Sales KPI Dashboard Demo

In this document we'll work through a basic example of coding a sales KPI Exercise document from scratch. We'll then work through the use of AI to accomplish this task with increasingly advanced methods.

Section 1: Basic Sales KPIs

Build a simple Sales KPI dashboard from small CSV files and export an Excel with charts and a summary page.

Steps:

- Load, clean, and type-check CSV data with pandas.
- Compute core KPIs with groupby, pivot tables, and date logic.
- Create basic charts with matplotlib and save them to /output.
- Export clean tables and charts into a single Excel file using xlsxwriter.

Output

- An Excel workbook: output/dashboard.xlsx with three sheets (Summary, Weekly_Trend, Top_Products).
- Two PNG charts saved to /output and also embedded into the Summary sheet.

Dataset

- data/orders.csv — order_id, order_date (YYYY-MM-DD), region, sales_rep, customer_id, product_id, units, unit_price, revenue
- data/products.csv — product_id, product_name, category
- data/targets.csv — monthly revenue_target per region: year, month, region, revenue_target

Load and basic checksdata

Create main.py. All the code from part 1 will be pasted into the same main script. First to implement: read the CSVs, parse dates, set types, and add a few basic assertions so silent errors don’t sneak in.

```
# main.py
from pathlib import Path
import pandas as pd

ROOT = Path(__file__).parent
DATA = ROOT / 'data'
OUT = ROOT / 'output'
OUT.mkdir(exist_ok=True)

DTYPES_ORDERS = {
    'order_id': 'int64', 'region': 'category', 'sales_rep': 'category',
    'customer_id': 'int64', 'product_id': 'int64', 'units': 'int64'
}

def read_csv_safe(path: Path, parse_dates=None, dtypes=None) ->
pd.DataFrame:
    df = pd.read_csv(path)
    if parse_dates:
        for col in parse_dates:
            df[col] = pd.to_datetime(df[col], errors='coerce')
    if dtypes:
        for col, dt in dtypes.items():
            if col in df.columns:
                df[col] = df[col].astype(dt)
    return df

orders = read_csv_safe(DATA / 'orders.csv', parse_dates=['order_date'],
dtypes=DTYPES_ORDERS)
products = read_csv_safe(DATA / 'products.csv')
targets  = read_csv_safe(DATA / 'targets.csv', dtypes={'year': 'int64',
'month': 'int64'})

assert orders['order_id'].is_unique
assert (orders['revenue'] >= 0).all()
```

KPI calculations

We will compute three core views. Weekly revenue by region, top 5 products in the current month, and monthly revenue vs target by region.

```
# Weekly revenue by region
orders['week'] = orders['order_date'].dt.to_period('W').apply(lambda p:
p.start_time.date())
weekly = (orders.groupby(['week', 'region'],
as_index=False)['revenue'].sum()
                .sort_values(['week', 'region']))
weekly_pivot = weekly.pivot(index='week', columns='region',
values='revenue').fillna(0)

# Top 5 products this month
latest_month = orders['order_date'].dt.to_period('M').max()
cur = orders[orders['order_date'].dt.to_period('M') ==
latest_month].merge(products, on='product_id', how='left')
prod_top5 = (cur.groupby(['product_id', 'product_name'],
as_index=False)['revenue'].sum()
               .sort_values('revenue', ascending=False).head(5))

# Monthly revenue vs target
orders['year'] = orders['order_date'].dt.year
```

```
orders['month'] = orders['order_date'].dt.month
monthly = orders.groupby(['year','month','region'],
as_index=False)['revenue'].sum()
rev_vs_target = monthly.merge(targets, on=['year','month','region'],
how='left')
rev_vs_target['variance'] = rev_vs_target['revenue'] -
rev_vs_target['revenue_target']
rev_vs_target['pct_to_target'] = (rev_vs_target['revenue'] /
rev_vs_target['revenue_target']).round(3)
```

Visualise

Output the plots as images.

```
import matplotlib.pyplot as plt

ax = weekly_pivot.plot(figsize=(9,5))
ax.set_title('Weekly Revenue by Region'); ax.set_xlabel('Week');
ax.set_ylabel('Revenue')
plt.tight_layout()
(OUT / 'weekly_by_region.png').parent.mkdir(exist_ok=True)
plt.savefig(OUT / 'weekly_by_region.png'); plt.close()

ax = prod_top5.plot(x='product_name', y='revenue', kind='bar', legend=False,
figsize=(8,5))
ax.set_title(f'Top 5 Products — {str(latest_month)}')
ax.set_xlabel('Product'); ax.set_ylabel('Revenue')
plt.tight_layout()
plt.savefig(OUT / 'top5_products.png'); plt.close()
```

Export to Excel

```
with pd.ExcelWriter(OUT / 'dashboard.xlsx', engine='xlsxwriter') as xw:
    rev_vs_target.to_excel(xw, sheet_name='Summary', index=False)
    weekly_pivot.reset_index().to_excel(xw, sheet_name='Weekly_Trend',
index=False)
    prod_top5.to_excel(xw, sheet_name='Top_Products', index=False)

    wb  = xw.book
    wsS = xw.sheets['Summary']
    money = wb.add_format({'num_format': '#,##0'})
    pct   = wb.add_format({'num_format': '0.0%'})

    # Column formats
    cols = list(rev_vs_target.columns)
    for cidx, cname in enumerate(cols):
        if cname in {'revenue','revenue_target','variance'}:
            wsS.set_column(cidx, cidx, 14, money)
        if cname == 'pct_to_target':
            wsS.set_column(cidx, cidx, 12, pct)
    # Conditional format: highlight misses < 100%
    last_row = len(rev_vs_target) + 1
    pct_col = cols.index('pct_to_target')
    wsS.conditional_format(1, pct_col, last_row, pct_col, {
        'type': 'cell', 'criteria': '<', 'value': 1,
        'format': wb.add_format({'bg_color': '#FCE4E4'})
    })

    # Insert charts
    wsS.insert_image('J2', str((OUT / 'weekly_by_region.png').resolve()))
    wsS.insert_image('J22', str((OUT / 'top5_products.png').resolve()))
```

Create Documentation (README.md)

```
# How to run
1) python data_gen.py
2) python main.py

What we get:
- output/dashboard.xlsx with three sheets
- two PNG charts embedded on the Summary sheet
```

Goes from raw CSVs to a readable Excel output.

Tips:

- Mixed dtypes in revenue_target: use pd.to_numeric(..., errors='coerce').
- NaT in order_date: ensure parse_dates and drop bad rows for this exercise.
- Chart paths: insert images with absolute paths (Path(...).resolve()).

Section 2: Building with AI and Spec-Driven Development

Spec driven development is a methodology for using AI to develop code. It is the process of first building up a rich description of the requirements of the project (i.e. the 'spec' or specifications) before asking the AI to convert that to an organised set of implementation steps and then write the code.

A summarised version of the process looks like:
Write detailed spec (User) → Convert to implementation steps (AI) → Write code (AI)

This method is important because of a simple but obvious fact: AI cannot read our minds. If an AI lacks an understanding of the context of the problem, then it realistically isn't able to solve the problem. A common pitfall is that this is often forgotten.

It's very easy to ask an AI to write code that *technically* works, however it might not actually address the problem correctly. One can ask something like: "Make me a dashboard in Python", while this may work, there are many unanswered questions. What data will it use? What will the dashboard look like? What tools and libraries will it use? The key is to provide all this information in advance via spec-driven development so that we don't leave anything to chance.

Below we will do a similar project to the one we just did but in three different ways (parts A, B and C below). For part A we will look at how to reproduce our project using AI. For part B we'll make an enhanced version. For part C we'll create an interactive prototype as a web application (which would typically require a much higher level of programming expertise).

Part A: Development with Cline

This demo shows the same KPI task built with an AI tool called Cline. Instead of writing the code from scratch, we can provide the AI a detailed set of specifications so that it can write the code. We'll start by writing out our requirements in a text file (.clinerules), ask Cline to plan out the implementation steps based on it, and request that the code be generated.

Setup

- Create a folder KPI_project and open it in VS Code.
- Ensure Cline is installed and set up with an API key.
- Copy the CSV data into ./KPI_project/data.
- Create a .clinerules text file (will be used to enforce project conventions).
- Create a kpi_dashboard.md text file (will be used for specifications/requirements).

Spec (.clinerules file)

We paste the following in a file called ".clinerules" which specifies the general background and requirements for the project. This can include general information about what version of python to use, approaches for solving problems and specific libraries/packages to use (where relevant).

```
# Project conventions for Cline
- Use Python 3.12 only.
- Use pip for installs.
- Project layout: data/, output/, main.py, data_gen.py, README.md, specs/kpi_dashboard.md
- Ask if unsure about next steps.
- Validate the final code by automatically creating and running tests.
```

Create the spec directly

Create a file called kpi_dashboard.md. In this file, we'll outline the requirements for the project, including the business outcomes, data schemas, KPI definitions, outputs, and acceptance checks. Paste the following into kpi_dashboard.md.

```
# KPI Dashboard Spec
Business outcome: Excel KPI pack with Summary, Weekly_Trend, Top_Products; two charts on
Summary.

Data: orders.csv, products.csv, targets.csv (Jan–Jun 2024, fixed seed).
KPIs: weekly revenue by region; current-month top 5 products; monthly revenue vs target with
pct_to_target and variance. Inspect a sample of these files to see the structure.
Expected outputs: output/dashboard.xlsx + PNG charts inserted; number formats as integers and
0.0%.
Technical: Python 3.12; pandas, numpy, matplotlib, xlsxwriter, openpyxl; scripts data_gen.py
and main.py.
Acceptance before 'done':
  1) output/dashboard.xlsx with Summary, Weekly_Trend, Top_Products
  2) PNG charts inserted on Summary
  3) pct_to_target correct; misses (<100%) highlighted
Ensure no negative revenue; unique order_id; some pct_to_target < 1.0; both charts visible on
Summary.
Ensure to obey the .clinerules context. Propose a minimal plan of:
- Files to create
- Packages to install
- Logical steps to be implemented
```

Now that we have written out the project spec (requirements) in kpi_dashboard.md, we can instruct Cline to actually build the project based on it. First, switch to Cline's chat window. Before we submit the prompt, ensure to select the 'Act Mode' so that Cline can make the implementation.

Validate (Ask Cline to verify the output)

- Open output/dashboard.xlsx and show the first rows of Summary.
- Report min/max of pct_to_target and confirm at least one value is < 1.0.
- Confirm both images are inserted on the Summary sheet.

- If something goes wrong, specify to Cline what the issue is and ask for it to be fixed..

Tips

- If Cline generates something incorrect, use the rollback feature (in the chat-window history)
- Break complex tasks/requests into smaller chunks with more detail
- If possible, review the generated code before approving
- If Cline gets stuck, stop and rephrase the request more specifically

Part B: Create A More Comprehensive Spec

If we were to create a more complex project, it is important to provide detailed information about the project, how the code should be structured, the features and the process steps.

The following specific method is inspired by the tool "Spec Kit" (we won't go into the use of that specific tool for the moment). Instead of having to create the specifications and requirements up-front, it allows us to work with the AI in creating these specifications by telling it to ask us questions. Ultimately we'll work with the AI to flesh out the following categories:

- **Constitution:** Define some general guiding principles for the project including formatting rules, regulations, etc.
- **Specify:** Define *what* to build (project requirements).
- **Clarify:** Explain underspecified or unique parts of the project which may be hard for the AI to understand or infer.
- **Plan:** State the technologies to use or even technical implementation steps.
- **Tasks:** Specify a list of tasks or steps that the AI should follow as part of the process.

To see more about spec-kit:

`https://github.com/github/spec-kit`

Create the Spec

Create a new folder called 'spec' in the project folder. We will now ask Cline to create the spec (in this folder) based on some questions that it will ask. We start by loosely defining the problem we wish to solve, and ask Cline to flesh out the details as a spec. Paste in the following prompt into Cline in the 'Planning' mode:

```
I want to build a Sales KPI Dashboard that converts CSV sales data into an Excel report with
charts.It should be based on the files inside the ./data folder. You will first assist in
creating a spec for the project, which will outline the detailed requirements for the project.
The specs (when complete) shall be written to the ./spec folder as separate md files.

Help me create specifications by asking a few questions for each area:
1. Constitution (constitution.md) - coding standards
2. Specify (specify.md) - what we're building
3. Clarify (clarify.md) - handle edge cases
4. Plan (plan-md) - technical approach
5. Tasks (tasks.md) - implementation steps

Keep each section concise (under 150 words). Save each to spec/[name].md once all questions
have been answered.

Start with the constitution and work through the rest. Ask me up to 3 questions about each (if
necessary), and feel free to skip sections if reasonable (with my approval). Once complete,
then ask for permission to write the md files in the ./spec folder, telling me to switch to the
'act' mode.
```

Now we can work with the AI to create the full spec by answering all the questions (and instructions) that Cline provides. After this is complete, switch to the 'act' mode to allow the spec to be saved (Cline should instruct us to do this).

Once the spec has been created, we paste in the following prompt with the 'act' mode enabled:

```
Build a Sales KPI Dashboard based on the specifications and requirements inside ./spec folder.
Ensure to follow all the instructions of all the md files in this directory. Once the code is
written, verify the code matches the requirements. Then finally execute the code to generate
the output.
```

Once this is done, the code should be generated along with the final Excel file.

Part C: Using Spec Kit to build a quality web app

We would like to show that non-coders can also generate a viable web app. The goal is to create a web application where we can upload CSVs (or click 'Generate demo data') to get interactive charts and filters, and can export the same Excel pack.

We will use the following prompt which loosely defines the problem, and instructs Cline to ask us questions for building the spec (which is similar to part B above).

```
I want to build a Sales KPI Dashboard web application that converts CSV sales data into an
Excel report with charts.It should be based on a CSV upload, but also testable via some dummy
data which can automatically be generated by a button 'Generate demo data'. You will first
assist in creating a spec for the project, which will outline the detailed requirements for the
project. The specs (when complete) shall be written to the ./spec folder as separate md files.

Help me create specifications by asking a few questions for each area:
1. Constitution (constitution.md) - coding standards
2. Specify (specify.md) - what we're building
3. Clarify (clarify.md) - handle edge cases
4. Plan (plan-md) - technical approach
5. Tasks (tasks.md) - implementation steps

Keep each section concise (under 150 words). Save each to spec/[name].md once all questions
have been answered.

Start with the constitution and work through the rest. Ask me up to 3 questions about each (if
necessary), and feel free to skip sections if reasonable (with my approval). Once complete,
then ask for permission to write the md files in the ./spec folder, telling me to switch to the
'act' mode.
```

Proposed architecture

As part of the answers to Cline, we should answer with these:

- Backend: FastAPI with endpoints: POST /api/upload (files), POST /api/kpi (compute), GET /api/export (download Excel).
- Frontend: Vite + React + Tailwind + Recharts.
- Processing: reuse the pandas logic in a shared module.
- Output: Excel export mirrors earlier deliverable.

Improvement via spec changes

- 1. First improvement: add WoW % deltas and a color scale for % to target; reflect both in frontend and Excel export.

```
Add a WoW % delta column to Summary and apply a color scale to
pct_to_target (>=1.0 greenish; <1.0 reddish). Update backend fields,
frontend table, and Excel export.

/speckit.plan
Confirm new fields and minimal code changes.

/speckit.tasks
/speckit.implement
```

- 2. Second improvement: saved filter presets and shareable links using query-string state in the URL; store presets in localStorage.

```
Add "Save preset" (name + filters) persisted to localStorage, and encode
current filters into the URL so views are shareable.

/speckit.plan
Outline minimal changes to state management and routing.
```

Conclusion

- Spec first, code second.
- For small changes, ask Cline to update the code
- If something is very wrong or major changes needed, fix the spec and rebuild.
- Agents are most useful when they plan first and only then run approved commands.