Python for Analytics Interviews
Structured Learning Guide

A complete roadmap — from workflow to code to business thinking
Covering Pandas, NumPy, Matplotlib, Seaborn and EDA patterns

By a Data Scientist @ PayPal — Analyst Roadmap Series

Analytics interviews do not test OOPs, decorators or DSA. They test whether you can take a messy dataset, clean it, manipulate it and extract meaningful business insights from it. This document gives you the structured approach to learn exactly what is needed — in the right order — so you do not waste a single hour.

How to use this document:

- Follow the chapters in order — each builds on the previous one
- Do not move to the next chapter until you can write the code from memory
- For every concept — ask yourself: how would I use this on a business dataset?
- Practice on real datasets from Kaggle — not toy examples

Chapters covered:

Chapter 1: The Analyst Workflow: Where Python Fits In

Understanding why Python matters and exactly where it sits in your analysis workflow

---

THE WORKFLOW

Most people learn Python in isolation. They never understand where it actually sits in the data pipeline. Here is the complete picture:

Step 1 — SQL: Extract and filter

Write SQL to extract exactly the data you need from the database. Filter it. Aggregate it. Join it. You never pull the entire table — only what you need for your analysis. This keeps the data small enough for Python to handle efficiently.

Step 2 — Python: Load into memory

The filtered SQL result is loaded into Python as a Pandas DataFrame. It now lives in your RAM. This is why SQL filtering matters — Python loads data into memory, so you need to control how much data you bring in.

Step 3 — Pandas: Clean and manipulate

Handle missing values. Remove duplicates. Fix data types. Rename columns. Create new features. Merge additional datasets. Your dataset must be clean and correct before any analysis begins.

Step 4 — NumPy + Pandas: Analyse

Calculate statistics. Find distributions. Detect outliers using IQR or percentiles. Calculate correlations. GroupBy and aggregate to find patterns. This is where the actual analytical thinking happens.

Step 5 — Matplotlib + Seaborn: Visualise and communicate

Plot distributions with histograms. Detect outliers with box plots. Understand variable relationships with scatter plots. Check correlations with heatmaps. Visualisation is not decoration — it is how you find and communicate patterns that numbers alone cannot show.

> **Business context:**
> Interviewers test Step 3, 4 and 5. They give you a dataset and ask you to walk through your analysis. The candidate who understands the workflow and applies it systematically always stands out. The candidate who jumps straight to analysis on uncleaned data always fails.

Why SQL alone is not enough:

- SQL can calculate percentiles. Python can visualise them as a box plot — which immediately reveals outliers, skewness and distribution shape that a number alone cannot show.
- SQL cannot easily produce a correlation heatmap across all variables. One line of Seaborn code does it.
- SQL cannot produce distribution plots, scatter matrices or pair plots that show the relationship between every variable simultaneously.
- SQL cannot flag multicollinearity in your feature set before modelling. Python can in 3 lines.

> **Resources for Chapter 1**
>
> - Kaggle Learn — kaggle.com/learn/pandas — free hands-on Pandas course with real datasets
> - Real Python — realpython.com — Python for data analysis tutorial series
> - Towards Data Science — towardsdatascience.com — analyst workflow articles

Chapter 2: Python Foundations You Actually Need

Not OOPs. Not decorators. The specific Python concepts that appear in analytics interviews

---

WHAT YOU NEED VS WHAT YOU DON'T

Do NOT waste time on:

- OOP, classes, inheritance, decorators, generators
- DSA — linked lists, binary trees, sorting algorithms
- Lambda functions beyond basic use
- Regular expressions in depth

DO master these:

- Lists, dictionaries, tuples — and when to use each
- List comprehensions — the Pythonic way to filter and transform
- Functions — writing reusable analysis functions
- File I/O — reading CSV, Excel, JSON into Python
- Basic string operations — cleaning messy text columns
- Error handling — try/except for robust data pipelines

Q1 What is the difference between a list, tuple and dictionary? When do you use each in analytics?

Answer:

List: ordered, mutable, allows duplicates. Use for sequences of data you need to modify.
Example: list of column names to drop, list of values to filter on.

Tuple: ordered, immutable, allows duplicates. Use for fixed data that should not change.
Example: storing (latitude, longitude) pairs, fixed category mappings.

Dictionary: key-value pairs, unordered, mutable. Use for lookups and mappings.
Example: mapping customer_id to customer_name, mapping category codes to labels.

```
# List — column names to keep
cols_to_keep = ['customer_id', 'revenue', 'country', 'created_at']
df = df[cols_to_keep]

# Dictionary — map category codes to readable labels
category_map = {1: 'Premium', 2: 'Standard', 3: 'Basic'}
df['tier_label'] = df['tier_code'].map(category_map)

# List comprehension — filter columns containing 'amount'
amount_cols = [col for col in df.columns if 'amount' in col.lower()]
```

Interview tip: List comprehensions appear in almost every Python analytics interview. Know how to use them to filter, transform and create new lists in one line. They are faster and more Pythonic than for loops.

Q2 What is a list comprehension? Write one to filter all transactions above Rs 10,000.

Answer:

A list comprehension is a compact way to create a new list by applying an expression and optionally filtering an existing iterable — all in one line.

Syntax: [expression for item in iterable if condition]

```
# Traditional for loop
high_value = []
for txn in transactions:
    if txn > 10000:
        high_value.append(txn)

# List comprehension — same result, one line
high_value = [txn for txn in transactions if txn > 10000]

# With transformation — get 10% of each high value transaction
tax = [txn * 0.10 for txn in transactions if txn > 10000]

# Dictionary comprehension — create lookup from two lists
customer_revenue = {cid: rev for cid, rev in zip(customer_ids, revenues)}
```

Resources for Chapter 2

- Python Official Docs — docs.python.org/3/tutorial — foundation concepts
- Real Python — realpython.com/python-lists-tuples — lists and tuples guide
- W3Schools Python — w3schools.com/python — quick reference for all basics
- Practice — HackerRank Python domain — hackerrank.com/domains/python

Chapter 3: Pandas: Data Cleaning and Manipulation

The most tested Python topic in every analytics interview — master this chapter first

---

CORE CONCEPT

A DataFrame is a 2D labelled data structure — like an Excel sheet but in Python. Every analytics task starts with loading data into a DataFrame, inspecting it and cleaning it before any analysis.

Q1 How do you inspect a new dataset before starting any analysis?

Answer:

Always run these 5 commands in order before touching any data:

```
import pandas as pd

df = pd.read_csv('transactions.csv')

# 1. Shape — how many rows and columns?
print(df.shape)               # (1000000, 15)

# 2. Column names and data types
print(df.dtypes)

# 3. First 5 rows — what does the data look like?
print(df.head())

# 4. Missing values — how many NULLs per column?
print(df.isnull().sum())

# 5. Summary statistics — distributions at a glance
print(df.describe())

# Bonus: unique values in categorical columns
print(df['status'].value_counts())
```

Business context:

Interviewers give you a dataset and watch what you do first. Candidates who immediately start calculating metrics fail. Candidates who inspect the data first — checking shape, types, nulls, distributions — show analytical maturity. Always run these 5 commands before anything else.

Interview tip: df.describe() gives you count, mean, std, min, 25%, 50%, 75%, max for all numeric columns. A large gap between mean and 75th percentile immediately signals outliers or right skew. Spotting this from describe() in an interview shows strong statistical intuition.

Q2 How do you handle missing values in a dataset? Walk through your decision process.

Answer:

Never blindly drop or fill. Always investigate first.

Step 1: Identify — how many NULLs? What percentage of the column?
Step 2: Understand — is the NULL random or does it mean something? (e.g. NULL in 'discount' means no discount was applied — not missing data)
Step 3: Decide based on context:
- < 5% missing — usually safe to drop rows
- > 30% missing — consider dropping the column
- Numerical column — fill with median (robust to outliers) or mean
- Categorical column — fill with mode (most frequent value)
- Time series — forward fill (last known value carried forward)
- Business-specific — fill with a domain-meaningful default (e.g. 0 for no discount)

```
# Check missing values
missing = df.isnull().sum()
missing_pct = (df.isnull().sum() / len(df) * 100).round(2)
print(pd.DataFrame({'count': missing, 'pct': missing_pct}))

# Drop rows with any NULL in critical columns
df.dropna(subset=['customer_id', 'revenue'], inplace=True)

# Fill numerical with median (robust to outliers)
df['transaction_amount'].fillna(df['transaction_amount'].median(), inplace=True)

# Fill categorical with mode
df['payment_method'].fillna(df['payment_method'].mode()[0], inplace=True)

# Forward fill for time series
df['daily_metric'].fillna(method='ffill', inplace=True)

# Fill with a domain-specific default
df['discount_pct'].fillna(0, inplace=True)  # NULL = no discount
```

Interview tip: Using median instead of mean for numerical imputation shows statistical maturity. Median is robust to outliers. Mean is pulled by extreme values. Always mention this reasoning in interviews.

Q3 How do you handle duplicates? What is the difference between exact duplicates and business-level duplicates?

Answer:

Exact duplicates: every column value is identical across two rows.
Business duplicates: rows that represent the same real-world event but may differ in some columns (e.g. same transaction_id but different timestamps due to retries).

Exact duplicates are easy to remove. Business duplicates require understanding the data and keeping the most relevant version.

```
# Check for exact duplicates
```

```
print(f'Duplicate rows: {df.duplicated().sum()}')

# Remove exact duplicates
df.drop_duplicates(inplace=True)

# Business duplicates — keep most recent transaction per customer
df_deduped = (
    df.sort_values('created_at', ascending=False)
      .drop_duplicates(subset=['customer_id'], keep='first')
)

# Check duplicates on specific columns
print(df.duplicated(subset=['transaction_id']).sum())

# Find which rows are duplicates
print(df[df.duplicated(subset=['transaction_id'], keep=False)])
```

Business context:
This connects directly to the SQL deduplication pattern from Chapter 2 of the SQL document. In SQL you use ROW_NUMBER. In Pandas you use sort_values + drop_duplicates. The business logic is identical — the tool is different.

Q4 How do you filter, select and transform columns in Pandas?

Answer:
Three selection methods — know all three and when to use each:
df['col'] or df[['col1','col2']] — column selection
df.loc[rows, cols] — label-based selection
df.iloc[rows, cols] — integer position-based selection

The loc vs iloc distinction is a very common interview question.

```
# Column selection
revenue = df['revenue']                    # single column -> Series
subset = df[['customer_id', 'revenue']]    # multiple columns -> DataFrame

# Row filtering — boolean indexing
high_value = df[df['revenue'] > 10000]
india_users = df[df['country'] == 'India']
combined = df[(df['revenue'] > 1000) & (df['country'] == 'India')]

# loc — label based (use with column names)
df.loc[df['revenue'] > 10000, 'customer_id']

# iloc — integer position based
df.iloc[0:5, 0:3]    # first 5 rows, first 3 columns

# Create new column
df['revenue_inr'] = df['revenue_usd'] * 83.5

# Apply function to column
df['name_clean'] = df['name'].str.strip().str.lower()
```

```
# Conditional column creation
df['tier'] = df['revenue'].apply(lambda x: 'High' if x > 10000 else 'Low')
```

Interview tip: The difference between loc and iloc is asked in almost every Python analytics interview. loc uses labels (column names, index labels). iloc uses integer positions (row 0, column 2). Confusing them causes silent wrong selections.

Q5 How do you merge two DataFrames? What are the different types of merges?

Answer:

Pandas merge is equivalent to SQL JOIN. Same logic, same types.
inner (default), left, right, outer — exactly like INNER, LEFT, RIGHT, FULL OUTER JOIN in SQL.

Critical: always check the shape before and after merging to catch fan-out (same problem as SQL joins).

```
# Inner merge — only matching rows
merged = pd.merge(transactions, customers,
                  on='customer_id', how='inner')

# Left merge — all transactions, matched customer info
merged = pd.merge(transactions, customers,
                  on='customer_id', how='left')

# Check for fan-out before and after
print(f'Before merge: {len(transactions)} rows')
print(f'After merge:  {len(merged)} rows')
# If after > before -> fan-out -> check customers table grain

# Merge on different column names
merged = pd.merge(transactions, customers,
                  left_on='cust_id', right_on='customer_id',
                  how='left')

# Concatenate — stack rows vertically (like UNION ALL)
combined = pd.concat([df_jan, df_feb, df_mar], ignore_index=True)
```

Business context:

Fan-out in Pandas merge is the exact same problem as in SQL. If the right DataFrame has multiple rows per merge key, every left row gets duplicated. Always check len(df) before and after any merge. This is one of the most common silent data corruption issues in analytics.

Resources for Chapter 3

- Pandas Official Docs — pandas.pydata.org/docs/user_guide — complete reference
- DataCamp — datacamp.com/courses/data-manipulation-with-pandas — best practical course
- Kaggle Learn — kaggle.com/learn/pandas — free hands-on exercises
- Real Python — realpython.com/pandas-dataframe — comprehensive DataFrame guide
- Practice datasets — Kaggle.com — use Titanic, Sales, or any business dataset

Chapter 4: Pandas: Aggregation, GroupBy and Pivot

The analytics operations that mirror SQL GROUP BY — but with more flexibility

---

CORE CONCEPT

GroupBy in Pandas = GROUP BY in SQL. Split data into groups, apply an aggregation, combine results. Pivot tables summarise data across two dimensions simultaneously.

Q1 Show the top 5 customers by total revenue and number of transactions.

Answer:

This is the most commonly asked Python analytics interview question. My friend froze on this.
Three steps: groupby → aggregate → sort → head.

```
# Top 5 customers by revenue
top_customers = (
    df.groupby('customer_id')
      .agg(
          total_revenue=('revenue', 'sum'),
          transaction_count=('transaction_id', 'count'),
          avg_order_value=('revenue', 'mean'),
          last_transaction=('created_at', 'max')
      )
      .sort_values('total_revenue', ascending=False)
      .head(5)
      .reset_index()
)

print(top_customers)
```

Business context:

This is the analytics equivalent of TOP-N SQL pattern. GroupBy + agg + sort + head. Every analytics interview tests some version of this. Know it cold. The named aggregation syntax (col=('source_col', 'func')) is cleaner and more readable than the dictionary form.

Interview tip: Always reset_index() after groupby to convert the group keys back to regular columns. Forgetting this leaves customer_id as the index, which causes issues in downstream operations.

Q2 How do you calculate month-over-month revenue growth using Pandas?

Answer:

Python equivalent of the SQL LAG() pattern. Use resample or groupby on date column, then pct_change().

```
# Ensure datetime type
df['created_at'] = pd.to_datetime(df['created_at'])
```

```
# Monthly revenue
monthly = (
    df.groupby(df['created_at'].dt.to_period('M'))['revenue']
      .sum()
      .reset_index()
)
monthly.columns = ['month', 'revenue']

# Month-over-month growth %
monthly['mom_growth_pct'] = monthly['revenue'].pct_change() * 100
monthly['mom_growth_pct'] = monthly['mom_growth_pct'].round(2)

print(monthly)

# Rolling 3-month average
monthly['rolling_3m_avg'] = monthly['revenue'].rolling(window=3).mean()
```

Q3 What is a pivot table in Pandas? Write one to show revenue by product category and month.

Answer:

A pivot table summarises data across two dimensions simultaneously. Rows = one dimension, columns = another dimension, values = the aggregated metric.

```
df['month'] = pd.to_datetime(df['created_at']).dt.to_period('M')

# Pivot: rows=category, columns=month, values=revenue
pivot = df.pivot_table(
    values='revenue',
    index='category',
    columns='month',
    aggfunc='sum',
    fill_value=0         # fill missing combos with 0
)

print(pivot)

# Add row and column totals
pivot['Total'] = pivot.sum(axis=1)           # row total
pivot.loc['Total'] = pivot.sum(axis=0)       # column total
```

Interview tip: fill_value=0 is critical in pivot tables. Without it, combinations with no data show as NaN. In a revenue report, NaN looks like missing data. 0 is the correct business representation of zero revenue.

Q4 How do you use apply() and transform()? What is the difference?

Answer:

apply(): applies a function to each row or column, returns a reduced result.
transform(): applies a function to each group but KEEPS the original index — returns same shape as input.

transform() is used when you want to add a column based on group-level statistics while keeping all rows intact — like adding a 'department average salary' column to each employee row.

```
# apply: calculate custom metric per row
df['risk_score'] = df.apply(
    lambda row: row['amount'] * row['fraud_rate'], axis=1
)

# transform: add group-level statistic to each row
# Add 'average revenue for this customer's country' to every row
df['country_avg_revenue'] = df.groupby('country')['revenue'].transform('mean')

# Now you can compare each customer to their country average
df['vs_country_avg'] = df['revenue'] - df['country_avg_revenue']

# transform for normalisation within groups
df['revenue_normalised'] = df.groupby('category')['revenue'].transform(
    lambda x: (x - x.mean()) / x.std()
)
```

Business context:
transform() is the Pandas equivalent of SQL window functions with PARTITION BY. When an interviewer asks you to add a column showing 'what % of their country's total revenue does this customer represent' — transform() is the answer. This is a senior-level Pandas question.

Resources for Chapter 4

- Pandas Docs — pandas.pydata.org/docs/reference/groupby — GroupBy complete reference
- DataCamp — datacamp.com — GroupBy and pivot table tutorial
- Towards Data Science — groupby transform vs apply article
- Practice — use any sales dataset on Kaggle to practice all aggregation patterns

Chapter 5: NumPy: Statistical Analysis and Percentiles

The library underneath Pandas — and the one that powers every statistical calculation

---

CORE CONCEPT

NumPy provides fast numerical operations on arrays. For analytics interviews — the key NumPy skills are: statistical functions, percentile calculations, array operations and vectorisation.

Q1 How do you detect and handle outliers using NumPy and Pandas?

Answer:

Two methods. Know both and when to use each:
IQR method — robust to outliers, works on skewed data. Best for business analytics.
Z-score method — assumes normal distribution. Use only when data is roughly symmetric.

```
import numpy as np

# Method 1: IQR (recommended for skewed business data)
Q1 = df['revenue'].quantile(0.25)
Q3 = df['revenue'].quantile(0.75)
IQR = Q3 - Q1
lower_fence = Q1 - 1.5 * IQR
upper_fence = Q3 + 1.5 * IQR

outliers = df[(df['revenue'] < lower_fence) | (df['revenue'] > upper_fence)]
print(f'Outliers found: {len(outliers)}')

# Remove outliers
df_clean = df[(df['revenue'] >= lower_fence) & (df['revenue'] <= upper_fence)]

# Method 2: Z-score (for normally distributed data)
from scipy import stats
z_scores = np.abs(stats.zscore(df['revenue']))
df_clean = df[z_scores < 3]   # keep within 3 standard deviations

# Cap instead of remove (Winsorization)
df['revenue_capped'] = df['revenue'].clip(lower=lower_fence, upper=upper_fence)
```

Business context:

IQR method is preferred for financial data — transaction amounts, revenue, LTV — all tend to be right-skewed. Z-score assumes normality and is inappropriate for skewed distributions. Knowing this distinction and explaining it in an interview shows statistical depth.

Q2 What is PERCENTILE_CONT in SQL equivalent in Python? How do you use percentiles for business thresholds?

Answer:

np.percentile() and df.quantile() — the Python equivalents of SQL PERCENTILE_CONT.

Business use case: Setting a fraud threshold at the 95th percentile instead of using the mean — which is distorted by outliers.

```
# Calculate key percentiles
percentiles = df['transaction_amount'].quantile([0.25, 0.50, 0.75, 0.90, 0.95, 0.99])
print(percentiles)

# Using numpy
p95 = np.percentile(df['transaction_amount'].dropna(), 95)
print(f'95th percentile: Rs {p95:,.0f}')

# Set fraud threshold at 95th percentile instead of mean
fraud_threshold = np.percentile(df['transaction_amount'], 95)
print(f'Mean threshold would be: Rs {df["transaction_amount"].mean() * 3:,.0f}')
print(f'Percentile threshold is: Rs {fraud_threshold:,.0f}')

# Flag potential fraud
df['is_high_value'] = df['transaction_amount'] > fraud_threshold

# Full percentile summary
print(df['transaction_amount'].describe(percentiles=[.1,.25,.5,.75,.9,.95,.99]))
```

Interview tip: This directly connects Python to the statistics chapter. Mean-based thresholds on right-skewed data miss most outliers. Percentile-based thresholds reflect the actual data distribution. Always explain why you chose a percentile-based threshold in interviews.

Q3 What is vectorisation and why is it critical for large datasets?

Answer:

Vectorisation means applying an operation to an entire array at once rather than looping row by row.
NumPy and Pandas operations are vectorised — they use optimised C code underneath.
Python for loops are NOT vectorised — they are slow on large datasets.

```
import time

# Slow: Python loop
start = time.time()
result = []
for val in df['revenue']:
    result.append(val * 1.18)  # add 18% GST
print(f'Loop: {time.time()-start:.3f}s')

# Fast: vectorised operation
start = time.time()
result = df['revenue'] * 1.18
print(f'Vectorised: {time.time()-start:.3f}s')
# Vectorised is 100x+ faster on large datasets

# Vectorised string operations
df['name_clean'] = df['name'].str.lower().str.strip()
```

```
# Vectorised conditional
df['category'] = np.where(df['revenue'] > 10000, 'High', 'Low')

# np.where for multiple conditions
df['tier'] = np.select(
    [df['revenue'] > 100000, df['revenue'] > 10000, df['revenue'] > 1000],
    ['Platinum', 'Gold', 'Silver'],
    default='Bronze'
)
```

Resources for Chapter 5

- NumPy Official Docs — numpy.org/doc — complete NumPy reference
- DataCamp — numpy tutorial for data analysts
- Real Python — realpython.com/numpy-tutorial — comprehensive guide
- Practice — apply NumPy operations on any Kaggle financial dataset

Chapter 6: Matplotlib and Seaborn: Visualisation for EDA

Visualisation is not decoration — it is how you find patterns that numbers cannot show

---

CORE CONCEPT

In analytics interviews — you need to know which chart to use for which purpose. Not every chart for every situation. One wrong chart choice signals poor analytical thinking.

Chart selection guide:

- Distribution of one variable → Histogram
- Outlier detection → Box plot
- Relationship between two continuous variables → Scatter plot
- Correlation across all variables → Heatmap
- Categorical comparison → Bar chart
- Trend over time → Line chart
- All pairwise relationships → Pair plot (seaborn)

Q1 Plot a distribution and explain what you see.

Answer:

Always plot BEFORE calculating statistics. The distribution shape determines which statistics are appropriate.

```
import matplotlib.pyplot as plt
import seaborn as sns

fig, axes = plt.subplots(1, 2, figsize=(12, 4))

# Histogram — distribution shape
axes[0].hist(df['revenue'], bins=50, color='steelblue', edgecolor='white')
axes[0].axvline(df['revenue'].mean(), color='red', linestyle='--', label=f'Mean:
{df["revenue"].mean():.0f}')
axes[0].axvline(df['revenue'].median(), color='green', linestyle='--', label=f'Median:
{df["revenue"].median():.0f}')
axes[0].set_title('Revenue Distribution')
axes[0].legend()

# Box plot — outlier detection
axes[1].boxplot(df['revenue'].dropna())
axes[1].set_title('Revenue Box Plot — Outlier Detection')

plt.tight_layout()
plt.show()

# Seaborn — cleaner histplot
sns.histplot(df['revenue'], kde=True)  # kde=True adds smooth distribution curve
plt.show()
```

Interview tip: In an interview — when you see mean >> median in the histogram, immediately say: 'This distribution is right-skewed. The mean is pulled up by high-value outliers. I would use median as the representative central value and IQR for outlier detection.'

Q2 How do you create a correlation heatmap? How do you interpret it?

Answer:

A correlation heatmap shows the Pearson correlation coefficient between every pair of numerical variables simultaneously.

Range: -1 to +1. Values close to 1 = strong positive correlation. Close to -1 = strong negative. Close to 0 = weak or no correlation.

Why it matters in analytics: Highly correlated features (>0.8) should not both be used in a model — they add no independent information and inflate variance.

```
import seaborn as sns
import matplotlib.pyplot as plt

# Calculate correlation matrix
corr_matrix = df.select_dtypes(include='number').corr()

# Plot heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(
    corr_matrix,
    annot=True,           # show correlation values
    fmt='.2f',            # 2 decimal places
    cmap='RdYlGn',        # red=negative, green=positive
    center=0,             # center colour scale at 0
    vmin=-1, vmax=1,      # fix scale
    square=True,
    linewidths=0.5
)
plt.title('Correlation Heatmap — All Numerical Features')
plt.tight_layout()
plt.show()

# Find highly correlated pairs (|corr| > 0.8)
high_corr = (corr_matrix.abs() > 0.8) & (corr_matrix != 1)
print('Highly correlated pairs:')
print(corr_matrix[high_corr].stack().reset_index())
```

Business context:

A heatmap is the most powerful single visualisation for feature understanding in analytics and data science. It immediately reveals multicollinearity, which is critical before building any model. Knowing to look for and explain highly correlated pairs in an interview demonstrates end-to-end analytical thinking.

Q3 What is a pair plot? When is it useful?

Answer:

A pair plot shows scatter plots for every combination of numerical variables, with distribution histograms on the diagonal.

It gives a complete picture of all pairwise relationships in one visualisation.

Use when: you have 3-10 numerical variables and want to understand their relationships before modelling.

```
# Basic pair plot
sns.pairplot(df[['revenue', 'transaction_count', 'avg_order_value', 'churn']])
plt.show()

# Pair plot coloured by category
sns.pairplot(
    df[['revenue', 'txn_count', 'days_active', 'segment']],
    hue='segment',     # colour by customer segment
    diag_kind='kde',   # kernel density on diagonal
    plot_kws={'alpha': 0.5}
)
plt.show()
```

Resources for Chapter 6

- Matplotlib Docs — matplotlib.org/stable/tutorials — complete visualisation guide
- Seaborn Docs — seaborn.pydata.org/tutorial — gallery with code for every chart type
- Python Graph Gallery — python-graph-gallery.com — visual chart picker with code
- Kaggle — EDA notebooks — look at top-voted EDA notebooks for any dataset to see professional visualisation patterns

Chapter 7: Exploratory Data Analysis: The Full Workflow

This is what interviews actually test — the end-to-end analytical thinking process

---

THE EDA FRAMEWORK

EDA is not a list of functions. It is a thinking process. Interviewers give you a dataset and watch how you explore it. The structure of your thinking matters as much as the code.

The 6-step EDA framework:

1. Understand the data — shape, columns, types, what each variable represents
2. Check data quality — missing values, duplicates, invalid values, data type issues
3. Univariate analysis — understand each variable independently
4. Bivariate analysis — understand relationships between variables
5. Multivariate analysis — understand patterns across multiple variables simultaneously
6. Business insights — what does this tell us? What decisions does it support?

Q1 Walk through a complete EDA on a customer transactions dataset.

Answer:

This is the most comprehensive interview question. Here is the complete framework with code.

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

# ── STEP 1: UNDERSTAND THE DATA ──────────────────────────────
df = pd.read_csv('transactions.csv')
print(f'Shape: {df.shape}')
print(df.dtypes)
print(df.head(3))

# ── STEP 2: DATA QUALITY ─────────────────────────────────────
# Missing values
missing = (df.isnull().sum() / len(df) * 100).round(2)
print(missing[missing > 0])

# Duplicates
print(f'Exact duplicates: {df.duplicated().sum()}')
print(f'Duplicate transaction_ids: {df.duplicated("transaction_id").sum()}')

# Fix data types
df['created_at'] = pd.to_datetime(df['created_at'])
df['revenue'] = pd.to_numeric(df['revenue'], errors='coerce')

# ── STEP 3: UNIVARIATE ANALYSIS ──────────────────────────────
# Numerical — distribution and outliers
print(df['revenue'].describe(percentiles=[.1,.25,.5,.75,.9,.95,.99]))

fig, axes = plt.subplots(1, 2, figsize=(12,4))
sns.histplot(df['revenue'], kde=True, ax=axes[0])
axes[0].set_title('Revenue Distribution')
```

```
sns.boxplot(y=df['revenue'], ax=axes[1])
axes[1].set_title('Revenue Outliers')
plt.tight_layout(); plt.show()

# Categorical — frequency
print(df['country'].value_counts())
df['country'].value_counts().plot(kind='bar')
plt.title('Transactions by Country'); plt.show()

# ── STEP 4: BIVARIATE ANALYSIS ──────────────────────────
# Revenue by category
df.groupby('category')['revenue'].agg(['mean','median','sum']).sort_values('sum',
ascending=False)

# Scatter — revenue vs transaction count
plt.scatter(df['txn_count'], df['revenue'], alpha=0.3)
plt.xlabel('Transaction Count'); plt.ylabel('Revenue')
plt.title('Revenue vs Transaction Count'); plt.show()

# ── STEP 5: MULTIVARIATE ANALYSIS ──────────────────────
# Correlation heatmap
corr = df.select_dtypes(include='number').corr()
sns.heatmap(corr, annot=True, fmt='.2f', cmap='RdYlGn', center=0)
plt.title('Correlation Heatmap'); plt.show()

# ── STEP 6: BUSINESS INSIGHTS ──────────────────────────
# Top 10 customers by revenue
top10 = df.groupby('customer_id')['revenue'].sum().nlargest(10)
print(f'Top 10 customers drive {top10.sum()/df["revenue"].sum()*100:.1f}% of revenue')

# Revenue trend
df.groupby(df['created_at'].dt.to_period('M'))['revenue'].sum().plot()
plt.title('Monthly Revenue Trend'); plt.show()
```

Business context:

The difference between a 6 LPA and 18 LPA analyst is not the code they write. It is step 6 — turning numbers into business insights. Every interviewer is waiting for you to say 'the top 10 customers drive 60% of revenue' or 'revenue has been declining since March'. The code is just the tool. The insight is what gets you hired.

Interview tip: Always narrate as you code in interviews. Say what you are doing and why. 'I am checking for missing values before any analysis because missing data in the revenue column would make all my aggregations unreliable.' This shows thinking, not just typing.

Resources for Chapter 7

- Kaggle — search for 'EDA' notebooks on any dataset — study the top-voted ones
- Towards Data Science — EDA tutorial articles with real datasets
- DataCamp — Exploratory Data Analysis in Python course
- GitHub — search 'Python EDA' for real analyst notebooks

Chapter 8: Interview Patterns and Business Problems

Real questions asked in MNC analytics interviews — with the thinking framework

COMMON INTERVIEW PATTERNS

Q1 You are given a dataset with customer transactions. Find customers who have churned (no purchase in last 30 days) and those who are at risk (only one purchase in last 60 days).

Answer:

Framework: Calculate recency per customer. Apply business rules using conditional logic.

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

# Last purchase date and purchase count per customer
customer_stats = df.groupby('customer_id').agg(
    last_purchase=('created_at', 'max'),
    purchase_count=('transaction_id', 'count'),
    total_revenue=('revenue', 'sum')
).reset_index()

# Days since last purchase
customer_stats['days_since_last'] = (today - customer_stats['last_purchase']).dt.days

# Segment customers
def segment(row):
    if row['days_since_last'] > 30:
        return 'Churned'
    elif row['days_since_last'] <= 60 and row['purchase_count'] == 1:
        return 'At Risk'
    elif row['days_since_last'] <= 30 and row['purchase_count'] >= 5:
        return 'Loyal'
    else:
        return 'Active'

customer_stats['segment'] = customer_stats.apply(segment, axis=1)
print(customer_stats['segment'].value_counts())
```

Q2 Identify and remove outliers from a revenue column, then compare the mean and median before and after.

Answer:

Classic EDA question that tests outlier detection + statistical thinking.

```
# Before
print(f'Before — Mean: {df["revenue"].mean():,.0f} | Median: {df["revenue"].median():,.0f}')
print(f'Before — Rows: {len(df)}')

# IQR outlier removal
```

```
Q1 = df['revenue'].quantile(0.25)
Q3 = df['revenue'].quantile(0.75)
IQR = Q3 - Q1
df_clean = df[
    (df['revenue'] >= Q1 - 1.5*IQR) &
    (df['revenue'] <= Q3 + 1.5*IQR)
].copy()

# After
print(f'After  — Mean: {df_clean["revenue"].mean():,.0f} | Median:
{df_clean["revenue"].median():,.0f}')
print(f'After  — Rows: {len(df_clean)} (removed {len(df)-len(df_clean)})')

# Visualise before and after
fig, axes = plt.subplots(1, 2, figsize=(12,4))
sns.boxplot(y=df['revenue'], ax=axes[0]).set_title('Before')
sns.boxplot(y=df_clean['revenue'], ax=axes[1]).set_title('After Outlier Removal')
plt.show()
```

Q3 A senior analyst shares a Python script that takes 2 hours to run on 10 million rows. How do you optimise it?

Answer:

Framework: Profile first, then optimise. Never optimise blindly.

Common culprits in slow Pandas code:

1. iterrows() or for loops — replace with vectorised operations
2. apply() with complex Python functions — replace with Pandas/NumPy built-ins
3. Loading entire dataset when only a subset is needed
4. Wrong data types — int64 where int32 would do, object where category would do
5. Multiple passes through the data — combine into one operation

```
# SLOW: iterrows loop
for idx, row in df.iterrows():
    df.loc[idx, 'tax'] = row['revenue'] * 0.18

# FAST: vectorised
df['tax'] = df['revenue'] * 0.18

# SLOW: apply with Python function
df['tier'] = df['revenue'].apply(lambda x: 'High' if x > 10000 else 'Low')

# FAST: np.where
df['tier'] = np.where(df['revenue'] > 10000, 'High', 'Low')

# Reduce memory: convert object to category
print(f'Before: {df.memory_usage(deep=True).sum()/1e6:.1f} MB')
for col in df.select_dtypes(include='object').columns:
    if df[col].nunique() < 50:   # low cardinality -> category
        df[col] = df[col].astype('category')
print(f'After: {df.memory_usage(deep=True).sum()/1e6:.1f} MB')

# Load only needed columns
df = pd.read_csv('big_file.csv', usecols=['customer_id','revenue','country'])
```

```
# Read in chunks for very large files
chunks = []
for chunk in pd.read_csv('big_file.csv', chunksize=100000):
    chunk_result = chunk.groupby('country')['revenue'].sum()
    chunks.append(chunk_result)
result = pd.concat(chunks).groupby(level=0).sum()
```

Business context:

Performance optimisation in Python is tested at mid-to-senior level analytics interviews. Knowing the hierarchy — vectorised operations > NumPy > Pandas built-ins > apply > iterrows — shows production Python maturity.

Interview tip: The single biggest performance gain in Pandas is eliminating iterrows(). It is 100-1000x slower than vectorised operations on large datasets. If you see iterrows() in any production code — that is your first target for optimisation.
