DMV Practical1

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import json

# Load sales data from CSV file
try:
    csv = pd.read_csv("sales_data_sample.csv", encoding="cp1252")
except FileNotFoundError:
    print("CSV file not found. Please check the file path.")
except pd.errors.ParserError:
    print("Error parsing CSV file.")

# Load sales data from Excel file
try:
    ed = pd.read_excel("Sample-Sales-Data.xlsx")
except FileNotFoundError:
    print("Excel file not found. Please check the file path.")

# Load customer data from JSON file and convert to DataFrame
try:
    with open("customers.json", "r") as json_file:
        json_data = json.load(json_file)
    json_df = pd.DataFrame(json_data)
except FileNotFoundError:
    print("JSON file not found. Please check the file path.")
except json.JSONDecodeError:
    print("Error decoding JSON file.")

# Explore and clean the CSV data
print("CSV Data Info:")
csv.info()
print(csv.tail())
print(csv.describe())
csv = csv.dropna().drop_duplicates()

# Explore and clean the Excel data
print("Excel Data Info:")
ed.info()
print(ed.head())
print(ed.tail())
print(ed.describe())
ed = ed.dropna().drop_duplicates()
```

```
# Explore and clean the JSON data
print("JSON Data Info:")
json_df.info()
print(json_df.head())
json_df = json_df.dropna().drop_duplicates()

# Combine all three datasets into a unified DataFrame
# Note: Adjust the merge logic based on the common key if applicable
unified_data = pd.concat([csv, ed, json_df], ignore_index=True)

# Perform data analysis
# Calculate total sales
total_sales = unified_data['SALES'].sum()
print("Total Sales:", total_sales)

# Calculate average sales per order
average_sales_per_order = unified_data['SALES'].mean()
print("Average Sales per Order:", average_sales_per_order)

# Calculate sales by product category (example, adjust as needed)
# Assuming there is a column 'PRODUCTCATEGORY' in your dataset
if 'PRODUCTCATEGORY' in unified_data.columns:
    sales_by_category = unified_data.groupby('PRODUCTCATEGORY')['SALES'].sum()
    print("Sales by Category:")
    print(sales_by_category)

# Visualize sales by product category using a pie chart
if 'PRODUCTCATEGORY' in unified_data.columns:
    sales_by_category.plot(kind='pie', autopct='%1.1f%%')
    plt.title('Sales Distribution by Product Category')
    plt.ylabel('')
    plt.show()

# Calculate the distribution of sales and create a box plot
unified_data['SALES'].plot(kind='box')
plt.title('Sales Value Distribution')
plt.ylabel('Sales')
plt.show()

sales_mean = unified_data['SALES'].mean()
sales_median = unified_data['SALES'].median()

plt.hist(unified_data['SALES'], bins=20, edgecolor='black', alpha=0.7)
plt.axvline(sales_mean, color='r', linestyle='--', label=f'Mean: {sales_mean:.2f}')
plt.axvline(sales_median, color='g', linestyle='--', label=f'Median: {sales_median:.2f}')
plt.title('Sales Value Distribution')
plt.xlabel('Sales Value')
plt.ylabel('Frequency')
plt.legend()
plt.grid(axis='y', alpha=0.5)
```

plt.show()
Explain this full code, line by line in detail

Certainly! Here's a detailed line-by-line explanation of this code:

Importing Libraries

python

```python
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import json
```

- **numpy**: A library for numerical operations in Python, though it is not used here directly.
- **pandas**: A powerful library for data manipulation and analysis.
- **matplotlib.pyplot**: A plotting library for data visualization.
- **json**: A module for working with JSON data.

Loading Sales Data from CSV

python

```python
# Load sales data from CSV file
try:
    csv = pd.read_csv("sales_data_sample.csv", encoding="cp1252")
except FileNotFoundError:
    print("CSV file not found. Please check the file path.")
except pd.errors.ParserError:
    print("Error parsing CSV file.")
```

- This block attempts to read a CSV file named `` `sales_data_sample.csv` `` using `` `pd.read_csv()` ``.
  - **encoding="cp1252"**: Specifies character encoding to handle special characters.
  - **Error handling**:
    - **FileNotFoundError**: Prints a message if the file is missing.
    - **pd.errors.ParserError**: Prints a message if there’s an issue parsing the file.

Loading Sales Data from Excel

python

```python
# Load sales data from Excel file
try:
    ed = pd.read_excel("Sample-Sales-Data.xlsx")
except FileNotFoundError:
    print("Excel file not found. Please check the file path.")
```

- Attempts to load an Excel file named `` `Sample-Sales-Data.xlsx` `` using `` `pd.read_excel()` ``.
    - **FileNotFoundError**: Catches and prints an error message if the file is not found.

Loading Customer Data from JSON

```python
# Load customer data from JSON file and convert to DataFrame
try:
    with open("customers.json", "r") as json_file:
        json_data = json.load(json_file)
    json_df = pd.DataFrame(json_data)
except FileNotFoundError:
    print("JSON file not found. Please check the file path.")
except json.JSONDecodeError:
    print("Error decoding JSON file.")
```

- Loads customer data from a JSON file named `` `customers.json` ``.
    - **with open("customers.json", "r")**: Opens the JSON file in read mode.
    - **json.load(json_file)**: Reads the JSON data.
    - **pd.DataFrame(json_data)**: Converts the JSON data into a pandas DataFrame.
    - **Error handling**:
        - **FileNotFoundError**: Catches if the JSON file is missing.
        - **json.JSONDecodeError**: Catches if the file is not a valid JSON.

Exploring and Cleaning the CSV Data

```python
# Explore and clean the CSV data
print("CSV Data Info:")
csv.info()
print(csv.tail())
print(csv.describe())
csv = csv.dropna().drop_duplicates()
```

- Provides insights into the structure and statistics of the CSV data:
    - **csv.info()**: Shows column names, non-null counts, and data types.
    - **csv.tail()**: Displays the last few rows.
    - **csv.describe()**: Provides statistical summary of numerical columns.
    - **csv.dropna().drop_duplicates()**: Cleans the data by removing any missing values (`` `dropna()` ``) and duplicate rows (`` `drop_duplicates()` ``).

Exploring and Cleaning the Excel Data

```python
# Explore and clean the Excel data
print("Excel Data Info:")
ed.info()
print(ed.head())
print(ed.tail())
print(ed.describe())
ed = ed.dropna().drop_duplicates()
```

- Similar to the CSV data, this block examines and cleans the Excel data:
    - **ed.info(), ed.head(), ed.tail(), ed.describe()**: Similar functionality as above for inspecting the data.
    - **ed.dropna().drop_duplicates()**: Removes missing and duplicate rows.

Exploring and Cleaning the JSON Data

```python
# Explore and clean the JSON data
print("JSON Data Info:")
json_df.info()
print(json_df.head())
json_df = json_df.dropna().drop_duplicates()
```

- Performs the same examination and cleaning operations on the JSON data.
    - **json_df.info()**: Displays column details and non-null counts.
    - **json_df.head()**: Displays the first few rows.
    - **json_df.dropna().drop_duplicates()**: Removes any missing values and duplicate rows.

Combining All Datasets

```python
# Combine all three datasets into a unified DataFrame
# Note: Adjust the merge logic based on the common key if applicable
unified_data = pd.concat([csv, ed, json_df], ignore_index=True)
```

- **pd.concat()**: Combines `` `csv` ``, `` `ed` ``, and `` `json_df` `` DataFrames into a single DataFrame called `` `unified_data` ``.
    - **ignore_index=True**: Resets the index for the combined data.

Note: This simple concatenation assumes there is no specific key for merging; modifications may be needed if there's a shared column.

Calculating Total and Average Sales

```python
# Perform data analysis
# Calculate total sales
total_sales = unified_data['SALES'].sum()
print("Total Sales:", total_sales)

# Calculate average sales per order
```

```
average_sales_per_order = unified_data['SALES'].mean()
print("Average Sales per Order:", average_sales_per_order)
```

- **total_sales**: Calculates the total sales by summing the `SALES` column.
- **average_sales_per_order**: Calculates the average by taking the mean of the `SALES` column.
- Prints both values.

Calculating and Visualizing Sales by Product Category

```python
# Calculate sales by product category (example, adjust as needed)
# Assuming there is a column 'PRODUCTCATEGORY' in your dataset
if 'PRODUCTCATEGORY' in unified_data.columns:
    sales_by_category = unified_data.groupby('PRODUCTCATEGORY')['SALES'].sum()
    print("Sales by Category:")
    print(sales_by_category)
```

- Checks if `PRODUCTCATEGORY` is a column in `unified_data`.
    - **groupby('PRODUCTCATEGORY')['SALES'].sum()**: Groups data by product category and calculates the total sales per category.
- **sales_by_category**: Stores the grouped sales data and prints it.

```python
# Visualize sales by product category using a pie chart
if 'PRODUCTCATEGORY' in unified_data.columns:
    sales_by_category.plot(kind='pie', autopct='%1.1f%%')
    plt.title('Sales Distribution by Product Category')
    plt.ylabel('')
    plt.show()
```

- **sales_by_category.plot(kind='pie', autopct='%1.1f%%')**: Creates a pie chart.
    - **autopct='%1.1f%%'**: Shows each slice's percentage to one decimal.
    - **plt.show()**: Displays the pie chart.

Creating a Box Plot of Sales Distribution

```python
# Calculate the distribution of sales and create a box plot
unified_data['SALES'].plot(kind='box')
plt.title('Sales Value Distribution')
plt.ylabel('Sales')
plt.show()
```

- **unified_data['SALES'].plot(kind='box')**: Plots a box plot of `SALES` values.
- **plt.title() and plt.ylabel()**: Adds title and labels.
- **plt.show()**: Displays the box plot.

Creating a Histogram with Mean and Median Lines

```python
sales_mean = unified_data['SALES'].mean()
sales_median = unified_data['SALES'].median()

plt.hist(unified_data['SALES'], bins=20, edgecolor='black', alpha=0.7)
plt.axvline(sales_mean, color='r', linestyle='--', label=f'Mean: {sales_mean:.2f}')
plt.axvline(sales_median, color='g', linestyle='--', label=f'Median: {sales_median:.2f}')
plt.title('Sales Value Distribution')
plt.xlabel('Sales Value')
plt.ylabel('Frequency')
plt.legend()
plt.grid(axis='y', alpha=0.5)
plt.show()
```

- **plt.hist()**: Plots a histogram for the distribution of `` `SALES` ``.
    - **bins=20**: Sets 20 bins for the histogram.
    - **edgecolor='black' and alpha=0.7**: Style adjustments.
- **plt.axvline()**: Adds vertical lines for mean and median.
    - **label=f'Mean: {sales_mean:.2f}'**: Adds label to the line for legend.
- **plt.legend() and plt.grid()**: Adds legend and grid for clarity.
- **plt.show()**: Displays the histogram.

Explain all the graphs in detail in this practical.

Certainly! Here’s a detailed explanation of each graph used in this code:

1. Pie Chart for Sales by Product Category

```python
if 'PRODUCTCATEGORY' in unified_data.columns:
    sales_by_category.plot(kind='pie', autopct='%1.1f%%')
    plt.title('Sales Distribution by Product Category')
    plt.ylabel('')
    plt.show()
```

Explanation:

- **Purpose**: The pie chart shows the proportion of total sales generated by each product category.
- **Details**:
    - `` `kind='pie'` ``: Specifies a pie chart, which is ideal for visualizing proportions of a whole.
    - `` `autopct='%1.1f%%'` ``: Displays the percentage of each slice with one decimal place.
    - **Title**: "Sales Distribution by Product Category" helps understand what the chart represents.
    - **Blank `` `ylabel` ``**: Hides the default y-axis label, which is unnecessary for a pie chart.

- **Use Case**: Useful for understanding which product categories are the most or least popular based on sales. This can inform product focus areas or marketing strategies.

---

2. Box Plot of Sales Value Distribution

```python
unified_data['SALES'].plot(kind='box')
plt.title('Sales Value Distribution')
plt.ylabel('Sales')
plt.show()
```

Explanation:

- **Purpose**: The box plot visualizes the spread and distribution of sales values, including key summary statistics such as the median, quartiles, and potential outliers.
- **Details**:
    - `` `kind='box'` ``: Specifies a box plot, which is helpful in detecting data spread and outliers.
    - **Box Plot Components**:
        - **Median**: The line inside the box represents the median (50th percentile) of sales values.
        - **Quartiles**: The edges of the box represent the 25th (Q1) and 75th (Q3) percentiles.
        - **Whiskers**: Lines extending from the box indicate the range within 1.5 times the interquartile range (IQR). Beyond these, data points are considered outliers.
        - **Outliers**: Any points beyond the whiskers are outliers, indicating unusually high or low sales values.
    - **Title and y-axis label**: Indicate that this plot shows sales distribution.
- **Use Case**: Helps in understanding the typical range of sales values and identifying unusual data points. This insight can reveal trends like occasional high-value sales or potential data errors.

---

3. Histogram of Sales Value Distribution with Mean and Median Lines

```python
sales_mean = unified_data['SALES'].mean()
sales_median = unified_data['SALES'].median()

plt.hist(unified_data['SALES'], bins=20, edgecolor='black', alpha=0.7)
plt.axvline(sales_mean, color='r', linestyle='--', label=f'Mean: {sales_mean:.2f}')
plt.axvline(sales_median, color='g', linestyle='--', label=f'Median: {sales_median:.2f}')
plt.title('Sales Value Distribution')
plt.xlabel('Sales Value')
plt.ylabel('Frequency')
plt.legend()
plt.grid(axis='y', alpha=0.5)
plt.show()
```

Explanation:

- **Purpose**: This histogram shows the distribution of sales values, highlighting how frequently different sales amounts occur. The mean and median lines provide additional insight into the central tendency.

- **Details**:
    - **Histogram**:
        - `` `plt.hist()` ``: Creates a histogram with 20 bins, which groups sales values into ranges, or "bins."
        - `` `edgecolor='black'` `` **and** `` `alpha=0.7` ``: Add a black edge to each bin for clarity and set transparency to 70%.
        - **X-axis (`` `Sales Value` ``)**: Represents the range of sales amounts.
        - **Y-axis (`` `Frequency` ``)**: Shows the count of sales within each sales range.
    - **Vertical Lines for Mean and Median**:
        - **Mean (Red Line)**: Calculated by averaging all sales values and represented by a red dashed line.
        - **Median (Green Line)**: Middle value of sales data, represented by a green dashed line.
    - **Title and Labels**: Clearly describe the chart, with a legend indicating the mean and median lines.
- **Interpretation**:
    - The shape of the histogram reveals if the sales data is skewed (asymmetric) or normally distributed.
    - Mean and median lines help identify any skewness. For instance:
        - **If mean > median**: The distribution is likely right-skewed (tail on the right), indicating a few high-value sales.
        - **If mean < median**: Left-skewed (tail on the left).
        - **If mean ≈ median**: The distribution is closer to symmetric or normal.
- **Use Case**: Useful in understanding sales distribution patterns, identifying skewness, and visualizing data clustering around specific sales values. This can be particularly valuable for pricing strategies or sales analysis.

```
# Visualize sales by product category using a pie chart
if 'PRODUCTCATEGORY' in unified_data.columns:
    sales_by_category.plot(kind='pie', autopct='%1.1f%%')
    plt.title('Sales Distribution by Product Category')
    plt.ylabel('')
    plt.show()
```

I cannot see this pie chart in code?

The code you've shared should generate a pie chart when executed, provided that the `` `PRODUCTCATEGORY` `` column exists in the `` `unified_data` `` DataFrame and contains categorical data that can be grouped by sales.

Here's how to troubleshoot and ensure the pie chart displays correctly:

1. **Check Column Existence**:
    - Ensure the `` `PRODUCTCATEGORY` `` column is present in your dataset. You can verify this by printing the `` `unified_data.columns` `` list to see if `` `PRODUCTCATEGORY` `` is included.

```python
print(unified_data.columns)
```

2. **Group Data by Category**:
   - Make sure to group the data by `PRODUCTCATEGORY` and sum the `SALES` values for each category before creating the pie chart.

```python
if 'PRODUCTCATEGORY' in unified_data.columns and 'SALES' in unified_data.columns:
    sales_by_category = unified_data.groupby('PRODUCTCATEGORY')['SALES'].sum()
    sales_by_category.plot(kind='pie', autopct='%1.1f%%')
    plt.title('Sales Distribution by Product Category')
    plt.ylabel('')  # Optional, hides the default ylabel
    plt.show()
```

3. **Ensure Data Completeness**:
   - If `unified_data['SALES']` contains `NaN` values or non-numeric entries, the pie chart may fail to render properly. You can clean the data by ensuring only non-null, numeric values are used:

```python
unified_data = unified_data.dropna(subset=['PRODUCTCATEGORY', 'SALES'])
unified_data['SALES'] = pd.to_numeric(unified_data['SALES'], errors='coerce')
```

4. **Matplotlib Backend**:
   - In some environments (like Jupyter notebooks), you may need to explicitly set up `matplotlib` to display plots. If you're using Jupyter, add `%matplotlib inline` at the beginning of the code.
5. **Testing with Sample Data**:
   - If you're unable to see the pie chart, try creating a small sample DataFrame to ensure the code is functioning:

```python
sample_data = pd.DataFrame({
    'PRODUCTCATEGORY': ['Category A', 'Category B', 'Category C'],
    'SALES': [500, 300, 200]
})
sales_by_category = sample_data.groupby('PRODUCTCATEGORY')['SALES'].sum()
sales_by_category.plot(kind='pie', autopct='%1.1f%%')
plt.title('Sales Distribution by Product Category')
plt.ylabel('')
plt.show()
```

This will confirm whether the plotting code is working independently of your main dataset. Let me know if this helps!

it is in if blokk is that why?

Yes, exactly! The pie chart code is within an `if` block, which means it will only execute if the condition `if 'PRODUCTCATEGORY' in unified_data.columns` is true. This condition checks whether the `PRODUCTCATEGORY` column is present in the `unified_data` DataFrame.

Why This Might Cause the Pie Chart Not to Appear

If `PRODUCTCATEGORY` does not exist in `unified_data`, then the entire block will be skipped, and the pie chart won't be generated or displayed. Here's how you can check if this is the issue:

1. **Check for `PRODUCTCATEGORY` Column**:
    - Run this line to see if `PRODUCTCATEGORY` is in your columns:

```python
print(unified_data.columns)
```

2. **Modify Code for Debugging**:
    - You can temporarily add a message inside the `if` block to confirm if it's being executed.

```python
if 'PRODUCTCATEGORY' in unified_data.columns:
    print("PRODUCTCATEGORY column found, generating pie chart...")
    sales_by_category = unified_data.groupby('PRODUCTCATEGORY')['SALES'].sum()
    sales_by_category.plot(kind='pie', autopct='%1.1f%%')
    plt.title('Sales Distribution by Product Category')
    plt.ylabel('')
    plt.show()
else:
    print("PRODUCTCATEGORY column not found in unified_data.")
```

If the message `PRODUCTCATEGORY column not found in unified_data.` appears, then that column is missing, and the code inside the `if` block isn't running.

Solution

If `PRODUCTCATEGORY` is missing and you still need to visualize sales by product category, you may need to adjust your code or the data loading process to ensure this column is present.