GroupBy and Groupby Plotting

Got it! Let's focus on **concise, code-heavy examples** for **grouping and plotting** with `pandas` and `matplotlib`.

---

1. Grouping and Aggregation

1.1 Group by Single Column

```
# Group by 'Course' and calculate mean of 'Score'
df.groupby('Course')['Score'].mean()
```

1.2 Group by Multiple Columns

```
# Group by 'Course' and 'Gender' and calculate the mean of 'Score'
df.groupby(['Course', 'Gender'])['Score'].mean()
```

1.3 Multiple Aggregations

```
# Apply multiple aggregations (mean and sum) on 'Score' and 'Attendance'
df.groupby('Course').agg({'Score': ['mean', 'sum'], 'Attendance': 'mean'})
```

---

2. Applying Functions

2.1 Custom Function with `.apply()`

```
# Calculate range of 'Score' for each course
df.groupby('Course')['Score'].apply(lambda x: x.max() - x.min())
```

2.2 Adding Group-Level Information with `.transform()`

```
# Add average score by course to each row
df['Avg_Score_By_Course'] = df.groupby('Course')['Score'].transform('mean')
```

2.3 Filtering Groups with `.filter()`

```
# Keep only courses with average score > 85
df.groupby('Course').filter(lambda x: x['Score'].mean() > 85)
```

---

3. Plotting Grouped Data

3.1 Bar Plot: Total Scores by Course

```
import matplotlib.pyplot as plt

course_totals = df.groupby('Course')['Score'].sum()
course_totals.plot(kind='bar', color='skyblue', figsize=(8, 5))
plt.title('Total Scores by Course')
plt.ylabel('Total Score')
plt.grid(True)
plt.show()
```

3.2 Grouped Bar Plot with Seaborn

```
import seaborn as sns

sns.barplot(x='Course', y='Score', hue='Gender', data=df)
plt.title('Scores by Course and Gender')
plt.grid(True)
plt.show()
```

3.3 Box Plot: Distribution of Scores by Course

```
sns.boxplot(x='Course', y='Score', data=df)
plt.title('Score Distribution by Course')
plt.grid(True)
plt.show()
```

3.4 Line Plot: Average Score Over Time

```
df['Timestamp'] = pd.to_datetime(df['Timestamp'])
time_series = df.groupby(df['Timestamp'].dt.date)['Score'].mean()
time_series.plot(kind='line', marker='o', figsize=(10, 5))
```

```
plt.title('Average Score Over Time')
plt.grid(True)
plt.show()
```

---

4. Advanced Grouping with Pivot Tables

4.1 Creating a Pivot Table

```
# Average scores by Course and Gender
pivot_table = pd.pivot_table(df, values='Score', index='Course',
columns='Gender', aggfunc='mean')
print(pivot_table)
```

4.2 Heatmap of Correlation Matrix

```
import seaborn as sns

sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```

---

5. Extracting Insights from Grouped Data

5.1 Find Course with Highest Average Score

```
avg_scores = df.groupby('Course')['Score'].mean()
highest_avg_course = avg_scores.idxmax()
print(f"Highest Average Score: {highest_avg_course} - {avg_scores.max()}")
```

5.2 Find Most Common Gender in Course with Highest Depression

```
most_depressed_course = df.groupby('Course')['Depression'].sum().idxmax()
most_depressed_gender = df[df['Course'] ==
most_depressed_course].groupby('Gender')['Depression'].sum().idxmax()
```

```
print(f"Most Depressed Gender in {most_depressed_course}:
{most_depressed_gender}")
```

---

6. Groupby with Multiple Aggregations for Multiple Columns

```
df.groupby('Course').agg(
    Mean_Score=('Score', 'mean'),
    Max_Score=('Score', 'max'),
    Total_Attendance=('Attendance', 'sum')
)
```

Alright, let's go even deeper with more **concise, code-heavy examples** focusing on advanced `groupby` operations and visualizations.

---

1. Groupby with Advanced Aggregations

1.1 Aggregating Multiple Columns with Different Functions

```
# Group by 'Course' and apply different aggregations on multiple columns
df.groupby('Course').agg(
    Mean_Score=('Score', 'mean'),
    Max_Score=('Score', 'max'),
    Min_Score=('Score', 'min'),
    Total_Attendance=('Attendance', 'sum')
)
```

1.2 Aggregate with Named Aggregations

```
# Group by 'Gender' with multiple aggregations
df.groupby('Gender').agg(
    avg_score=('Score', 'mean'),
    total_score=('Score', 'sum'),
    count=('Score', 'count')
)
```

2. Using `.transform()` and `.apply()` for Advanced Data Manipulation

2.1 Using `.transform()` for Group-Level Calculations

```
# Add a column showing the percentage of each student's score relative to their course's total score
df['Score_Percentage'] = df['Score'] / df.groupby('Course')['Score'].transform('sum')
```

2.2 Applying Custom Functions with `.apply()`

```
# Apply a custom function to calculate a weighted average score by course
def weighted_avg(x):
    return np.average(x['Score'], weights=x['Attendance'])

df.groupby('Course').apply(weighted_avg)
```

3. Groupby with Filtering and Conditional Logic

3.1 Filtering Groups with `.filter()`

```
# Filter groups to include only courses where the total score is greater than 150
df.groupby('Course').filter(lambda x: x['Score'].sum() > 150)
```

3.2 Using `.query()` with Grouped Data

```
# Get courses where the average attendance is above 85
df.groupby('Course').mean().query('Attendance > 85')
```

4. Visualization Examples for Grouped Data

4.1 Stacked Bar Plot for Scores by Course and Gender

```
grouped = df.groupby(['Course', 'Gender'])['Score'].sum().unstack()
grouped.plot(kind='bar', stacked=True, figsize=(10, 6))
plt.title('Stacked Bar Plot of Scores by Course and Gender')
plt.ylabel('Total Score')
plt.grid(True)
plt.show()
```

4.2 Scatter Plot with Grouping

```
# Scatter plot of scores colored by course
colors = {'Data Science': 'blue', 'AI': 'green', 'Cyber Security': 'red'}
plt.scatter(df['Score'], df['Attendance'], c=df['Course'].map(colors),
alpha=0.7)
plt.xlabel('Score')
plt.ylabel('Attendance')
plt.title('Scatter Plot of Score vs Attendance by Course')
plt.show()
```

---

5. Time-Series Analysis Using `groupby()`

5.1 Grouping by Date and Calculating Weekly Averages

```
df['Timestamp'] = pd.to_datetime(df['Timestamp'])
df.set_index('Timestamp', inplace=True)

# Resample by week and calculate mean scores
weekly_avg = df['Score'].resample('W').mean()
weekly_avg.plot(kind='line', marker='o')
plt.title('Weekly Average Scores')
plt.grid(True)
plt.show()
```

5.2 Group by Month and Plot

```
monthly_avg = df.resample('M')['Score'].mean()
monthly_avg.plot(kind='bar', color='purple', figsize=(10, 6))
plt.title('Monthly Average Scores')
plt.ylabel('Average Score')
plt.grid(True)
plt.show()
```

6. Handling Missing Values with Grouping

6.1 Fill Missing Values with Group Averages

```
# Fill NaN scores with the mean score of each course
df['Score'] = df['Score'].fillna(df.groupby('Course')
['Score'].transform('mean'))
```

6.2 Interpolate Missing Values Within Groups

```
# Interpolate missing attendance values within each course group
df['Attendance'] = df.groupby('Course')['Attendance'].apply(lambda x:
x.interpolate())
```

7. Pivot Tables for Advanced Grouping

7.1 Creating a Pivot Table with Multiple Aggregations

```
pivot = pd.pivot_table(
    df,
    values='Score',
    index='Course',
    columns='Gender',
    aggfunc={'Score': ['mean', 'max', 'min']},
    fill_value=0
)
print(pivot)
```

7.2 Heatmap of Pivot Table

```
# Visualize pivot table data as a heatmap
sns.heatmap(pivot, annot=True, cmap='coolwarm')
```

```
plt.title('Pivot Table Heatmap')
plt.show()
```

8. Extracting Insights with `groupby()`

8.1 Finding the Course with the Highest Attendance

```
max_attendance_course = df.groupby('Course')['Attendance'].sum().idxmax()
print(f"Course with highest attendance: {max_attendance_course}")
```

8.2 Finding the Gender with the Highest Average Score in Each Course

```
df.groupby(['Course', 'Gender'])['Score'].mean().unstack().idxmax(axis=1)
```

9. Advanced Grouping with `pd.cut()` and Binning

9.1 Grouping Scores into Bins

```
# Bin scores into categories
df['Score_Bin'] = pd.cut(df['Score'], bins=[0, 60, 70, 80, 90, 100], labels=
['F', 'D', 'C', 'B', 'A'])
df.groupby('Score_Bin').size()
```

9.2 Visualizing Binned Data

```
df['Score_Bin'].value_counts().plot(kind='bar', color='orange')
plt.title('Distribution of Score Bins')
plt.xlabel('Score Bin')
plt.ylabel('Count')
plt.show()
```

Let's dive deeper into **plotting with `groupby`** using Pandas, Seaborn, and Matplotlib. We'll focus on different types of visualizations you can create using grouped data. This will help you analyze your data more effectively.

---

1. Bar Plots with Grouped Data

1.1 Grouped Bar Plot: Total Scores by Course

```python
import matplotlib.pyplot as plt

# Group by 'Course' and sum 'Score'
course_totals = df.groupby('Course')['Score'].sum()

# Create a bar plot
course_totals.plot(kind='bar', color='skyblue', figsize=(8, 5))
plt.title('Total Scores by Course')
plt.xlabel('Course')
plt.ylabel('Total Score')
plt.grid(True)
plt.show()
```

1.2 Grouped Bar Plot with Multiple Categories

```python
import seaborn as sns

# Bar plot of scores by 'Course' and 'Gender'
plt.figure(figsize=(10, 6))
sns.barplot(x='Course', y='Score', hue='Gender', data=df)
plt.title('Scores by Course and Gender')
plt.xlabel('Course')
plt.ylabel('Average Score')
plt.legend(title='Gender')
plt.grid(True)
plt.show()
```

---

2. Stacked Bar Plots

2.1 Stacked Bar Plot: Total Scores by Course and Gender

```
# Group by 'Course' and 'Gender' and sum 'Score'
grouped = df.groupby(['Course', 'Gender'])['Score'].sum().unstack()

# Create a stacked bar plot
grouped.plot(kind='bar', stacked=True, figsize=(10, 6), colormap='viridis')
plt.title('Total Scores by Course and Gender')
plt.xlabel('Course')
plt.ylabel('Total Score')
plt.grid(True)
plt.legend(title='Gender')
plt.show()
```

---

3. Line Plots with Grouped Data

3.1 Line Plot: Average Scores Over Time

```
df['Timestamp'] = pd.to_datetime(df['Timestamp'])

# Group by date and calculate mean scores
time_series = df.groupby(df['Timestamp'].dt.date)['Score'].mean()

# Plot the time series
plt.figure(figsize=(12, 6))
time_series.plot(kind='line', marker='o', color='blue')
plt.title('Average Scores Over Time')
plt.xlabel('Date')
plt.ylabel('Average Score')
plt.grid(True)
plt.show()
```

3.2 Line Plot by Group (Course)

```
# Group by 'Timestamp' and 'Course'
df.set_index('Timestamp', inplace=True)
df.groupby(['Course']).resample('M')
['Score'].mean().unstack().plot(kind='line', figsize=(12, 6))
plt.title('Monthly Average Scores by Course')
plt.xlabel('Month')
plt.ylabel('Average Score')
plt.grid(True)
plt.show()
```

4. Box Plots for Distribution Analysis

4.1 Box Plot: Distribution of Scores by Course

```
plt.figure(figsize=(10, 6))
sns.boxplot(x='Course', y='Score', data=df, palette='coolwarm')
plt.title('Score Distribution by Course')
plt.xlabel('Course')
plt.ylabel('Score')
plt.grid(True)
plt.show()
```

4.2 Box Plot by Course and Gender

```
plt.figure(figsize=(10, 6))
sns.boxplot(x='Course', y='Score', hue='Gender', data=df, palette='Set3')
plt.title('Score Distribution by Course and Gender')
plt.grid(True)
plt.show()
```

5. Heatmaps for Correlation Analysis

5.1 Correlation Matrix Heatmap

```
import seaborn as sns

# Compute the correlation matrix
corr_matrix = df.corr()

# Plot the heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Matrix Heatmap')
plt.show()
```

5.2 Heatmap of Grouped Data

```
# Pivot table for average scores by 'Course' and 'Gender'
pivot = df.pivot_table(values='Score', index='Course', columns='Gender',
aggfunc='mean')

# Heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(pivot, annot=True, cmap='YlGnBu')
plt.title('Average Score by Course and Gender')
plt.show()
```

---

6. Scatter Plots with Grouping

6.1 Scatter Plot: Score vs. Attendance Colored by Course

```
colors = {'Data Science': 'blue', 'AI': 'green', 'Cyber Security': 'red'}
plt.figure(figsize=(10, 6))
plt.scatter(df['Score'], df['Attendance'], c=df['Course'].map(colors),
alpha=0.7)
plt.title('Scatter Plot of Score vs Attendance')
plt.xlabel('Score')
plt.ylabel('Attendance')
plt.grid(True)
plt.show()
```

6.2 Scatter Plot with Regression Line

```
sns.lmplot(x='Score', y='Attendance', hue='Course', data=df, height=6,
aspect=1.5)
plt.title('Regression Plot of Score vs Attendance by Course')
plt.grid(True)
plt.show()
```

---

7. Pie Charts for Proportional Data

7.1 Pie Chart: Proportion of Students by Course

```
course_counts = df['Course'].value_counts()
plt.figure(figsize=(8, 8))
plt.pie(course_counts, labels=course_counts.index, autopct='%1.1f%%',
startangle=90)
plt.title('Proportion of Students by Course')
plt.show()
```

8. Histograms with Grouping

8.1 Histogram: Distribution of Scores

```
plt.figure(figsize=(10, 6))
plt.hist(df['Score'], bins=10, color='purple', edgecolor='black', alpha=0.7)
plt.title('Distribution of Scores')
plt.xlabel('Score')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
```

8.2 Histogram by Gender

```
plt.figure(figsize=(10, 6))
df[df['Gender'] == 'Male']['Score'].plot(kind='hist', bins=10, alpha=0.5,
label='Male')
df[df['Gender'] == 'Female']['Score'].plot(kind='hist', bins=10, alpha=0.5,
label='Female')
plt.title('Distribution of Scores by Gender')
plt.xlabel('Score')
plt.ylabel('Frequency')
plt.legend()
plt.grid(True)
plt.show()
```

9. Violin Plots for Distribution and Density

9.1 Violin Plot: Score Distribution by Course

```
plt.figure(figsize=(10, 6))
sns.violinplot(x='Course', y='Score', data=df, palette='cool')
plt.title('Score Distribution by Course')
plt.grid(True)
plt.show()
```

9.2 Violin Plot by Course and Gender

```
plt.figure(figsize=(10, 6))
sns.violinplot(x='Course', y='Score', hue='Gender', data=df, split=True)
plt.title('Score Distribution by Course and Gender')
plt.grid(True)
plt.show()
```

---

10. Pair Plot for Multivariate Analysis

10.1 Pair Plot of Numerical Columns

```
sns.pairplot(df, hue='Course', diag_kind='kde')
plt.suptitle('Pair Plot of Numerical Columns', y=1.02)
plt.show()
```
