```
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv("daily_groceries_prices.csv", parse_dates=["date"])

# Set date as index
df.set_index("date", inplace=True)

df.head()
```

date	eggs	almond_life	lactose_milk	red_onions	brown_onions	ravioli	pasta_sauce	chicken_tikka	pesto
2025-05-13	8.7	2.0	1.9	4.5	3.6	5.3	2.3	4.0	5.3
2025-05-14	8.7	2.0	1.9	4.5	3.6	5.3	2.3	4.0	5.3
2025-05-15	8.7	2.0	1.9	4.5	3.6	5.3	2.3	4.0	5.3
2025-									

Next steps: Generate code with df | New interactive sheet

Goal

Make sure the data is loaded correctly, dates are parsed, and we understand the basic structure: number of rows, columns, and date range. First, I loaded the daily Woolworths pricing dataset into a Pandas dataframe, converted the date column to a proper datetime type, and set it as the index. This turns the table into a time-series, which is ideal for tracking how prices evolve over days and weeks. The dataset covers 185 days between 2025-05-13 and 2025-11-14 and includes daily prices for nine essential grocery items.

```
df.describe()
```

	eggs	almond_life	lactose_milk	red_onions	brown_onions	ravioli	pasta_sauce	chicken_tikka	pe
count	185.000000	185.000000	185.000000	185.000000	185.000000	185.000000	185.000000	185.000000	185.000
mean	8.647568	1.984595	1.931892	5.108108	2.768649	4.957838	3.151351	4.371351	4.618
std	0.197283	0.045659	0.046732	0.632188	0.461673	0.500712	0.676396	0.248454	0.467
min	7.000000	1.850000	1.900000	3.000000	2.500000	3.500000	2.300000	3.600000	4.300
25%	8.700000	2.000000	1.900000	4.900000	2.500000	4.500000	2.750000	4.500000	4.300
50%	8.700000	2.000000	1.900000	5.500000	2.500000	5.300000	2.750000	4.500000	4.300
75%	8.700000	2.000000	2.000000	5.500000	3.000000	5.300000	3.600000	4.500000	5.300
max	8.700000	2.000000	2.000000	5.500000	3.600000	5.300000	4.600000	4.500000	5.300

Using df.describe(), I generated key metrics for each product:

Mean Price → average daily price

Std Dev → volatility indicator

Min/Max → lowest and highest daily price

IQR (25%–75%) → consistency range

Key Findings :

Most products stayed stable for long periods.

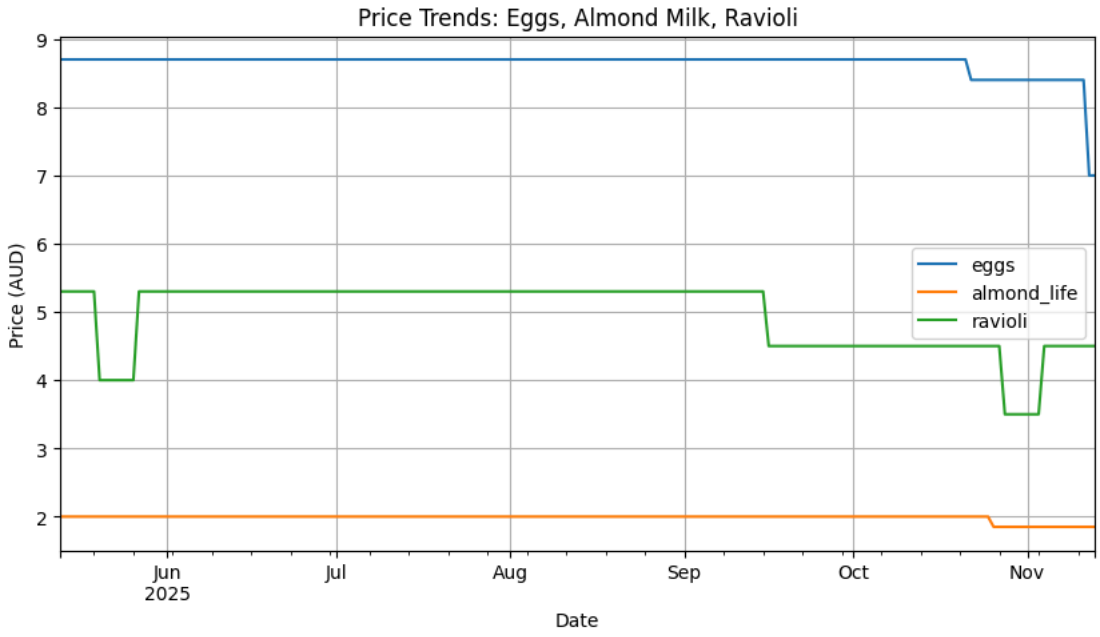
Products like brown onions, pasta sauce, red onions, ravioli showed noticeable fluctuations.

Items such as almond milk and lactose milk had extremely stable pricing.

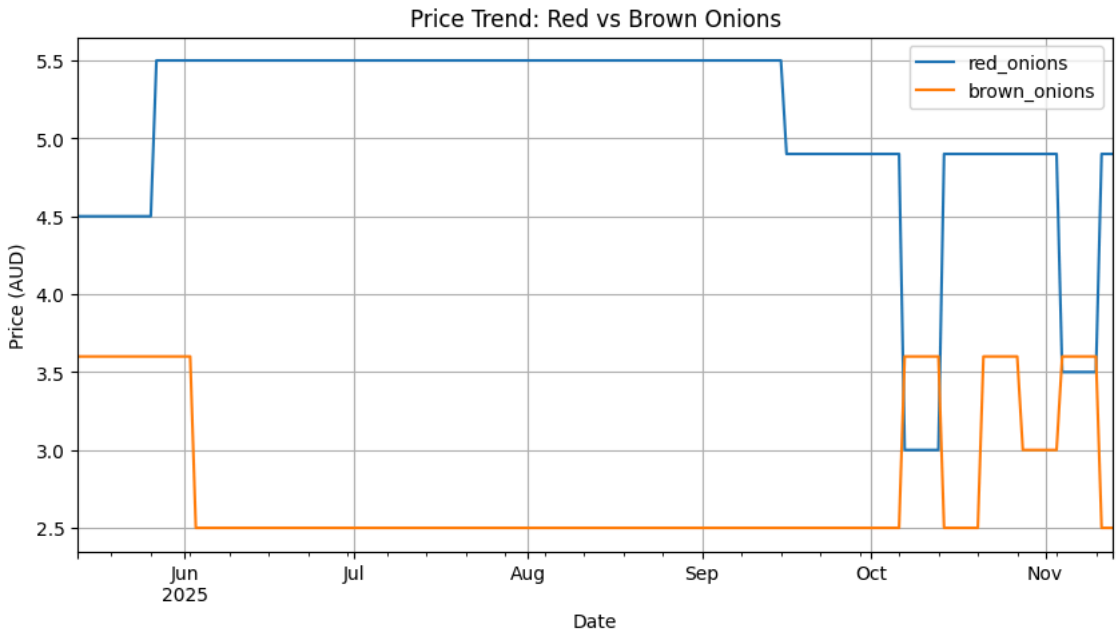
This establishes a baseline understanding before deeper analysis.

```
df[["eggs", "almond_life", "ravioli"]].plot(figsize=(10,5))
plt.title("Price Trends: Eggs, Almond Milk, Ravioli")
plt.ylabel("Price (AUD)")
plt.xlabel("Date")
plt.grid(True)
```

```
plt.show()
```




```
df[["red_onions", "brown_onions"]].plot(figsize=(10,5))
plt.title("Price Trend: Red vs Brown Onions")
plt.ylabel("Price (AUD)")
plt.xlabel("Date")
plt.grid(True)
plt.show()
```



Daily Price Trends (Line Charts)

I plotted daily price movements for:

Eggs, Almond Milk, Ravioli

Red vs Brown Onions

Other essential items

Key Observations :

Eggs stayed flat at ~8.7 AUD until a sharp drop in November.

Almond milk remained almost perfectly stable (1.95–2.00 AUD).

Ravioli dropped from 5.3 AUD to 3.5 AUD around Nov.

Red onions fluctuated heavily between 3.0–5.5 AUD.

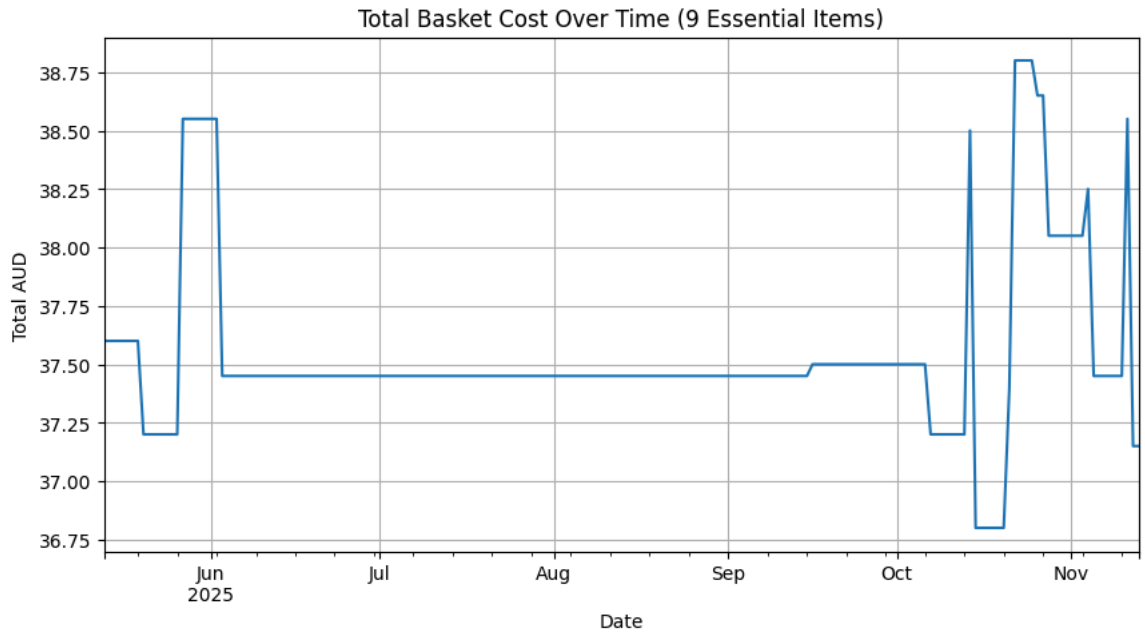
Brown onions dipped below 2.5 AUD then recovered.

This shows which items Woolworths changes frequently and which stay stable.

```
price_cols = df.columns.tolist()

df["basket_total"] = df[price_cols].sum(axis=1)

df["basket_total"].plot(figsize=(10,5))
plt.title("Total Basket Cost Over Time (9 Essential Items)")
plt.ylabel("Total AUD")
plt.xlabel("Date")
plt.grid(True)
plt.show()
```



Total Basket Cost Over Time

I created a combined basket_total metric by summing the daily price of all 9 items.

What the chart shows :

A long flat period around ~37.4 AUD

A mid-October dip (~36.8 AUD)

A November spike (up to ~38.8 AUD)

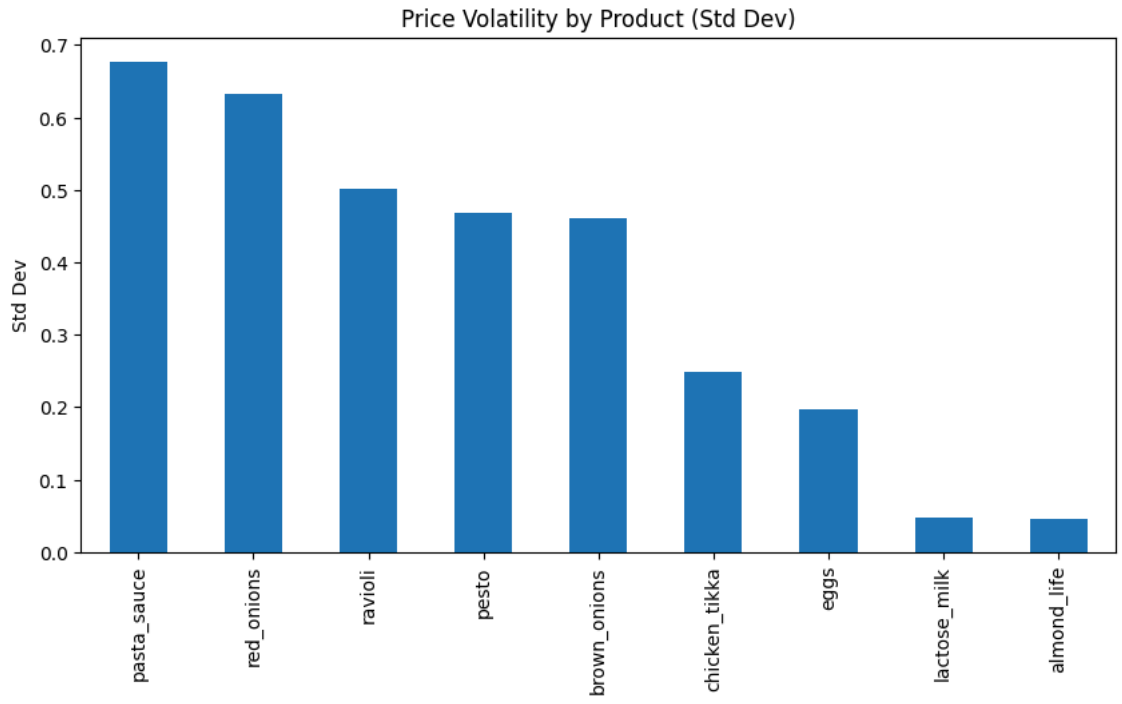
Several sudden up/down movements driven mostly by onion and pasta sauce price jumps

This gives a clear overview of real household cost impact.

```
vol = df[price_cols].std().sort_values(ascending=False)

vol.plot(kind="bar", figsize=(10,5))
plt.title("Price Volatility by Product (Std Dev)")
plt.ylabel("Std Dev")
plt.show()

vol
```



	0
pasta_sauce	0.676396
red_onions	0.632188
ravioli	0.500712
pesto	0.467322
brown_onions	0.461673
chicken_tikka	0.248454
eggs	0.197283
lactose_milk	0.046732
almond_life	0.045659

dtype: float64

Price Volatility (Standard Deviation)

I calculated the daily standard deviation for each product.

Most Volatile (Highest std-dev) :

Pasta Sauce

Red Onions

Ravioli

These items experienced the most inconsistent pricing.

Most Stable (Lowest std-dev) :

Almond Milk

Lactose Milk

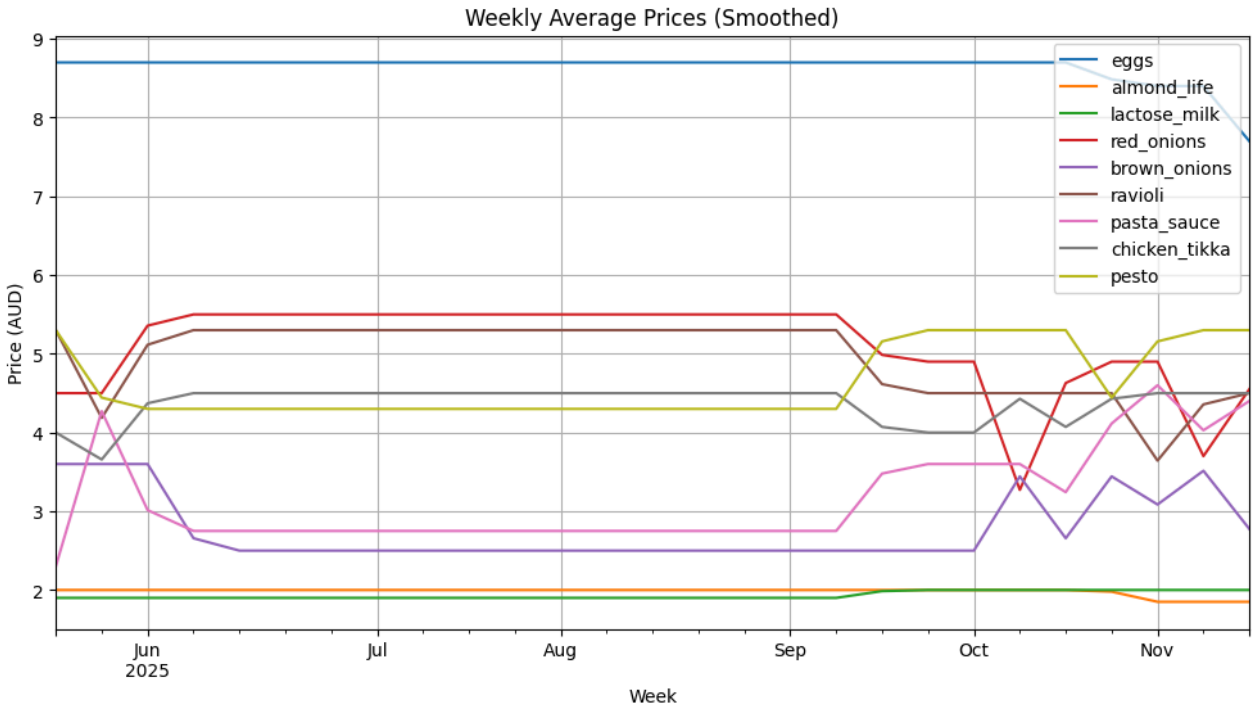
Eggs

These prices hardly changed across months.

This metric is essential for understanding predictability.

```
df_weekly = df[price_cols].resample("W").mean()

df_weekly.plot(figsize=(12,6))
plt.title("Weekly Average Prices (Smoothed)")
plt.ylabel("Price (AUD)")
plt.xlabel("Week")
plt.grid(True)
plt.show()
```



Weekly Average Prices (Smoothed Trends)

I used .resample("W").mean() to generate weekly averages to smooth out noise.

Insights :

Weekly data highlights long flat periods and sudden changes more clearly.

Pasta sauce shows the largest weekly spikes.

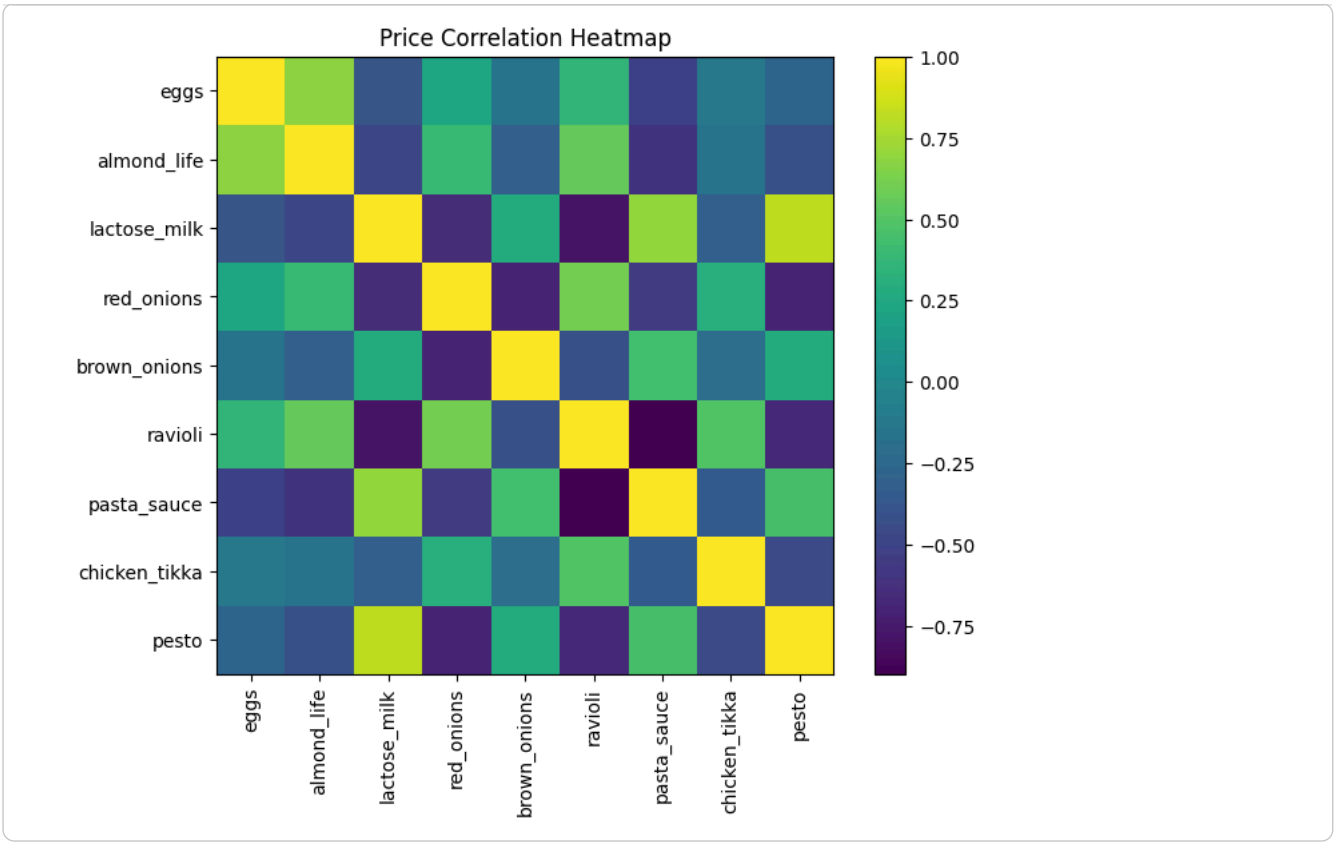
Onions show clear seasonality and supply-driven jumps.

Pesto fluctuates but follows a clearer mid-year trend.

This smoothed view helps explain medium-term price behavior.

```
corr = df[price_cols].corr()
corr


plt.figure(figsize=(8,6))
plt.imshow(corr, interpolation='nearest')
plt.colorbar()
plt.xticks(range(len(price_cols)), price_cols, rotation=90)
plt.yticks(range(len(price_cols)), price_cols)
plt.title("Price Correlation Heatmap")
plt.show()
```



Price Correlation Heatmap

I computed the correlation matrix to see how product prices move together.

Interpretation :

Several items show moderate positive correlations (e.g., onions ↔ pasta sauce).

Some items move independently, like almond milk.

This can indicate:

Shared supply chain shocks

Seasonal produce effects

Promotion cycles

A correlation map is a powerful visual for price relationships.

```
returns = df[price_cols].pct_change() * 100   # percentage change
returns.head()


for col in price_cols:
    print("\n==", col, "==")
    print(returns[col].sort_values(ascending=False).head(3))
```

```
2025-05-16    0.0
Name: eggs, dtype: float64

== almond_life ==
date
2025-05-14    0.0
2025-05-15    0.0
2025-05-16    0.0
Name: almond_life, dtype: float64

== lactose_milk ==
date
2025-09-16    5.263158
2025-05-14    0.000000
2025-05-16    0.000000
Name: lactose_milk, dtype: float64
```

```
Name: red_onions, dtype: float64

== brown_onions ==
date
2025-10-07    44.0
2025-10-21    44.0
2025-11-04    20.0
Name: brown_onions, dtype: float64

== ravioli ==
date
2025-05-27    32.500000
2025-11-04    28.571429
2025-05-16     0.000000
Name: ravioli, dtype: float64

== pasta_sauce ==
date
2025-05-20    100.000000
2025-10-22     58.620690
2025-09-16     30.909091
Name: pasta_sauce, dtype: float64

== chicken_tikka ==
date
2025-05-27    25.0
2025-10-21    12.5
2025-10-07    12.5
Name: chicken_tikka, dtype: float64

== pesto ==
date
2025-09-16    23.255814
2025-10-28    23.255814
2025-05-14     0.000000
Name: pesto, dtype: float64
```

% Price Change (Day-over-Day)

For each product, I calculated percentage change and extracted the top 3 biggest jumps.

Highlights :

Pasta Sauce had a +100% spike (largest change).

Red Onions jumped over +60% on certain dates.

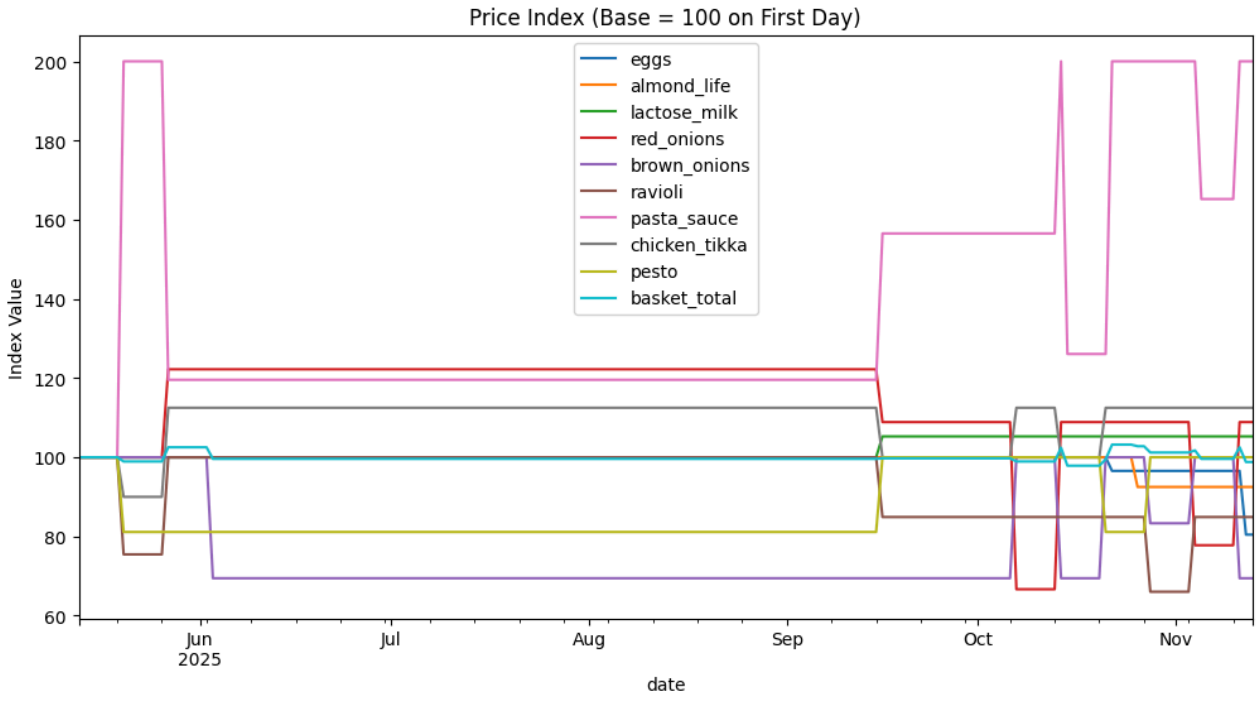
Ravioli had +30% days.

Chicken Tikka saw +25% price increases on two occasions.

Eggs & Almond Milk had almost no change throughout the dataset.

This helps identify shock days that affect consumer prices.

```
price_index = (df / df.iloc[0]) * 100
price_index.plot(figsize=(12,6))
plt.title("Price Index (Base = 100 on First Day)")
plt.ylabel("Index Value")
plt.show()
```



Price Index" (CPI-style index)

What it shows: How each product's price has changed relative to the first recorded day.

This highlights inflation-like behaviour we can directly compare which items became more expensive or cheaper over time, even if their original prices were different.

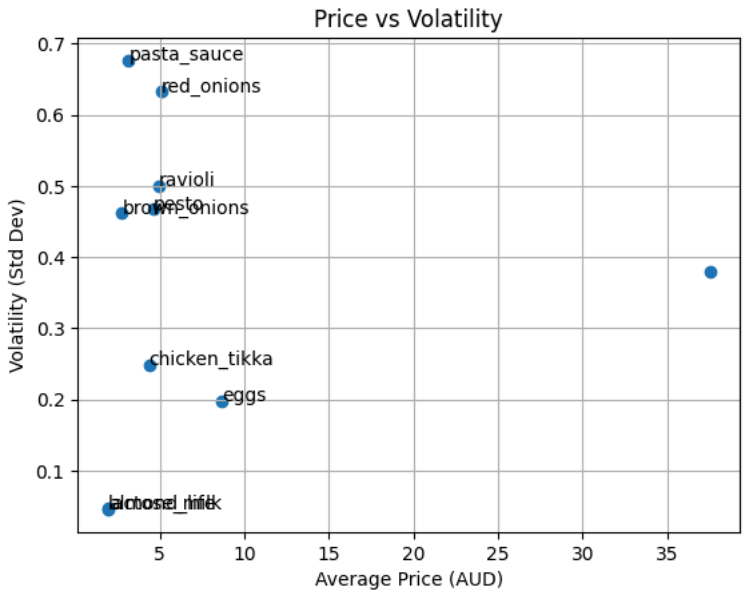
Pasta sauce and red onions show the largest relative price increase, while eggs and almond milk remained stable.

It shows inflation relative to starting point Makes every product comparable and can plot everything in one chart

```
means = df.mean()
vols = df.std()

plt.scatter(means, vols)
for p in price_cols:
    plt.text(means[p], vols[p], p)

plt.xlabel("Average Price (AUD)")
plt.ylabel("Volatility (Std Dev)")
plt.title("Price vs Volatility")
plt.grid(True)
plt.show()
```



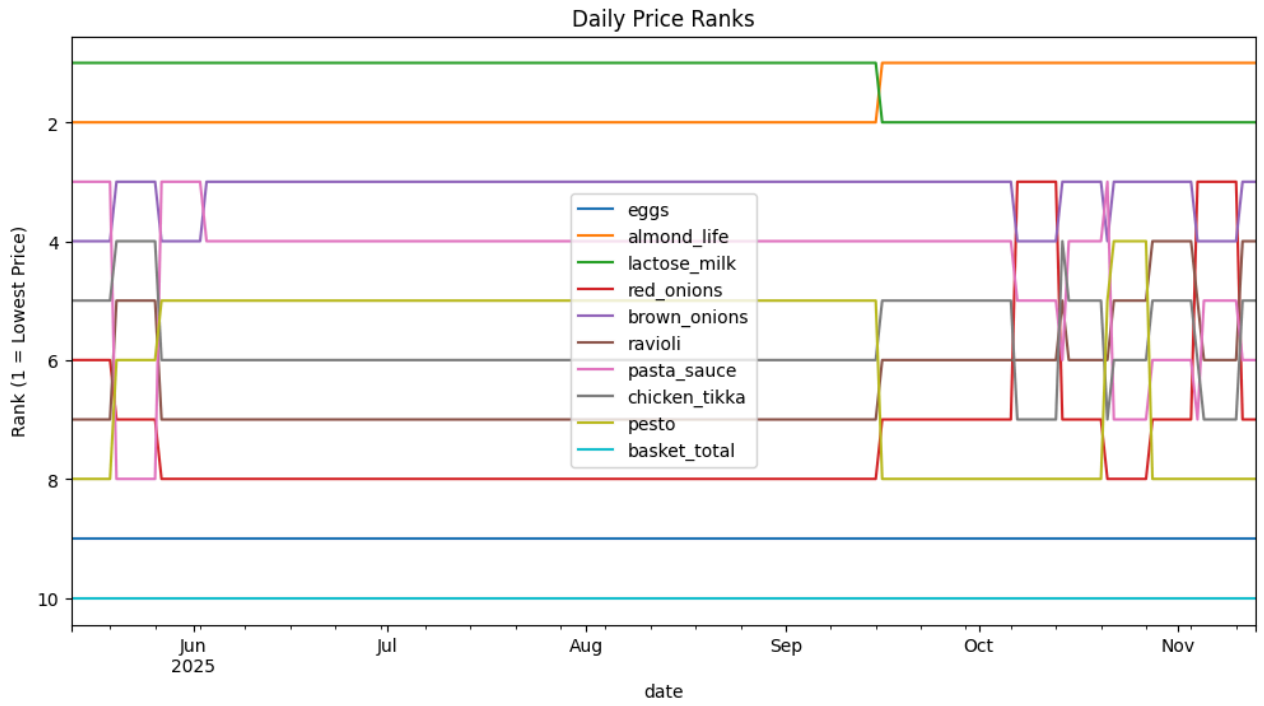
Price vs Volatility Scatter Plot

What it shows: Each product plotted by average price (x-axis) and volatility / standard deviation (y-axis).

It clearly identifies which items are expensive AND unstable, which are risky for consumers.

Pasta sauce is moderately priced but extremely volatile, while eggs are high priced but very stable.

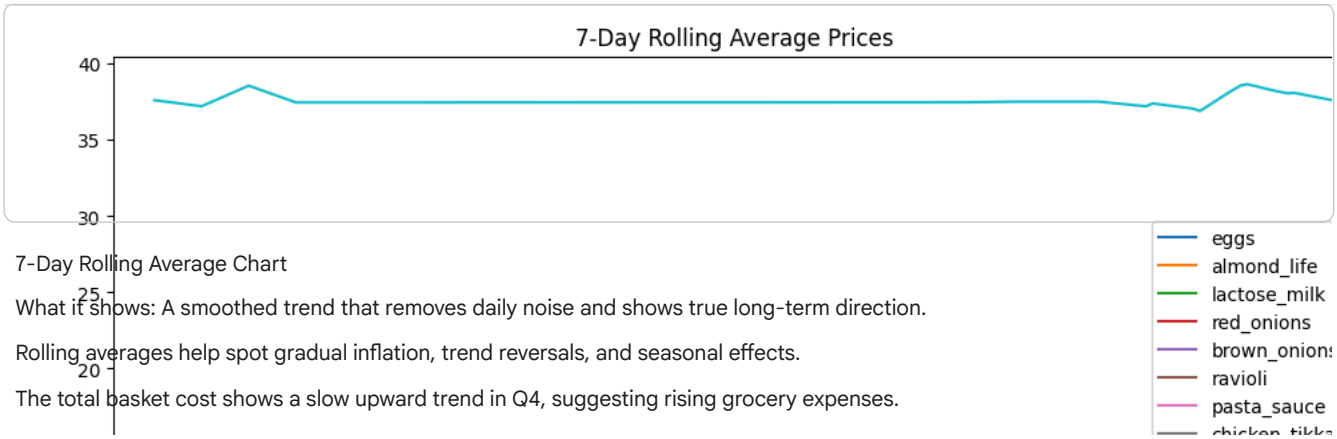
```
rank_df = df.rank(axis=1, method="first")
rank_df.plot(figsize=(12,6))
plt.title("Daily Price Ranks")
plt.ylabel("Rank (1 = Lowest Price)")
plt.gca().invert_yaxis()
plt.show()
```



Daily Price Ranking Chart

What it shows: Each product ranked daily from cheapest to most expensive. Shows competitive pricing, seasonal shifts, and promotional dips, which you can't see from raw prices alone. Onions frequently change rank due to price swings, indicating unstable supply or promotions.

```
df_rolling = df.rolling(7).mean()
df_rolling.plot(figsize=(12,6))
plt.title("7-Day Rolling Average Prices")
plt.show()
```



7-Day Rolling Average Chart

What it shows: A smoothed trend that removes daily noise and shows true long-term direction.

Rolling averages help spot gradual inflation, trend reversals, and seasonal effects.

The total basket cost shows a slow upward trend in Q4, suggesting rising grocery expenses.