Data Analysis Project with Python and Pandas

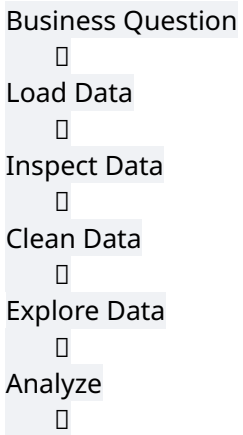
1. Project Overview

Learning pandas methods individually is useful, but real data analysis requires connecting those methods into a complete workflow.

In this project, you will work with a fictional **online store sales dataset** and answer practical business questions such as:

- How much revenue did the store generate?
- Which products sold the most?
- Which category generated the most revenue?
- Which city generated the most sales?
- What are the monthly sales trends?
- Who are the highest-value customers?
- Which products have strong sales volume but low revenue?
- Which payment methods are most commonly used?
- What recommendations can be made from the data?

The project follows a realistic data-analysis workflow:



Visualize

Find Insights

Communicate Results

This reflects the way pandas is intended to support practical data analysis: importing, preparing, analyzing, combining, and visualizing tabular data.

---

2. What You Will Build

By the end of this project, you will have a Python analysis script or Jupyter Notebook that:

1. Loads a sales dataset.
2. Inspects its structure.
3. Cleans inconsistent data.
4. Handles missing values.
5. Removes duplicates.
6. Converts data types.
7. Creates useful calculated columns.
8. Calculates business metrics.
9. Groups and aggregates data.
10. Analyzes customers, products, cities, and categories.
11. Performs time-based analysis.
12. Creates charts.
13. Identifies important patterns.
14. Produces a final business summary.

---

3. Skills Practiced

This project combines concepts from the previous Python and pandas lessons:

- Python variables
- Functions
- Lists and dictionaries
- NumPy
- pandas
- DataFrames
- CSV files
- Data cleaning
- Missing values
- Duplicate detection
- Data type conversion
- Filtering
- Sorting
- GroupBy
- Aggregation
- `merge()`
- Date handling
- Statistical summaries
- Data visualization
- Basic business analysis

The pandas documentation specifically covers selection, creating derived columns, summary statistics, combining tables, time-series handling, and plotting as core analysis workflows.

---

4. Project Structure

A simple version:

```
sales_analysis/
│
├── data/
│   └── sales.csv
│
├── analysis.py
│
├── charts/
│
└── README.md
```

For a notebook-based project:

```
sales_analysis/
│
├── data/
│   └── sales.csv
│
├── sales_analysis.ipynb
│
├── charts/
│
└── README.md
```

---

5. Install the Required Libraries

Install pandas:

```
pip install pandas
```

Install NumPy:

```
pip install numpy
```

Install Matplotlib:

```
pip install matplotlib
```

Or:

```
pip install pandas numpy matplotlib
```

---

6. Import the Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

---

7. The Dataset

Create a file:

```
sales.csv
```

Use columns such as:

```
order_id
order_date
```

customer_id
customer_name
city
category
product
quantity
unit_price
discount
payment_method

Example:

```
order_id,order_date,customer_id,customer_name,city,category,product,quantity,unit_price,discount,payment_method
1001,2026-01-05,C001,Hafsa,Lahore,Electronics,Keyboard,2,3500,0.05,Card
1002,2026-01-07,C002,Asim,Islamabad,Electronics,Mouse,3,1800,0.00,Cash
1003,2026-01-10,C003,Ali,Karachi,Accessories,Headphones,1,5000,0.10,Card
1004,2026-01-12,C001,Hafsa,Lahore,Electronics,Monitor,1,28000,0.05,Bank Transfer
1005,2026-01-15,C004,Sara,Lahore,Accessories,USB Cable,5,800,0.00,Cash
```

You can expand this dataset to hundreds or thousands of rows for a more realistic project.

---

8. Load the Data

```
df = pd.read_csv("data/sales.csv")
```

Immediately inspect it:

```
print(df.head())
```

---

9. Understand the Dataset

Check its size:

`df.shape`

Check columns:

`df.columns`

Check data types:

`df.dtypes`

Get an overview:

`df.info()`

Statistical summary:

`df.describe()`

These are core pandas inspection operations for understanding a DataFrame before analysis.

---

10. First Business Question

Before writing analysis code, define the question.

Instead of:

> "Let's run some pandas commands."

Start with:

> "What do I want to learn from this data?"

For this project:

```
How is the online store performing,
and what patterns can we discover
about products, customers, locations,
and sales over time?
```

This difference is important.

Data analysis is question-driven, not method-driven.

---

11. Inspect Missing Values

```
df.isna().sum()
```

Calculate percentages:

```
df.isna().mean() * 100
```

If a required field is missing, investigate it before deciding what to do.

---

12. Check Duplicate Orders

```
df.duplicated().sum()
```

Check duplicate order IDs:

```
df["order_id"].duplicated().sum()
```

If every order ID should be unique:

```
duplicates = df[
    df["order_id"].duplicated(keep=False)
]

print(duplicates)
```

---

13. Clean Column Names

```
df.columns = (
    df.columns
    .str.strip()
    .str.lower()
    .str.replace(" ", "_")
)
```

Now your columns follow a predictable naming style.

---

14. Clean Text Columns

```
text_columns = [
    "customer_name",
    "city",
```

```
    "category",
    "product",
    "payment_method"
]

for column in text_columns:
    df[column] = (
        df[column]
        .astype("string")
        .str.strip()
    )
```

Standardize categories:

```
df["city"] = df["city"].str.title()
df["category"] = df["category"].str.title()
df["payment_method"] = df["payment_method"].str.title()
```

---

15. Convert Numeric Columns

```
df["quantity"] = pd.to_numeric(
    df["quantity"],
    errors="coerce"
)

df["unit_price"] = pd.to_numeric(
    df["unit_price"],
    errors="coerce"
)

df["discount"] = pd.to_numeric(
    df["discount"],
```

```
    errors="coerce"
)
```

---

16. Convert the Date

```
df["order_date"] = pd.to_datetime(
    df["order_date"],
    errors="coerce"
)
```

Now pandas can perform time-based analysis.

---

17. Check the Cleaned Dataset

```
print(df.info())
print(df.isna().sum())
print(df.duplicated().sum())
```

Validate important numeric fields:

```
print(df[df["quantity"] <= 0])
print(df[df["unit_price"] < 0])
print(df[df["discount"] < 0])
print(df[df["discount"] > 1])
```

---

18. Create the Revenue Column

This is one of the most important calculations in the project.

First calculate the original value:

```
df["gross_sales"] = (
    df["quantity"] *
    df["unit_price"]
)
```

Then calculate the discount amount:

```
df["discount_amount"] = (
    df["gross_sales"] *
    df["discount"]
)
```

Finally:

```
df["revenue"] = (
    df["gross_sales"] -
    df["discount_amount"]
)
```

Or in one expression:

```
df["revenue"] = (
    df["quantity"] *
    df["unit_price"] *
    (1 - df["discount"])
)
```

---

19. First KPI: Total Revenue

```
total_revenue = df["revenue"].sum()
```

```
print(total_revenue)
```

This answers:

> How much revenue did the store generate?

---

20. Total Orders

```
total_orders = df["order_id"].nunique()

print(total_orders)
```

Do not automatically use:

```
len(df)
```

because one order could theoretically contain multiple rows.

---

21. Total Units Sold

```
total_units = df["quantity"].sum()

print(total_units)
```

This answers:

>

> How many individual products were sold?

---

22. Average Order Value

A useful business metric is:

```
Average Order Value =
Total Revenue / Number of Orders
```

In Python:

```
average_order_value = (
    total_revenue /
    total_orders
)

print(average_order_value)
```

---

23. Number of Customers

```
total_customers = df["customer_id"].nunique()

print(total_customers)
```

This tells you how many unique customers are represented.

---

24. Basic Dataset Summary

Create one summary:

```
summary = {
    "Total Revenue": total_revenue,
    "Total Orders": total_orders,
    "Total Units": total_units,
    "Customers": total_customers,
    "Average Order Value": average_order_value
}

for key, value in summary.items():
    print(f"{key}: {value:,.2f}")
```

---

25. Product Analysis

Ask:

> Which products generate the most revenue?

```
product_revenue = (
    df.groupby("product")["revenue"]
    .sum()
    .sort_values(ascending=False)
)

print(product_revenue)
```

Pandas `groupby()` follows the split-apply-combine model: data is divided into groups, calculations are applied to each group, and the results are combined.

---

26. Top 10 Products

```
top_products = (
    product_revenue
    .head(10)
)

print(top_products)
```

---

27. Product Units Sold

Revenue is not the only useful metric.

Calculate quantity:

```
product_units = (
    df.groupby("product")["quantity"]
    .sum()
    .sort_values(ascending=False)
)

print(product_units)
```

Now you can compare:

`Product performance`

□□□ Revenue
□□□ Units sold

---

28. Top Products by Multiple Metrics

```
product_summary = (
    df.groupby("product")
    .agg(
        revenue=("revenue", "sum"),
        units_sold=("quantity", "sum"),
        orders=("order_id", "nunique"),
        average_price=("unit_price", "mean")
    )
    .sort_values("revenue", ascending=False)
)

print(product_summary)
```

This gives a much more useful view than looking at one metric.

---

29. Category Analysis

Calculate revenue by category:

```
category_revenue = (
    df.groupby("category")["revenue"]
    .sum()
    .sort_values(ascending=False)
)
```

```
print(category_revenue)
```

Calculate units:

```
category_units = (
    df.groupby("category")["quantity"]
    .sum()
    .sort_values(ascending=False)
)

print(category_units)
```

---

30. Category Performance Table

```
category_summary = (
    df.groupby("category")
    .agg(
        revenue=("revenue", "sum"),
        units=("quantity", "sum"),
        orders=("order_id", "nunique"),
        customers=("customer_id", "nunique")
    )
    .sort_values("revenue", ascending=False)
)

print(category_summary)
```

This allows you to compare several dimensions simultaneously.

---

31. City Analysis

Question:

> Which cities generate the most revenue?

```
city_revenue = (
    df.groupby("city")["revenue"]
    .sum()
    .sort_values(ascending=False)
)

print(city_revenue)
```

Orders by city:

```
city_orders = (
    df.groupby("city")["order_id"]
    .nunique()
    .sort_values(ascending=False)
)

print(city_orders)
```

---

32. Customer Analysis

Calculate customer revenue:

```
customer_revenue = (
    df.groupby(
        ["customer_id", "customer_name"]
    )["revenue"]
```

```
    .sum()
    .sort_values(ascending=False)
)

print(customer_revenue)
```

Top customers:

```
print(customer_revenue.head(10))
```

---

33. Customer Purchase Frequency

```
customer_orders = (
    df.groupby("customer_id")["order_id"]
    .nunique()
    .sort_values(ascending=False)
)

print(customer_orders.head(10))
```

This answers:

> Which customers place orders most frequently?

---

34. Customer Summary

```
customer_summary = (
    df.groupby(
```

```
        ["customer_id", "customer_name"]
    )
    .agg(
        revenue=("revenue", "sum"),
        orders=("order_id", "nunique"),
        units=("quantity", "sum")
    )
    .sort_values("revenue", ascending=False)
)

print(customer_summary.head(10))
```

---

35. Payment Method Analysis

```
payment_summary = (
    df.groupby("payment_method")
    .agg(
        orders=("order_id", "nunique"),
        revenue=("revenue", "sum")
    )
    .sort_values("revenue", ascending=False)
)

print(payment_summary)
```

This can answer:

> Which payment methods are most frequently used?

and:

>

> Which payment methods are associated with the most revenue?

---

36. Monthly Revenue

Create a month column:

```
df["month"] = (
    df["order_date"]
    .dt.to_period("M")
)
```

Then:

```
monthly_revenue = (
    df.groupby("month")["revenue"]
    .sum()
)

print(monthly_revenue)
```

This helps identify trends over time.

---

37. Monthly Orders

```
monthly_orders = (
    df.groupby("month")["order_id"]
    .nunique()
)
```

```
print(monthly_orders)
```

Now compare:

```
Month
□□□ Revenue
□□□ Orders
```

---

38. Revenue by Day

```
daily_revenue = (
    df.groupby("order_date")["revenue"]
    .sum()
)

print(daily_revenue)
```

This can reveal spikes or unusual sales days.

---

39. Weekday Analysis

Create:

```
df["weekday"] = (
    df["order_date"]
    .dt.day_name()
)
```

Then:

```
weekday_revenue = (
    df.groupby("weekday")["revenue"]
    .sum()
)
```

To put weekdays in the correct order:

```
weekday_order = [
    "Monday",
    "Tuesday",
    "Wednesday",
    "Thursday",
    "Friday",
    "Saturday",
    "Sunday"
]

weekday_revenue = weekday_revenue.reindex(
    weekday_order
)
```

---

40. Discount Analysis

Calculate average discount:

```
average_discount = df["discount"].mean()

print(average_discount)
```

Compare revenue for discounted and non-discounted orders:

```
df["has_discount"] = df["discount"] > 0
```

Then:

```
discount_summary = (
    df.groupby("has_discount")
    .agg(
        orders=("order_id", "nunique"),
        revenue=("revenue", "sum"),
        average_order_value=("revenue", "mean")
    )
)

print(discount_summary)
```

Remember:

> A correlation or difference does not automatically prove that discounts caused a revenue change.

---

41. Revenue by Category and City

Group by multiple columns:

```
category_city = (
    df.groupby(["category", "city"])["revenue"]
    .sum()
    .sort_values(ascending=False)
)

print(category_city)
```

This can reveal combinations such as:

```
Electronics + Lahore
Accessories + Karachi
Electronics + Islamabad
```

---

42. Pivot Tables

A pivot table can make multi-dimensional analysis easier.

```
pivot = pd.pivot_table(
    df,
    values="revenue",
    index="city",
    columns="category",
    aggfunc="sum",
    fill_value=0
)

print(pivot)
```

Now you can compare categories across cities.

---

43. Finding the Best City for Each Category

```
city_category = (
    df.groupby(
        ["city", "category"]
    )["revenue"]
    .sum()
    .reset_index()
)
```

```
print(city_category)
```

Sort:

```
city_category = city_category.sort_values(
    "revenue",
    ascending=False
)
```

---

44. Analyze Product Pricing

Calculate average price:

```
product_price = (
    df.groupby("product")["unit_price"]
    .mean()
    .sort_values(ascending=False)
)

print(product_price)
```

Compare price and quantity:

```
product_analysis = (
    df.groupby("product")
    .agg(
        average_price=("unit_price", "mean"),
        units_sold=("quantity", "sum"),
        revenue=("revenue", "sum")
    )
)
```

```
print(product_analysis)
```

This lets you investigate whether expensive products or high-volume products contribute more revenue.

---

45. Correlation

You can inspect relationships between numeric columns:

```
numeric_columns = [
    "quantity",
    "unit_price",
    "discount",
    "revenue"
]

print(
    df[numeric_columns].corr()
)
```

Correlation can help identify relationships worth investigating.

But:

> Correlation does not establish causation.

---

46. Visualization

Numbers are useful, but charts can reveal patterns quickly.

Pandas provides plotting methods for Series and DataFrames, with Matplotlib used as the default plotting backend.

Import Matplotlib:

```
import matplotlib.pyplot as plt
```

---

47. Revenue by Category Chart

```
category_revenue.plot(
    kind="bar"
)

plt.title("Revenue by Category")
plt.xlabel("Category")
plt.ylabel("Revenue")
plt.tight_layout()
plt.show()
```

---

48. Monthly Revenue Chart

```
monthly_revenue.plot(
    kind="line",
    marker="o"
)

plt.title("Monthly Revenue")
plt.xlabel("Month")
plt.ylabel("Revenue")
```

```
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

---

49. Top Products Chart

```
top_products.sort_values().plot(
    kind="barh"
)

plt.title("Top Products by Revenue")
plt.xlabel("Revenue")
plt.ylabel("Product")
plt.tight_layout()
plt.show()
```

---

50. Revenue by City

```
city_revenue.plot(
    kind="bar"
)

plt.title("Revenue by City")
plt.xlabel("City")
plt.ylabel("Revenue")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

---

51. Quantity vs Revenue

A scatter plot can help compare product volume and revenue.

```
product_analysis.plot(
    kind="scatter",
    x="units_sold",
    y="revenue"
)

plt.title("Units Sold vs Revenue")
plt.xlabel("Units Sold")
plt.ylabel("Revenue")
plt.tight_layout()
plt.show()
```

Pandas supports several standard plot types, including line, bar, histogram, box, scatter, area, and others.

---

52. Save Charts

Create a chart:

```
monthly_revenue.plot(
    kind="line",
    marker="o"
)

plt.title("Monthly Revenue")
plt.tight_layout()
plt.savefig(
    "charts/monthly_revenue.png"
```

)

plt.show()

A good project should save important visualizations rather than only displaying them.

---

53. Ask Better Questions

A beginner might ask:

> What is the total revenue?

A stronger analyst asks:

> How did revenue change over time?

An even stronger analysis asks:

> Which products, categories, cities, and customer groups contributed to those changes?

The goal is to move from:

Numbers

to:

`Patterns`

to:

`Insights`

---

54. Example Insight

Suppose your analysis produces:

```
Electronics:
Revenue = 5,200,000

Accessories:
Revenue = 2,100,000
```

The raw observation is:

> Electronics generated more revenue than Accessories.

A useful analysis then asks:

- Is this because electronics are more expensive?
- Are more electronics sold?
- Which electronics products contribute most?
- Which cities purchase them?
- Are discounts affecting the result?

The second layer is where real analysis begins.

---

55. Revenue Is Not Everything

Suppose:

```
Product A
Units sold = 5,000
Revenue = 500,000

Product B
Units sold = 100
Revenue = 800,000
```

Product B generates more revenue.

But Product A has much greater sales volume.

You need multiple metrics to understand the business.

Useful metrics include:

```
Revenue
Orders
Units sold
Average order value
Average selling price
Customers
Purchase frequency
Discount rate
```

---

56. Create a KPI Dashboard DataFrame

```
kpis = pd.DataFrame({
    "metric": [
        "Total Revenue",
        "Total Orders",
        "Total Units",
        "Total Customers",
        "Average Order Value"
    ],
    "value": [
        total_revenue,
        total_orders,
        total_units,
        total_customers,
        average_order_value
    ]
})

print(kpis)
```

This can later become the foundation of a dashboard.

---

57. Build an Automated Analysis Function

Instead of keeping everything in global code:

```
def calculate_kpis(df):
    total_revenue = df["revenue"].sum()
    total_orders = df["order_id"].nunique()
    total_units = df["quantity"].sum()
    total_customers = df["customer_id"].nunique()

    average_order_value = (
```

```
        total_revenue / total_orders
        if total_orders
        else 0
    )

    return {
        "total_revenue": total_revenue,
        "total_orders": total_orders,
        "total_units": total_units,
        "total_customers": total_customers,
        "average_order_value": average_order_value
    }
```

Use it:

```
kpis = calculate_kpis(df)

print(kpis)
```

---

58. Create a Reusable Analysis Function

```
def analyze_sales(df):
    results = {}

    results["kpis"] = calculate_kpis(df)

    results["top_products"] = (
        df.groupby("product")["revenue"]
        .sum()
        .sort_values(ascending=False)
        .head(10)
    )
```

```
    results["category_revenue"] = (
        df.groupby("category")["revenue"]
        .sum()
        .sort_values(ascending=False)
    )

    results["city_revenue"] = (
        df.groupby("city")["revenue"]
        .sum()
        .sort_values(ascending=False)
    )

    results["monthly_revenue"] = (
        df.groupby("month")["revenue"]
        .sum()
    )

    return results
```

Then:

```
results = analyze_sales(df)
```

---

59. Export Analysis Results

Save product analysis:

```
product_summary.to_csv(
    "product_analysis.csv"
)
```

Save category analysis:

```
category_summary.to_csv(
    "category_analysis.csv"
)
```

Save customer analysis:

```
customer_summary.to_csv(
    "customer_analysis.csv"
)
```

---

60. Create an Excel Report

You can put multiple analysis tables into one workbook:

```
with pd.ExcelWriter(
    "sales_analysis_report.xlsx"
) as writer:

    kpis.to_excel(
        writer,
        sheet_name="KPIs",
        index=False
    )

    product_summary.to_excel(
        writer,
        sheet_name="Products"
    )

    category_summary.to_excel(
```

```
        writer,
        sheet_name="Categories"
    )

    city_revenue.to_excel(
        writer,
        sheet_name="Cities"
    )

    customer_summary.to_excel(
        writer,
        sheet_name="Customers"
    )
```

This creates a reusable analysis report.

---

61. Add a Data Quality Report

Create:

```
quality_report = {
    "rows": len(df),
    "columns": len(df.columns),
    "duplicate_rows": df.duplicated().sum(),
    "missing_values": int(df.isna().sum().sum()),
    "unique_customers": df["customer_id"].nunique(),
    "unique_products": df["product"].nunique()
}

for key, value in quality_report.items():
    print(f"{key}: {value}")
```

This tells you whether the data is ready for analysis.

---

62. A Complete Project Pipeline

Your final project should roughly follow:

1. Load data

2. Inspect

3. Clean

4. Validate

5. Create calculated columns

6. Calculate KPIs

7. Analyze products

8. Analyze categories

9. Analyze customers

10. Analyze cities

11. Analyze time trends

12. Visualize

13. Export results

14. Write insights

---

63. Suggested Notebook Structure

If you use Jupyter Notebook, organize it like this:

```
# Sales Data Analysis

## 1. Project Objective

## 2. Import Libraries

## 3. Load Dataset

## 4. Initial Inspection

## 5. Data Cleaning

## 6. Data Validation

## 7. Feature Engineering

## 8. Key Performance Indicators

## 9. Product Analysis

## 10. Category Analysis

## 11. Customer Analysis

## 12. City Analysis
```

13. Time Analysis

14. Visualization

15. Key Findings

16. Recommendations Based on Evidence

17. Conclusion

This structure makes the analysis easy for another developer or analyst to follow.

---

64. What Makes This a Real Data Analysis Project?

A project is not:

```
df.head()
df.describe()
df.groupby(...)
```

followed by no explanation.

A real analysis connects:

```
Question
  
Data
  
Cleaning
  
Calculation
```

Evidence
Interpretation

For every important result, ask:

> What does this number tell me?

---

65. Example Final Findings

Your final notebook might contain findings such as:

Finding 1:
Revenue increased during the final two months of the dataset.

Finding 2:
One product category generated a substantially larger share
of total revenue than the others.

Finding 3:
A small number of products accounted for a large portion
of total revenue.

Finding 4:
The city-level distribution of revenue was uneven.

Finding 5:
Several customers generated repeated purchases and
represented a meaningful portion of total revenue.

These are examples of the **type** of conclusion you should derive from your actual dataset. Do not copy them as facts unless your analysis produces them.

---

66. Do Not Confuse Analysis With Causation

Suppose you discover:

`Discounted orders have higher average revenue.`

You cannot automatically conclude:

> Discounts caused customers to spend more.

Other factors may exist:

- Expensive products may receive discounts.
- Returning customers may receive discounts.
- Seasonal campaigns may coincide with discounts.
- Large orders may receive special pricing.

A responsible analyst distinguishes:

`Observed relationship`

from:

`Proven cause`

---

67. Final Project Challenge

Build the complete project without copying the analysis code above.

Your project should answer at least these questions:

Sales

1. What is total revenue?
2. How many orders were placed?
3. How many units were sold?
4. What is the average order value?

Products

5. What are the top 10 products by revenue?
6. What are the top 10 products by units sold?
7. Which products have high volume but comparatively low revenue?

Categories

8. Which category generates the most revenue?
9. Which category sells the most units?

Customers

10. Who are the top 10 customers by revenue?
11. Which customers place the most orders?

Locations

12. Which city generates the most revenue?
13. Which city has the most orders?

Time

14. How does revenue change by month?

15. Which weekday has the highest revenue?

Payments

16. Which payment method is most frequently used?
17. Which payment method generates the most revenue?

Discounts

18. What percentage of orders receive discounts?
19. How does average order value differ between discounted and non-discounted orders?

Visualization

Create at least:

- 1 revenue trend chart
- 1 category chart
- 1 top-products chart
- 1 city chart
- 1 relationship/scatter chart

---

68. Advanced Challenge

After completing the basic project, add:

Customer segmentation

Create groups such as:

```
High Value
Regular
Occasional
```

based on measurable rules you define.

For example:

```
customer_summary["orders"]
customer_summary["revenue"]
```

Then document the rules rather than choosing categories arbitrarily.

Monthly growth

Calculate month-over-month revenue change:

```
monthly_revenue = (
    df.groupby("month")["revenue"]
    .sum()
)

monthly_growth = (
    monthly_revenue
    .pct_change() * 100
)

print(monthly_growth)
```

Revenue contribution

Calculate each category's percentage of total revenue:

```
category_summary["revenue_share"] = (
```

```
    category_summary["revenue"]
    / category_summary["revenue"].sum()
    * 100
)
```

Repeat customers

Count customer orders:

```
customer_orders = (
    df.groupby("customer_id")["order_id"]
    .nunique()
)
```

Then:

```
repeat_customers = (
    customer_orders > 1
).sum()

print(repeat_customers)
```

---

69. Interview Questions

1. What is the difference between data cleaning and data analysis?

Data cleaning prepares data for reliable use. Data analysis uses that prepared data to answer questions and discover patterns.

2. Why should you inspect data before analyzing it?

Because incorrect types, missing values, duplicates, and inconsistent categories can produce misleading results.

3. Why use `groupby()`?

To split data into groups and calculate statistics for each group.

4. Why use `nunique()` for orders?

Because the same order could potentially appear across multiple rows.

5. What is a KPI?

A Key Performance Indicator is a measurable value used to monitor an important aspect of performance.

6. What is feature engineering?

Creating useful variables from existing data.

Example:

```
df["revenue"] = (
    df["quantity"] *
    df["unit_price"] *
    (1 - df["discount"])
)
```

7. Why are visualizations useful?

They can make trends, distributions, comparisons, and relationships easier to detect.

8. Does correlation prove causation?

No.

9. Why should you preserve raw data?

So the original dataset remains available for auditing, debugging, comparison, and reprocessing.

10. What makes an analysis trustworthy?

Clear questions, reliable cleaning, appropriate calculations, validation, transparent assumptions, and evidence-based conclusions.

---

70. Cheat Sheet

Load

```
pd.read_csv()
pd.read_excel()
pd.read_json()
```

Inspect

```
df.head()
df.shape
df.info()
df.describe()
df.dtypes
df.isna().sum()
```

Clean

```
df.drop_duplicates()
df.dropna()
df.fillna()
df.astype()
pd.to_numeric()
pd.to_datetime()
```

Transform

```
df["new"] = ...
df["column"].str.strip()
df["column"].str.lower()
df["column"].str.title()
```

Analyze

```
df["column"].sum()
df["column"].mean()
df["column"].median()
df["column"].nunique()
df["column"].value_counts()
```

Group

```
df.groupby("category")["revenue"].sum()
```

Multiple aggregations

```
df.groupby("category").agg(
    revenue=("revenue", "sum"),
    orders=("order_id", "nunique")
)
```

Sort

```
df.sort_values(
    "revenue",
    ascending=False
)
```

Dates

```
df["date"].dt.year
df["date"].dt.month
df["date"].dt.day_name()
```

```
df["date"].dt.to_period("M")
```

Visualize

```
df.plot()
df.plot(kind="bar")
df.plot(kind="line")
df.plot(kind="scatter")
```

---

71. Project Deliverables

Your final project should contain:

```
sales_analysis/
□
□□□ data/
□   □□□ sales.csv
□
□□□ charts/
□   □□□ monthly_revenue.png
□   □□□ category_revenue.png
□   □□□ top_products.png
□   □□□ city_revenue.png
□
□□□ sales_analysis.ipynb
□
□□□ sales_analysis_report.xlsx
□
□□□ README.md
```

The README should explain:

Project Objective
Dataset
Tools Used
Cleaning Steps
Analysis Performed
Key Metrics
Visualizations
Key Findings
Limitations
How to Run

---

72. Portfolio Description

When you eventually put this project on GitHub or a portfolio, describe the work rather than simply saying:

> "Made a pandas project."

A stronger description explains:

- What dataset was analyzed
- What questions were investigated
- How the data was cleaned
- Which metrics were calculated
- Which visualizations were created
- What insights were discovered

The project demonstrates that you can move from **raw data to evidence-based analysis**, not merely execute pandas commands.

---

73. Final Mental Model

Remember:

```
        DATA
         ↓
    ASK QUESTIONS
         ↓
       CLEAN
         ↓
     VALIDATE
         ↓
    TRANSFORM
         ↓
     ANALYZE
         ↓
    VISUALIZE
         ↓
     INTERPRET
         ↓
     INSIGHT
```

The most important skill is not memorizing:

```
groupby()
agg()
plot()
```

The important skill is knowing:

> **Which question should I ask, which data can answer it, which calculation represents that question correctly, and what does the result actually mean?**

That is the foundation of practical data analysis with Python.

---

5 Minute Challenge

Create a small sales DataFrame yourself with:

```
product
category
quantity
unit_price
discount
city
```

Then calculate:

1. Revenue for every row.
2. Total revenue.
3. Total units sold.
4. Revenue by category.
5. Revenue by city.
6. Top 3 products.
7. A bar chart of revenue by category.

Do not copy the project dataset. Create your own data and make the analysis work from scratch.

Key Takeaways

- A data analysis project begins with questions, not code.
- pandas provides the core DataFrame-based workflow for practical tabular analysis.

- Cleaning must happen before trusting the results.
- Calculated columns turn raw fields into useful business metrics.
- `groupby()` is one of the most important tools for grouped analysis.
- Multiple metrics provide a better understanding than one number.
- Time-based analysis reveals trends.
- Visualization helps communicate patterns.
- Correlation does not prove causation.
- Every major conclusion should be supported by data.
- A good analyst explains both **what happened** and **what the data can and cannot tell us**.

Official References

Pandas Documentation:
https://pandas.pydata.org/docs/

Pandas Getting Started Tutorials:
https://pandas.pydata.org/docs/getting_started/intro_tutorials/

Pandas User Guide:
https://pandas.pydata.org/docs/user_guide/

Pandas GroupBy Guide:
https://pandas.pydata.org/docs/user_guide/groupby.html

Pandas Visualization Guide:
https://pandas.pydata.org/docs/user_guide/visualization.html

The current pandas documentation is for version 3.0.6.

Visit haas.dev for more resources and guides.

https://dev-roast-app.vercel.app

Resources:

https://dev-roast-app.vercel.app/resources