enableEnterpriseModules=True. Without a licence key they still run in full, you just get a watermark and a console notice. Nothing is restricted.

NaN across the totals row

The grand total row has no value in the columns you did not sum, so every valueFormatter has to cope with null or d3 prints NaN across the footer.

Run it yourself

terminal

```
cd live-session-dashboard
source ../.venv/bin/activate
python generate_data.py
python app.py
```

THEN OPEN

<http://127.0.0.1:8050>

EXTEND IT

- Add a second risk measure, say CS01
- Swap the sample data for real yields
- Add a curve steepener trade blotter
- Put a scenario shock on the DV01 ladder

FOR YOUR CV

Built an interactive rates risk dashboard in Python (Dash, AG Grid) covering P&L attribution, DV01 laddering and historical simulation VaR across six sovereign markets.

Questions,

Change a yield. Switch a market off. Tick four rows and compare them.

staydeskready.com