Day 3, Session 2: Advanced Pandas & Funnel Analysis

Three-Day Data Analysis with Python Course

Session Overview

1. **Merging and Joining Datasets**
2. **Advanced Grouping and Pivoting**
3. **Time Series Basics**
4. **Punchline: Funnel Analysis**

Part 1: Merging and Joining Datasets

Why Merge Datasets?

Real-world data is often split across multiple tables:

- **Customers** table (id, name, email, signup_date)
- **Orders** table (order_id, customer_id, order_date, amount)
- **Products** table (product_id, name, category, price)

To answer business questions, you need to combine them!

Example: *"Which customers spend the most on electronics?"*
→ Requires joining Customers + Orders + Products

Sound familiar? It's like SQL JOINs!

Pandas Merge: Like SQL JOIN

```
# Merge two DataFrames
result = pd.merge(
    customers,              # Left table
    orders,                 # Right table
    on='customer_id',       # Join key
    how='inner'             # Join type
)
```

Join types (same as SQL):

- `inner`: Only matching rows from both tables
- `left`: All from left, matching from right
- `right`: All from right, matching from left
- `outer`: All rows from both tables

Merge Example

```python
# Customers table
customers = pd.DataFrame({
    'customer_id': [1, 2, 3],
    'name': ['Alice', 'Bob', 'Carol'],
    'city': ['NYC', 'LA', 'Chicago']
})

# Orders table
orders = pd.DataFrame({
    'order_id': [101, 102, 103, 104],
    'customer_id': [1, 1, 2, 4],  # Note: customer 4 doesn't exist!
    'amount': [100, 150, 200, 250]
})

# Inner join (only matching customers)
merged = pd.merge(customers, orders, on='customer_id', how='inner')
```

Result: 3 rows (Alice has 2 orders, Bob has 1, Carol has none, customer 4 excluded)

Different Join Types

```python
# Left join (all customers, matching orders)
left_merged = pd.merge(customers, orders, on='customer_id', how='left')
# Result: Carol appears with NaN order_id and amount

# Right join (all orders, matching customers)
right_merged = pd.merge(customers, orders, on='customer_id', how='right')
# Result: order from customer 4 appears with NaN name and city

# Outer join (all customers AND all orders)
outer_merged = pd.merge(customers, orders, on='customer_id', how='outer')
# Result: Carol AND customer 4 both appear
```

Choose based on your analysis needs!

Merging on Different Column Names

```
# When join keys have different names
customers = pd.DataFrame({
    'id': [1, 2, 3],
    'name': ['Alice', 'Bob', 'Carol']
})

orders = pd.DataFrame({
    'order_id': [101, 102],
    'cust_id': [1, 1],  # Different name!
    'amount': [100, 150]
})

# Specify left_on and right_on
merged = pd.merge(
    customers,
    orders,
    left_on='id',
    right_on='cust_id',
    how='inner'
)
```

Merging on Multiple Columns

```python
# Join on multiple keys
sales = pd.DataFrame({
    'date': ['2024-01-01', '2024-01-01', '2024-01-02'],
    'region': ['East', 'West', 'East'],
    'sales': [1000, 1500, 1200]
})

targets = pd.DataFrame({
    'date': ['2024-01-01', '2024-01-02'],
    'region': ['East', 'East'],
    'target': [1100, 1300]
})

# Merge on both date AND region
merged = pd.merge(sales, targets, on=['date', 'region'], how='left')
```

Useful when composite keys uniquely identify rows

Concatenating DataFrames

Concatenation: Stacking DataFrames vertically or horizontally

```
# Stack DataFrames vertically (append rows)
df1 = pd.DataFrame({'A': [1, 2], 'B': [3, 4]})
df2 = pd.DataFrame({'A': [5, 6], 'B': [7, 8]})

result = pd.concat([df1, df2], ignore_index=True)
# Result: 4 rows, columns A and B

# Stack horizontally (add columns)
result = pd.concat([df1, df2], axis=1)
# Result: 2 rows, columns A, B, A, B (duplicates!)
```

Use concat when columns match and you want to combine datasets from same source

Merge vs Concat vs Join

Operation	Use Case	Example
`merge()`	Combine based on key column(s)	Join customers and orders
`concat()`	Stack same-structure DataFrames	Combine monthly exports
`join()`	Merge on index	Quick index-based joins

Most common: `merge()` for relational data

Part 2: Advanced Grouping and Pivoting

Pivot Tables

Pivot table: Reshape data to show relationships between categories

Like Excel pivot tables but more powerful!

```
# From long format to wide format
sales = pd.DataFrame({
    'date': ['2024-01', '2024-01', '2024-02', '2024-02'],
    'region': ['East', 'West', 'East', 'West'],
    'sales': [1000, 1500, 1200, 1800]
})

pivot = sales.pivot(index='date', columns='region', values='sales')
```

Result:

```
region    East  West
date
2024-01   1000  1500
```

Pivot Table with Aggregation

```python
# When you have duplicate combinations, use pivot_table with aggregation
data = pd.DataFrame({
    'date': ['2024-01', '2024-01', '2024-01', '2024-02'],
    'region': ['East', 'West', 'East', 'East'],
    'sales': [1000, 1500, 800, 1200]
})

# Aggregate with mean (or sum, count, etc.)
pivot_table = data.pivot_table(
    index='date',
    columns='region',
    values='sales',
    aggfunc='sum',    # or 'mean', 'count', etc.
    fill_value=0      # Replace NaN with 0
)
```

Result: East column shows 1800 for 2024-01 (sum of 1000 + 800)

Practical Pivot Table Example

```
# Sales by region and product category
sales_data = pd.DataFrame({
    'region': ['East', 'West', 'East', 'West', 'East'],
    'category': ['Electronics', 'Clothing', 'Electronics', 'Clothing', 'Home'],
    'revenue': [5000, 3000, 4500, 3500, 2000]
})

pivot = sales_data.pivot_table(
    index='region',
    columns='category',
    values='revenue',
    aggfunc='sum',
    fill_value=0
)
```

Business question answered: "What categories perform best in each region?"

Crosstabs

Crosstab: Like pivot table but for counting frequencies

```
# Count occurrences of category combinations
orders = pd.DataFrame({
    'customer_tier': ['Gold', 'Silver', 'Gold', 'Bronze', 'Silver'],
    'product_category': ['Electronics', 'Clothing', 'Electronics', 'Home', 'Clothing']
})

crosstab = pd.crosstab(
    orders['customer_tier'],
    orders['product_category']
)
```

Result: Table showing count of each tier-category combination

Add percentages:

```
pd.crosstab(df['tier'], df['category'], normalize='index')  # Row percentages
```

Multi-Index DataFrames

Multi-index: Multiple levels in row or column index

```
# GroupBy with multiple columns creates multi-index
grouped = df.groupby(['region', 'product'])['sales'].sum()

# Result has two-level index
# region  product
# East    A          1000
#         B          1500
# West    A          1200
#         B          1800
```

Accessing multi-index data:

```
grouped.loc['East']      # All products in East
grouped.loc['East', 'A'] # Specific combination
```

Resetting Multi-Index

Flatten multi-index to regular columns:

```python
# Multi-index from groupby
grouped = df.groupby(['region', 'product'])['sales'].sum()

# Reset index to regular DataFrame
flat = grouped.reset_index()

# Now you have columns: region, product, sales
```

Use this when:

- You want to merge with other DataFrames
- You need to filter or sort by the index columns
- You want to export to CSV/Excel

Window Functions

Window functions: Calculate rolling statistics

```
# Rolling average (moving average)
df['rolling_avg'] = df['sales'].rolling(window=7).mean()

# Rolling sum
df['rolling_sum'] = df['sales'].rolling(window=30).sum()

# Expanding window (cumulative)
df['cumulative_sum'] = df['sales'].expanding().sum()
```

Use cases:

- Smooth out noise in time series
- Calculate moving averages (7-day, 30-day)
- Cumulative totals

Window Function Example

```
# Daily sales with 7-day moving average
sales = pd.DataFrame({
    'date': pd.date_range('2024-01-01', periods=30, freq='D'),
    'sales': [1000, 1100, 950, 1200, 1150, 1300, ...]  # 30 days
})

# Calculate 7-day moving average
sales['7day_avg'] = sales['sales'].rolling(window=7).mean()

# First 6 rows will be NaN (not enough data for window)
# Row 7 onward: average of current + previous 6 days
```

Great for smoothing noisy data and identifying trends!

Part 3: Time Series Basics

Working with Datetime Data

Remember from Day 2: `pd.to_datetime()` converts strings to datetime

```
df['date'] = pd.to_datetime(df['date_string'])
```

Now you can use the `.dt` accessor:

```
df['year'] = df['date'].dt.year
df['month'] = df['date'].dt.month
df['day'] = df['date'].dt.day
df['day_of_week'] = df['date'].dt.day_name()
df['quarter'] = df['date'].dt.quarter
df['hour'] = df['date'].dt.hour  # If datetime includes time
```

Datetime Filtering

```
# Filter by date range
jan_sales = df[(df['date'] >= '2024-01-01') &
               (df['date'] < '2024-02-01')]

# Filter by year
sales_2024 = df[df['date'].dt.year == 2024]

# Filter by month
january = df[df['date'].dt.month == 1]

# Filter by day of week (0=Monday, 6=Sunday)
weekends = df[df['date'].dt.dayofweek >= 5]
```

Pandas understands date comparisons!

Resampling Time Series

Resampling: Aggregate data by time period

```
# Daily sales → Monthly sales
daily_sales = pd.DataFrame({
    'date': pd.date_range('2024-01-01', periods=365, freq='D'),
    'sales': np.random.randint(800, 1500, 365)
})

# Set date as index (required for resample)
daily_sales = daily_sales.set_index('date')

# Resample to monthly totals
monthly = daily_sales.resample('M').sum()

# Resample to weekly averages
weekly = daily_sales.resample('W').mean()
```

Frequencies: 'D' (day), 'W' (week), 'M' (month), 'Q' (quarter), 'Y' (year)

Time-based Aggregations

```
# Group by month
monthly_revenue = df.groupby(df['date'].dt.to_period('M'))['revenue'].sum()

# Group by year and quarter
quarterly = df.groupby([
    df['date'].dt.year,
    df['date'].dt.quarter
])['sales'].sum()

# Group by day of week
by_weekday = df.groupby(df['date'].dt.day_name())['sales'].mean()
```

Business insights:

- Which months have highest revenue?
- Is there a weekly pattern (weekends vs weekdays)?
- How does this year compare to last year?

Part 4: Funnel Analysis

The Punchline!

What is a Funnel?

Funnel: Sequential stages that users/customers go through

Example: E-commerce funnel

1. **Visit website** → 10,000 visitors
2. **View product** → 3,000 viewers (30%)
3. **Add to cart** → 1,200 carts (40% of viewers)
4. **Purchase** → 600 purchases (50% of carts)

Conversion rate: Percentage moving from one stage to the next

Why it matters: Identify where customers drop off!

Business Value of Funnels

Questions funnels answer:

- Where do we lose customers?
- Which stage has the lowest conversion?
- Which user segment converts best?
- Is our checkout process broken?

Actions you can take:

- Improve low-performing stages
- A/B test changes, allocate resources
- Measure impact of improvements

Building a Funnel with Pandas

Example: Customer lifecycle funnel

```python
# Customer data with activity flags
customers = pd.DataFrame({
    'customer_id': range(1, 1001),
    'registered': True,
    'first_purchase': [True] * 600 + [False] * 400,
    'repeat_purchase': [True] * 300 + [False] * 700,
    'loyal_customer': [True] * 150 + [False] * 850
})

# Count customers at each stage
funnel = {
    'Registered': customers['registered'].sum(),
    'First Purchase': customers['first_purchase'].sum(),
    'Repeat Purchase': customers['repeat_purchase'].sum(),
    'Loyal Customer': customers['loyal_customer'].sum()
}
```

Calculating Conversion Rates

```
# Funnel stages
funnel_data = pd.DataFrame({
    'stage': ['Registered', 'First Purchase', 'Repeat', 'Loyal'],
    'count': [1000, 600, 300, 150]
})

# Calculate conversion rates (% from previous stage)
funnel_data['conversion_rate'] = (
    funnel_data['count'] / funnel_data['count'].shift(1) * 100
)

# Overall conversion (from first to last stage)
overall_conversion = (150 / 1000) * 100  # 15%
```

Insights:

- Registration → First purchase: 60%

- First → Repeat: 50%

Visualizing Funnels with Plotly

```
import plotly.graph_objects as go

fig = go.Figure(go.Funnel(
    y=['Registered', 'First Purchase', 'Repeat Purchase', 'Loyal'],
    x=[1000, 600, 300, 150],
    textinfo='value+percent previous'
))

fig.update_layout(title='Customer Lifecycle Funnel')
fig.show()
```

Result: Beautiful funnel chart showing drop-offs at each stage!

Segmented Funnel Analysis

Compare funnels across segments:

```python
# Funnel by customer segment
segments = ['Premium', 'Standard']

for segment in segments:
    segment_data = customers[customers['tier'] == segment]

    funnel = {
        'Registered': len(segment_data),
        'First Purchase': segment_data['first_purchase'].sum(),
        'Repeat': segment_data['repeat_purchase'].sum(),
        'Loyal': segment_data['loyal_customer'].sum()
    }

    print(f"{segment} funnel:", funnel)
```

Insight: Do premium customers convert better? Where's the difference?

Real-World Funnel Example

E-commerce checkout funnel:

```python
# Event-based data
events = pd.DataFrame({
    'user_id': [1, 1, 1, 2, 2, 3, 3, 3, 4],
    'event': ['view', 'cart', 'purchase', 'view', 'cart', 'view', 'cart', 'purchase', 'view']
})

# Count unique users at each stage
funnel = {
    'Viewed Product': events[events['event'] == 'view']['user_id'].nunique(),
    'Added to Cart': events[events['event'] == 'cart']['user_id'].nunique(),
    'Purchased': events[events['event'] == 'purchase']['user_id'].nunique()
}

# Result: 4 viewed, 3 added to cart, 2 purchased
```

Combining Everything

Funnel analysis uses all the skills we've learned:

1. **Data Cleaning** - Handle missing events, deduplicate
2. **Merging** - Join user properties with event data
3. **Grouping** - Count users at each stage
4. **Time Series** - Track funnel changes over time
5. **Visualization** - Communicate findings

This is how you do real data analysis!

Session 2 Summary

✓ **Merging**: Combine datasets like SQL JOINs
✓ **Pivot Tables**: Reshape data for analysis
✓ **Time Series**: Extract datetime features, resample
✓ **Funnel Analysis**: Measure conversion through stages

Key Takeaways

1. **Merge datasets** to enrich your analysis (inner, left, right, outer)
2. **Pivot tables** reshape data for easy comparison
3. `.dt` **accessor** unlocks datetime analysis
4. **Funnels measure conversion** and identify drop-offs
5. **Combine techniques** to answer complex business questions

These are the advanced tools that separate analysts from experts!

Questions?

Break time! ☕

Then: **Practical Session - E-commerce Analytics Capstone Project**

Put everything together in a real-world analysis!