Week 3, Lecture 6

Data Analysis with Pandas & Matplotlib Visualization

Lecture Overview

This lecture advances your pandas skills with groupby operations, data aggregation, merging/joining datasets, and introduces data visualization using Matplotlib. These techniques are essential for exploratory data analysis (EDA) and understanding patterns in your datasets before building machine learning models.

Topics Covered:

• GroupBy operations and aggregation

• Merging and joining DataFrames

• Data transformation and reshaping

• Matplotlib basics: plots, customization

• Statistical visualizations

• Multiple subplots and figure layouts

1. GroupBy Operations

1.1 Basic GroupBy

```
import pandas as pd
import numpy as np

# Create sample data
data = {
    'Department': ['Engineering', 'Marketing', 'Engineering', 'Sales',
                   'Marketing', 'Engineering', 'Sales', 'Marketing'],
    'Employee': ['Alice', 'Bob', 'Charlie', 'David',
                 'Eve', 'Frank', 'Grace', 'Henry'],
    'Salary': [70000, 65000, 80000, 60000, 68000, 75000, 62000, 70000],
    'Years': [2, 5, 8, 3, 6, 4, 2, 7]
}
df = pd.DataFrame(data)

print("GroupBy Operations")
print("="*70)
```

```
print("Original DataFrame:")
print(df)

# Group by Department
grouped = df.groupby('Department')

# Basic aggregation
print("\nAverage salary by department:")
print(grouped['Salary'].mean())

print("\nMultiple aggregations:")
print(grouped['Salary'].agg(['mean', 'min', 'max', 'count']))
```

```
GroupBy Operations
======================================================================
Original DataFrame:
   Department Employee  Salary  Years
0  Engineering    Alice   70000      2
1    Marketing      Bob   65000      5
2  Engineering  Charlie   80000      8
3        Sales    David   60000      3
4    Marketing      Eve   68000      6
5  Engineering    Frank   75000      4
6        Sales    Grace   62000      2
7    Marketing    Henry   70000      7

Average salary by department:
Department
Engineering    75000.0
Marketing      67666.7
Sales          61000.0
Name: Salary, dtype: float64

Multiple aggregations:
              mean    min    max  count
Department
Engineering  75000.0  70000  80000      3
Marketing    67666.7  65000  70000      3
Sales        61000.0  60000  62000      2
```

1.2 Advanced GroupBy

```
# Multiple aggregations per column
print("Advanced GroupBy")
print("="*70)

# Different aggregations for different columns
agg_dict = {
```

```
    'Salary': ['mean', 'sum'],
    'Years': ['mean', 'max']
}
result = df.groupby('Department').agg(agg_dict)
print("Different aggregations per column:")
print(result)

# Custom aggregation function
def salary_range(x):
    return x.max() - x.min()

print("\nCustom aggregation (salary range):")
print(df.groupby('Department')['Salary'].agg(salary_range))

# Group by multiple columns
df['Experience'] = pd.cut(df['Years'], bins=[0, 3, 6, 10],
                          labels=['Junior', 'Mid', 'Senior'])

print("\nGroup by Department and Experience:")
multi_group = df.groupby(['Department', 'Experience'])['Salary'].mean()
print(multi_group)
```

```
Advanced GroupBy
========================================================================
Different aggregations per column:
          Salary         Years
           mean    sum  mean  max
Department
Engineering  75000.0  225000  4.7   8
Marketing    67666.7  203000  6.0   7
Sales        61000.0  122000  2.5   3

Custom aggregation (salary range):
Department
Engineering    10000
Marketing       5000
Sales           2000
Name: Salary, dtype: int64

Group by Department and Experience:
Department   Experience
Engineering  Junior       70000.0
             Mid          75000.0
             Senior       80000.0
Marketing    Mid          66500.0
             Senior       70000.0
```

```
Sales       Junior       61000.0
Name: Salary, dtype: float64
```

2. Merging and Joining DataFrames

2.1 Concatenation

```
# Concatenating DataFrames
print("Concatenation")
print("="*70)

df1 = pd.DataFrame({
    'Name': ['Alice', 'Bob'],
    'Score': [85, 90]
})

df2 = pd.DataFrame({
    'Name': ['Charlie', 'David'],
    'Score': [88, 92]
})

print("DataFrame 1:")
print(df1)
print("\nDataFrame 2:")
print(df2)

# Vertical concatenation (stack rows)
concat_vertical = pd.concat([df1, df2], ignore_index=True)
print("\nVertical concatenation:")
print(concat_vertical)

# Horizontal concatenation (stack columns)
df3 = pd.DataFrame({'Age': [25, 30]})
concat_horizontal = pd.concat([df1, df3], axis=1)
print("\nHorizontal concatenation:")
print(concat_horizontal)
```

```
Concatenation
======================================================================
DataFrame 1:
   Name  Score
0  Alice    85
1    Bob    90

DataFrame 2:
     Name  Score
0  Charlie    88
```

```
1   David   92
```

Vertical concatenation:

```
    Name  Score
0   Alice    85
1     Bob    90
2 Charlie    88
3   David    92
```

Horizontal concatenation:

```
   Name  Score  Age
0 Alice    85   25
1   Bob    90   30
```

2.2 Merge (Join)

```
# Merging DataFrames (SQL-style joins)
print("Merging DataFrames")
print("="*70)

employees = pd.DataFrame({
    'EmployeeID': [1, 2, 3, 4],
    'Name': ['Alice', 'Bob', 'Charlie', 'David'],
    'DeptID': [101, 102, 101, 103]
})

departments = pd.DataFrame({
    'DeptID': [101, 102, 103],
    'Department': ['Engineering', 'Marketing', 'Sales']
})

print("Employees:")
print(employees)
print("\nDepartments:")
print(departments)

# Inner join (default)
merged_inner = pd.merge(employees, departments, on='DeptID')
print("\nInner join:")
print(merged_inner)

# Left join
employees_new = pd.DataFrame({
    'EmployeeID': [5],
    'Name': ['Eve'],
    'DeptID': [104]  # Department doesn't exist
})
```

```
all_employees = pd.concat([employees, employees_new], ignore_index=True)

merged_left = pd.merge(all_employees, departments, on='DeptID', how='left')
print("\nLeft join (with missing department):")
print(merged_left)
```

```
Merging DataFrames
========================================================================
Employees:
  EmployeeID    Name  DeptID
0          1   Alice     101
1          2     Bob     102
2          3 Charlie     101
3          4   David     103

Departments:
  DeptID   Department
0    101  Engineering
1    102    Marketing
2    103      Sales

Inner join:
  EmployeeID    Name  DeptID   Department
0          1   Alice     101  Engineering
1          3 Charlie     101  Engineering
2          2     Bob     102    Marketing
3          4   David     103      Sales

Left join (with missing department):
  EmployeeID    Name  DeptID   Department
0          1   Alice     101  Engineering
1          2     Bob     102    Marketing
2          3 Charlie     101  Engineering
3          4   David     103      Sales
4          5     Eve     104        NaN
```

3. Data Transformation

3.1 Apply and Map

```
# Apply functions to data
print("Apply and Map")
print("="*70)

df_salary = pd.DataFrame({
    'Name': ['Alice', 'Bob', 'Charlie'],
    'Salary': [70000, 80000, 75000]
})
```

```
print("Original DataFrame:")
print(df_salary)

# Apply function to column
df_salary['Salary_K'] = df_salary['Salary'].apply(lambda x: x / 1000)
print("\nAfter applying lambda (salary in thousands):")
print(df_salary)

# Map for categorical data
salary_category = {
    70000: 'Standard',
    75000: 'Standard',
    80000: 'High'
}
df_salary['Category'] = df_salary['Salary'].map(salary_category)
print("\nAfter mapping to categories:")
print(df_salary)

# Apply to entire row
def bonus_calculator(row):
    if row['Salary'] > 75000:
        return row['Salary'] * 0.15
    else:
        return row['Salary'] * 0.10

df_salary['Bonus'] = df_salary.apply(bonus_calculator, axis=1)
print("\nWith calculated bonus:")
print(df_salary)
```

```
Apply and Map
========================================================================
Original DataFrame:
    Name  Salary
0   Alice   70000
1     Bob   80000
2 Charlie   75000

After applying lambda (salary in thousands):
    Name  Salary  Salary_K
0   Alice   70000     70.0
1     Bob   80000     80.0
2 Charlie   75000     75.0

After mapping to categories:
    Name  Salary  Salary_K Category
0   Alice   70000     70.0 Standard
```

```
1     Bob   80000     80.0     High
2 Charlie   75000     75.0 Standard
```

With calculated bonus:

```
      Name  Salary  Salary_K Category    Bonus
0    Alice   70000     70.0 Standard   7000.0
1      Bob   80000     80.0     High  12000.0
2  Charlie   75000     75.0 Standard   7500.0
```

3.2 Pivot Tables

```
# Pivot tables for reshaping
print("Pivot Tables")
print("="*70)

sales_data = pd.DataFrame({
    'Date': ['2024-01', '2024-01', '2024-02', '2024-02'],
    'Product': ['A', 'B', 'A', 'B'],
    'Sales': [100, 150, 120, 160],
    'Region': ['East', 'East', 'West', 'West']
})

print("Sales data:")
print(sales_data)

# Create pivot table
pivot = sales_data.pivot_table(
    values='Sales',
    index='Date',
    columns='Product',
    aggfunc='sum'
)

print("\nPivot table (Sales by Date and Product):")
print(pivot)

# Pivot with multiple aggregations
pivot_multi = sales_data.pivot_table(
    values='Sales',
    index='Region',
    columns='Product',
    aggfunc=['sum', 'mean']
)

print("\nMultiple aggregations:")
print(pivot_multi)
```

```
Pivot Tables
======================================================================
Sales data:
   Date Product  Sales Region
0  2024-01     A   100   East
1  2024-01     B   150   East
2  2024-02     A   120   West
3  2024-02     B   160   West

Pivot table (Sales by Date and Product):
Product    A    B
Date
2024-01  100  150
2024-02  120  160

Multiple aggregations:
       sum      mean
Product   A   B   A   B
Region
East    100  150  100  150
West    120  160  120  160
```

4. Matplotlib Visualization

4.1 Basic Plots

```
import matplotlib.pyplot as plt

# Create sample data
x = np.linspace(0, 10, 100)
y1 = np.sin(x)
y2 = np.cos(x)

print("Matplotlib Basics")
print("="*70)

# Line plot
plt.figure(figsize=(10, 6))
plt.plot(x, y1, label='sin(x)', linewidth=2)
plt.plot(x, y2, label='cos(x)', linewidth=2, linestyle='--')
plt.xlabel('x', fontsize=12)
plt.ylabel('y', fontsize=12)
plt.title('Trigonometric Functions', fontsize=14, fontweight='bold')
plt.legend()
plt.grid(True, alpha=0.3)
plt.savefig('trig_plot.png', dpi=100, bbox_inches='tight')
```

```
print("Saved: trig_plot.png")

# Scatter plot
np.random.seed(42)
x_scatter = np.random.randn(100)
y_scatter = 2 * x_scatter + np.random.randn(100)

plt.figure(figsize=(8, 6))
plt.scatter(x_scatter, y_scatter, alpha=0.6, s=50, c='blue')
plt.xlabel('X', fontsize=12)
plt.ylabel('Y', fontsize=12)
plt.title('Scatter Plot', fontsize=14, fontweight='bold')
plt.grid(True, alpha=0.3)
plt.savefig('scatter_plot.png', dpi=100, bbox_inches='tight')
print("Saved: scatter_plot.png")
```

```
Matplotlib Basics
======================================================================
Saved: trig_plot.png
Saved: scatter_plot.png
```

4.2 Statistical Plots

```
# Statistical visualizations
print("Statistical Plots")
print("="*70)

# Histogram
data_hist = np.random.randn(1000)

plt.figure(figsize=(10, 6))
plt.hist(data_hist, bins=30, edgecolor='black', alpha=0.7)
plt.xlabel('Value', fontsize=12)
plt.ylabel('Frequency', fontsize=12)
plt.title('Histogram of Normal Distribution', fontsize=14, fontweight='bold')
plt.grid(True, alpha=0.3, axis='y')
plt.savefig('histogram.png', dpi=100, bbox_inches='tight')
print("Saved: histogram.png")

# Box plot
data_box = [np.random.randn(100) + i for i in range(4)]

plt.figure(figsize=(10, 6))
plt.boxplot(data_box, labels=['A', 'B', 'C', 'D'])
plt.ylabel('Value', fontsize=12)
plt.xlabel('Category', fontsize=12)
plt.title('Box Plot Comparison', fontsize=14, fontweight='bold')
```

```
plt.grid(True, alpha=0.3, axis='y')
plt.savefig('boxplot.png', dpi=100, bbox_inches='tight')
print("Saved: boxplot.png")

# Bar chart from pandas
df_bar = pd.DataFrame({
    'Category': ['A', 'B', 'C', 'D'],
    'Value': [23, 45, 37, 52]
})

plt.figure(figsize=(8, 6))
plt.bar(df_bar['Category'], df_bar['Value'], color='steelblue',
edgecolor='black')
plt.xlabel('Category', fontsize=12)
plt.ylabel('Value', fontsize=12)
plt.title('Bar Chart', fontsize=14, fontweight='bold')
plt.grid(True, alpha=0.3, axis='y')
plt.savefig('barchart.png', dpi=100, bbox_inches='tight')
print("Saved: barchart.png")
```

```
Statistical Plots
======================================================================
Saved: histogram.png
Saved: boxplot.png
Saved: barchart.png
```

4.3 Multiple Subplots

```
# Create multiple subplots
print("Multiple Subplots")
print("="*70)

fig, axes = plt.subplots(2, 2, figsize=(12, 10))

# Subplot 1: Line plot
axes[0, 0].plot(x, np.sin(x), 'b-', linewidth=2)
axes[0, 0].set_title('Sine Wave')
axes[0, 0].set_xlabel('x')
axes[0, 0].set_ylabel('sin(x)')
axes[0, 0].grid(True, alpha=0.3)

# Subplot 2: Scatter
axes[0, 1].scatter(x_scatter, y_scatter, alpha=0.6, c='red')
axes[0, 1].set_title('Scatter Plot')
axes[0, 1].set_xlabel('X')
axes[0, 1].set_ylabel('Y')
axes[0, 1].grid(True, alpha=0.3)
```

```
# Subplot 3: Histogram
axes[1, 0].hist(data_hist, bins=30, edgecolor='black', alpha=0.7)
axes[1, 0].set_title('Histogram')
axes[1, 0].set_xlabel('Value')
axes[1, 0].set_ylabel('Frequency')
axes[1, 0].grid(True, alpha=0.3, axis='y')

# Subplot 4: Bar chart
axes[1, 1].bar(df_bar['Category'], df_bar['Value'], color='green', edgecolor='black')
axes[1, 1].set_title('Bar Chart')
axes[1, 1].set_xlabel('Category')
axes[1, 1].set_ylabel('Value')
axes[1, 1].grid(True, alpha=0.3, axis='y')

plt.tight_layout()
plt.savefig('subplots.png', dpi=100, bbox_inches='tight')
print("Saved: subplots.png")

print("\nPlot customization tips:")
print("  - figsize=(width, height) for size in inches")
print("  - alpha for transparency (0-1)")
print("  - linewidth, linestyle for line plots")
print("  - color, edgecolor for styling")
print("  - grid(True, alpha=0.3) for subtle gridlines")
print("  - tight_layout() prevents label overlap")
```

```
Multiple Subplots
======================================================================
Saved: subplots.png

Plot customization tips:
  - figsize=(width, height) for size in inches
  - alpha for transparency (0-1)
  - linewidth, linestyle for line plots
  - color, edgecolor for styling
  - grid(True, alpha=0.3) for subtle gridlines
  - tight_layout() prevents label overlap
```

5. Practical Example: Sales Analysis

```
# Complete analysis workflow
print("Sales Analysis Example")
print("="*70)

# Generate sample sales data
np.random.seed(42)
dates = pd.date_range('2024-01-01', periods=100, freq='D')
```

```
sales_df = pd.DataFrame({
    'Date': dates,
    'Product': np.random.choice(['A', 'B', 'C'], 100),
    'Region': np.random.choice(['North', 'South', 'East', 'West'], 100),
    'Sales': np.random.randint(100, 1000, 100),
    'Units': np.random.randint(1, 50, 100)
})

print("Sample sales data:")
print(sales_df.head(10))

# Analysis 1: Total sales by product
print("\nTotal sales by product:")
product_sales = sales_df.groupby('Product')['Sales'].sum().sort_values(ascending=False)
print(product_sales)

# Analysis 2: Average units by region
print("\nAverage units sold by region:")
region_units = sales_df.groupby('Region')['Units'].mean().round(2)
print(region_units)

# Analysis 3: Time series
sales_df['Month'] = sales_df['Date'].dt.to_period('M')
monthly_sales = sales_df.groupby('Month')['Sales'].sum()
print("\nMonthly sales:")
print(monthly_sales)

# Visualization
fig, axes = plt.subplots(2, 2, figsize=(14, 10))

# Product sales
product_sales.plot(kind='bar', ax=axes[0, 0], color='skyblue', edgecolor='black')
axes[0, 0].set_title('Total Sales by Product')
axes[0, 0].set_ylabel('Sales ($)')
axes[0, 0].grid(True, alpha=0.3, axis='y')

# Regional distribution
region_units.plot(kind='barh', ax=axes[0, 1], color='lightgreen', edgecolor='black')
axes[0, 1].set_title('Average Units by Region')
axes[0, 1].set_xlabel('Units')
axes[0, 1].grid(True, alpha=0.3, axis='x')

# Sales distribution
axes[1, 0].hist(sales_df['Sales'], bins=20, edgecolor='black', alpha=0.7)
axes[1, 0].set_title('Sales Distribution')
axes[1, 0].set_xlabel('Sales ($)')
```

```
axes[1, 0].set_ylabel('Frequency')
axes[1, 0].grid(True, alpha=0.3, axis='y')

# Time series
monthly_sales.plot(ax=axes[1, 1], marker='o', linewidth=2)
axes[1, 1].set_title('Monthly Sales Trend')
axes[1, 1].set_xlabel('Month')
axes[1, 1].set_ylabel('Sales ($)')
axes[1, 1].grid(True, alpha=0.3)
axes[1, 1].tick_params(axis='x', rotation=45)

plt.tight_layout()
plt.savefig('sales_analysis.png', dpi=100, bbox_inches='tight')
print("\nSaved: sales_analysis.png")
```

```
Sales Analysis Example
======================================================================
Sample sales data:
      Date Product Region  Sales  Units
0 2024-01-01       B   East    774     38
1 2024-01-02       C  North    611     14
2 2024-01-03       C  South    139      7
3 2024-01-04       B   West    292      3
4 2024-01-05       A  South    366     47
5 2024-01-06       C   West    940     39
6 2024-01-07       C   West    900     36
7 2024-01-08       A   East    186     13
8 2024-01-09       B  North    341     29
9 2024-01-10       C   East    232     37

Total sales by product:
Product
C    19847
B    18144
A    14463
Name: Sales, dtype: int64

Average units sold by region:
Region
East     22.96
North    22.25
South    23.12
West     23.88
Name: Units, dtype: float64

Monthly sales:
Month
2024-01    27035
```

```
2024-02    13853
2024-03    11566
Freq: M, Name: Sales, dtype: int64

Saved: sales_analysis.png
```

Summary

Key Takeaways:

• groupby() enables powerful aggregation and analysis

• merge() and concat() combine DataFrames like SQL joins

• apply() and map() transform data with custom functions

• pivot_table() reshapes data for different views

• Matplotlib provides comprehensive plotting capabilities

• Subplots enable multi-panel visualizations

• Combine pandas analysis with matplotlib visualization for EDA

Practice Exercises

1. Load a dataset and create summary statistics by category

2. Merge two DataFrames with different join types

3. Create a pivot table showing averages across multiple dimensions

4. Generate a 2x2 subplot showing: line plot, histogram, box plot, scatter plot

5. Perform complete EDA on a dataset: clean, analyze, visualize

6. Calculate correlations between numeric columns and visualize with heatmap

Next Steps

With strong pandas and visualization skills, you're ready to move into machine learning. Next week, we'll begin implementing machine learning algorithms from scratch, starting with linear regression and gradient descent.