

Pandas and Data Visualization

Table of contents

Creating a DataFrame	1
Adding Dates	1
Grouping and Annual Summaries	2
Plotting with Matplotlib	2
Analyzing “What If” Scenarios	3

Because `mortgagemath` returns pure Python dataclasses and standard library `Decimal` types, it integrates seamlessly with the broader Python data science ecosystem.

Converting an amortization schedule into a pandas `DataFrame` unlocks powerful vectorized analysis, grouping, and plotting capabilities.

Creating a DataFrame

Pass the generated `Installment` objects directly into the `DataFrame` constructor. The fields map naturally to columns:

```
import pandas as pd
from mortgagemath import us_30_year_fixed, amortization_schedule

# 1. Create a schedule
loan = us_30_year_fixed("300000", "6.5")
schedule = amortization_schedule(loan)

# 2. Convert to DataFrame
df = pd.DataFrame(schedule)

# 3. View the first few rows
print(df.head())
```

Output:

	number	payment	interest	principal	total_interest	balance
0	0	0.00	0.00	0.00	0.00	300000.00
1	1	1896.21	1625.00	271.21	1625.00	299728.79
2	2	1896.21	1623.53	272.68	3248.53	299456.11
3	3	1896.21	1622.05	274.16	4870.58	299181.95
4	4	1896.21	1620.57	275.64	6491.15	298906.31

Adding Dates

While `mortgagemath` provides integer payment numbers (to represent the pure mathematical sequence), you frequently want calendar dates in your `DataFrame`.

You can easily compute real payment dates by anchoring off a starting date and adding months using `pd.DateOffset` or similar logic.

```
# Create a date range for the payments (e.g., first payment on Jan 1, 2024)
start_date = pd.Timestamp("2024-01-01")

# Use df.index directly for the offset multiplier
df["date"] = [start_date + pd.DateOffset(months=i-1) if i > 0 else start_date -
```

```
pd.DateOffset(months=1) for i in df["number"]

# Set the date as the index
df.set_index("date", inplace=True)
print(df.head())
```

Grouping and Annual Summaries

Once in pandas, computing yearly totals is trivial. For instance, to calculate the total interest paid per calendar year (useful for tax reporting):

```
# Group by the year component of our datetime index
annual_summary = df.groupby(df.index.year).agg(
    total_payment=("payment", "sum"),
    total_interest=("interest", "sum"),
    total_principal=("principal", "sum"),
    end_balance=("balance", "last") # The balance at the end of the year
)

print(annual_summary.head())
```

Plotting with Matplotlib

A classic visualization for a mortgage is showing how the proportion of the monthly payment shifts from interest to principal over time.

We can plot this easily using matplotlib.

```
import matplotlib.pyplot as plt

# Filter out the initial period 0
df_plot = df[df["number"] > 0]

fig, ax = plt.subplots(figsize=(10, 6))

# Plot principal and interest stacked
ax.stackplot(
    df_plot["number"],
    df_plot["principal"].astype(float),
    df_plot["interest"].astype(float),
    labels=['Principal', 'Interest'],
    alpha=0.8
)

ax.set_title("Amortization Schedule: Principal vs Interest Over Time", fontsize=14)
ax.set_xlabel("Payment Number")
ax.set_ylabel("Monthly Payment Amount ($)")
ax.set_xlim(1, df_plot["number"].max())
ax.margins(x=0, y=0)
ax.legend(loc='upper left')

plt.tight_layout()
plt.show()
```

Analyzing “What If” Scenarios

Because mortgagemath guarantees cent-accuracy and exact end-of-term trueups, comparing schedules in pandas is a reliable way to compute the exact cost difference of various choices (like a 15-year vs 30-year term, or finding the exact point an ARM becomes more expensive than a fixed loan).

```
from mortgagemath import us_15_year_fixed

loan_15 = us_15_year_fixed("300000", "5.5")
df_15 = pd.DataFrame(amortization_schedule(loan_15))

total_interest_30 = df["interest"].sum()
total_interest_15 = df_15["interest"].sum()
savings = total_interest_30 - total_interest_15

print(f"30-Year Total Interest: ${total_interest_30:,.2f}")
print(f"15-Year Total Interest: ${total_interest_15:,.2f}")
print(f"Total Savings: ${savings:,.2f}")
```