Practical 1 — Data Loading & Cleaning (Detailed Answers)

1. Difference between CSV, Excel, and JSON formats.
CSV: Plain text, comma-separated rows; light, human-readable, universally supported; no data types, no formulas, no multiple sheets.

Excel (.xls/.xlsx): Binary/OOXML; supports multiple sheets, cell types, formulas, styles, filters; heavier; great for business workflows.

JSON: Hierarchical (key–value, lists, nested objects); ideal for APIs and semi-structured data; preserves structure; not columnar by default.
When to use: CSV for tabular exports/ETL; Excel for analyst handoffs; JSON for nested/API data.

2. Libraries to load CSV, Excel, JSON into DataFrames

```
pandas.read_csv("file.csv")
pandas.read_excel("file.xlsx", sheet_name="Sheet1")
pandas.read_json("file.json", orient="records")
```

Tip: For big CSVs, add chunksize=100_000 and iterate; for Excel performance, install engines like openpyxl (xlsx) or xlrd (xls).

3. Check missing values after loading

```
df.info()                  # quick null counts & dtypes
df.isna().sum()            # per-column null totals
df.isna().mean()*100       # % missing per column
df[df.isna().any(1)].head()  # sample rows with any NaN
```

Viva tip: Mention pd.options.display.max_info_rows for large sets.

4. Handle missing data
Drop: df.dropna() (fast but may lose data).
Impute simple: df.fillna(0) / mean/median/mode per column.
Impute conditional: groupwise: df['col']=df.groupby('grp')['col'].transform(lambda s: s.fillna(s.median()))

Advanced: interpolation (df['col'].interpolate()), KNN/ML imputation (e.g., sklearn.impute.KNNImputer).
Guideline: Pick a method consistent with data generation process; always document imputation.

5. Remove duplicates
df.duplicated().sum()
df = df.drop_duplicates()   # all columns
df = df.drop_duplicates(subset=['id','date'])# key subset

Note: If you need first/last occurrence control, use keep='first'/'last'/False.

6. Merge multiple datasets
Relational join: pd.merge(a, b, on='key', how='inner|left|right|outer')
Row-wise concat: pd.concat([df1, df2], axis=0)
Column-wise concat: pd.concat([df1, df2], axis=1)
Index join: df1.join(df2, how='left')
Pitfalls: Key dtype mismatches (int vs str), duplicate keys (causing row explosion).

7. Purpose of data transformation
To convert raw data into analysis-ready form: type casting, normalization, deriving features, encoding categories, tidying (long↔wide), unit conversions—so models/plots are valid and comparable.

8. Compute total sales / average order value (AOV)
total_sales = df['Sales'].sum()
aov = df.groupby('OrderID')['Sales'].sum().mean()   # revenue per order
# or across time/category:
daily = df.groupby('Date')['Sales'].sum()
by_cat = df.groupby('Category')['Sales'].agg(['sum','mean','count'])

Tip: Ensure currency/units consistent before aggregating.

9. Visualizations for sales trends/product performance
Trends: Line chart (time on x-axis).
Comparisons: Bar/column charts (categories).
Share: Pie/donut (use sparingly).
Distribution: Box/violin (price, order value).
Variability: Error bars; seasonality: line with rolling mean.
Rule: One message per chart; label axes & units.

10. Why unify formats before analysis
Consistency reduces parsing errors, simplifies pipelines, ensures accurate joins/aggregations, and speeds EDA/modeling. Example: unify date formats (ISO 8601), currencies, and categorical labels.

Practical 2 — APIs & JSON (Detailed Answers)

1. What is an API & why use it in data analysis?
An API (Application Programming Interface) exposes data/services programmatically (HTTP endpoints). Analysts use APIs to pull fresh, filtered data on demand, enabling automation and reproducibility.

2. Purpose of an API key (e.g., OpenWeatherMap)
Identifies the caller, enforces rate limits/quotas, and prevents unauthorized use. Often passed as a query param or header; keep it secret (env vars, not hard-coded).

3. Library to send web requests
requests is the go-to: simple, robust, widely used. (Alternatives: httpx async; urllib stdlib.)

4. Typical format returned by OpenWeatherMap
JSON—nested objects (e.g., main, weather, wind, coord).

5. Extract specific info (temperature/humidity) from JSON
```
import requests
URL = "https://api.openweathermap.org/data/2.5/weather"
params = {"q":"Pune,IN","appid":API_KEY,"units":"metric"} # metric = °C
res = requests.get(URL, params=params)
data = res.json()
temp_c = data['main']['temp']
humidity = data['main']['humidity']
condition = data['weather'][0]['description']
```

Tip: Use res.raise_for_status() and guard keys: data.get('main', {}).get('temp').

6. Clean & preprocess weather data from an API

Normalize JSON to columns: pd.json_normalize(records)
Units: Kelvin→Celsius (°C = K − 273.15) if not using units=metric.
Types: parse timestamps → pd.to_datetime(dt, unit='s'); floats for numerics.
Missing: fillna() or drop; validate ranges (e.g., humidity 0–100).
Deduplicate: by city+timestamp; timezone alignment.

7. Analyze weather patterns
Descriptives: daily/weekly means, max/min, std.
Rolling stats: df['temp'].rolling(7).mean() for smoothing.
Anomalies: z-score/IQR on temp or pressure.
Seasonality: group by month or hour-of-day.

8. Plots for time-varying weather data
Line: temperature vs time; add rolling mean.
Bar: daily rainfall totals.
Scatter: wind speed vs humidity; temp vs pressure.
Box: monthly temp distribution.
Always label units (°C, mm, m/s, %).

9. How geography enhances visualization
Add lat/lon to plot maps (heatmaps/choropleths, bubble maps). Examples: compare city clusters, show temp gradient. With Python, use folium, geopandas, or export to BI tools.

10. Advantages of automated API collection vs manual downloads
Real-time/near-real-time updates
Repeatable pipelines (cron/Airflow)
Scalable to many locations/time points
Fewer manual errors; easier auditing/versioning.

PRACTICAL 3 – Customer Churn Dataset (Detailed Viva Answers)

1. What is customer churn? Why is it important to analyze it?
Customer churn means customers stopping the service.
In telecom, churn directly impacts revenue.
Analyzing churn helps companies identify unhappy customers and take steps to retain them (offers, call quality improvements, customer support).
Lower churn = higher customer lifetime value (CLV).

2. How to identify missing values in the dataset?
Use pandas functions:
df.isnull().sum()
df.info()
df[df.isnull().any(axis=1)]
Helps locate which columns and how many values are missing.

3. Techniques to handle missing data
Delete rows with missing values → df.dropna() (but may lose info).
Fill missing values using:
Mean → numeric continuous data.

Median → when outliers exist.
Mode → categorical data.
Group-wise filling: df.groupby('column').transform(lambda x: x.fillna(x.mean()))
Interpolation for time-based data.

4. How to detect and remove duplicate records?
df.duplicated().sum()
df = df.drop_duplicates()
Duplicates cause biased results, especially in churn prediction.

5. Data inconsistencies and how to fix them
Common inconsistencies:
Mixed uppercase/lowercase entries
Extra spaces
Misspelled categorical labels
Fix using:
df['column'] = df['column'].str.strip().str.lower()
df['column'] = df['column'].replace({'yes':'Yes','yess':'Yes'})

6. Why convert columns to correct data types? How?
Ensures mathematical operations and comparisons behave correctly.
Examples:
df['TotalCharges'] = df['TotalCharges'].astype(float)
df['Churn'] = df['Churn'].astype('category')

7. What are outliers and how do they affect analysis?
Outliers are values significantly different from the rest.
They can distort averages, produce misleading trends, and reduce model accuracy.
Detect using Boxplot, Z-score, or IQR method.

8. What is feature engineering? Give an example.
Creating new meaningful variables from existing data.
Example in churn dataset:
Create Tenure Group:
df['TenureGroup'] = pd.cut(df['tenure'], bins=[0,12,24,48,72], labels=['0-1yr','1-2yr','2-4yr','4-6yr'])
This helps models better capture churn patterns.

9. Why is normalization/scaling important?
To bring all numeric features to similar scale.
Prevents large-scale features from dominating model calculations.
Examples: StandardScaler, MinMaxScaler.

10. How to split data into training and testing sets? Why?
Use train_test_split() from sklearn.
Training set → learn model, Testing set → evaluate generalization.
Prevents overfitting.

PRACTICAL 4 – Data Wrangling (Detailed Viva Answers)

1. What is data wrangling and why is it important?
Data wrangling = cleaning, transforming, and organizing raw data.
Ensures data is accurate, structured, and ready for analysis or modeling.
2. How to clean column names in a dataset?
Replace spaces, special characters, and use consistent formatting:
df.columns = df.columns.str.strip().str.lower().str.replace(' ','_')
Good column names reduce errors in queries and functions.

3. Common strategies to handle missing values
Delete rows/columns with too many NaN.
Fill using:
mean/median (numerical)
mode (categorical)
forward-fill / backward-fill for time-series
Choose method based on data meaning, not convenience.

4. How to merge multiple datasets in pandas?
Use based on structure:
merge() → join on key columns
concat() → append rows/columns
join() → join on index
Correct merging ensures combined dataset has logical consistency.

5. Purpose of filtering and subsetting data
To focus on relevant records.
Example:
df[df['price'] > 500000]   # Filter
df[['price','area','location']] # Subset
Speeds up analysis and improves insights.

6. Difference between Label Encoding and One-Hot Encoding.
Label Encoding:
Converts categories into numeric values (0,1,2…).
Used when categories have order (e.g., small < medium < large).
One-Hot Encoding:
Creates a new binary column for each category.

Used when categories have no natural order (e.g., red, blue, green).

7. How to calculate average sale price by neighborhood?
df.groupby('Neighborhood')['SalePrice'].mean()
Helps compare pricing trends across areas.

8. Why handle outliers in housing price data?
Outliers inflate mean price and mislead decision-making.
Removing or capping stabilizes analysis and model accuracy.

9. Functions to detect outliers
Boxplot:
df.boxplot(column=['SalePrice'])
Z-score: values > 3 or < -3 often considered outliers.
IQR Method: (Q3 + 1.5*IQR) and (Q1 - 1.5*IQR) thresholds.

10. Next steps after data wrangling
EDA (Exploratory Data Analysis)
Data visualization
Feature selection
Model training
Evaluation and deployment

PRACTICAL 5 – Data Visualization using Matplotlib

1. What is Matplotlib and why is it used?
Matplotlib is a Python data visualization library.
It is used to create graphs and charts from data.
Helps convert numerical data into visual patterns for easier understanding.
Useful for EDA (exploratory data analysis) and presentations.

2. How do you import Matplotlib?
import matplotlib.pyplot as plt
pyplot is used for plotting functions similar to MATLAB style.
plt is a commonly used alias for convenience.

3. Which plot is best to show AQI trends over time?
Line plot is best when the x-axis represents time.
Shows increase/decrease trends clearly.

4. How to plot multiple pollutants (PM2.5, PM10, CO) on the same graph?
plt.plot(df['Date'], df['PM2.5'])
plt.plot(df['Date'], df['PM10'])

plt.plot(df['Date'], df['CO'])
plt.show()
Plot each pollutant with a separate plt.plot() call.
Add legend to differentiate lines.

5. Purpose of bar plots in AQI analysis
Used to compare pollutant levels across different days, locations, or stations.
Good for category vs quantity comparison.

6. What does a box plot show?
Displays distribution, median, quartiles, and outliers.
Helps detect abnormal AQI spikes.

7. How to add title, axis labels, and legend in Matplotlib?
plt.title("AQI Trend")
plt.xlabel("Date")
plt.ylabel("AQI Value")
plt.legend()
Title → tells what the graph represents
Labels → clarify axes
Legend → identifies plotted data

8. Use of scatter plots in AQI analysis
Shows relationship between two variables.
Example:
PM2.5 vs AQI → if dots align in a trend, stronger relationship.

9. How to change size and color in Matplotlib?
plt.figure(figsize=(10,5))
plt.plot(df['Date'], df['AQI'], color='red')
figsize adjusts plot size.
color changes line color.

10. Why is data visualization important in environmental analysis?
Helps understand pollution trends.
Makes decision-making easier for government & public health bodies.
Identifies peak pollution times and critical zones.

PRACTICAL 6 – Data Aggregation using Pandas

1. What is data aggregation and why is it important?
Aggregation means summarizing data to get meaningful insights.
Helps convert raw data → useful statistics (sum, mean, max, count).

Useful for business reports and dashboards.

2. Which pandas function is used for aggregation?
groupby() is used to group data based on one or more columns.
Example:
df.groupby('Region')['Sales'].sum()

3. How to calculate total sales by region?
df.groupby('Region')['SalesAmount'].sum()
Groups rows by Region and adds up all sales values.

4. Difference between grouping and aggregation (in points)
Grouping:
Splits data into categories (clusters).
Does not perform calculations on its own.
Aggregation:
Applies summary calculations like sum, mean, count on grouped data.

5. How to identify top-performing region?
df.groupby('Region')['Sales'].sum().sort_values(ascending=False)
Region with the highest value is the top performer.

6. Why group by both region and product category?
To analyze which product sells best in which region.
Helps in strategic marketing and stock distribution.

7. Suitable visualizations for showing sales distribution by region
Bar Chart → compare total sales
Pie Chart → show region-wise contribution to total revenue.

8. What is a stacked bar plot and when is it used?
A bar plot where each bar is divided into segments representing categories.
Used to show total value + category breakdown in one chart.
Example: Sales by Region and Product Type.

9. How to handle missing or inconsistent region names?
Clean text formatting:
df['Region'] = df['Region'].str.strip().str.title()
Remove leading/trailing spaces.
Standardize name spellings.

10. How does aggregated sales data help business decisions?
Companies can:
Identify high-performing regions

Allocate inventory & marketing budget effectively
Target low-performing regions for improvement
Makes decision-making data-driven instead of guesswork.

PRACTICAL 7 – Time Series Analysis (Stock Market Data)

1. What is time series data and how is it different from other data?
Time series data is data recorded in a sequence over time.
Time (date/hour) plays a major role in analysis.
Values are dependent on previous time values.
Unlike regular datasets, here trend and patterns change with time.
Example: Stock prices recorded daily.

2. Why convert the date column into datetime format in Python?
To enable date-based operations like sorting, filtering, indexing.
Python recognizes datetime format for:
Time-based grouping (day, month, year)
Rolling/moving averages
Time-series plots
df['Date'] = pd.to_datetime(df['Date'])

3. Which Python libraries are commonly used for time series analysis?
pandas → loading, cleaning, resampling
matplotlib / seaborn → visualization
statsmodels → forecasting models (ARIMA etc.)
numpy → numeric computations

4. Which plot is used to visualize stock price trends over time?
Line plot
It clearly shows how stock price increases or decreases over days.
plt.plot(df['Date'], df['Close'])

5. Purpose of calculating moving averages / rolling averages
To smooth short-term fluctuations.
Highlights long-term trend direction.
df['MA_7'] = df['Close'].rolling(7).mean()
Traders use moving averages to identify buy/sell signals.

6. How to detect seasonality in stock price data?
Plot the data over time and observe repeating patterns.
Use seasonal decomposition from statsmodels.
Use autocorrelation plots to check periodic relationships.

7. What does correlation between stock price and volume indicate?
Shows relationship between trading activity and price movement.
If price increases with high volume → strong market interest.
If price rises but volume is low → movement may be weak or temporary.

8. What is the ARIMA model and why is it used?
ARIMA stands for:
Auto Regressive component (AR) → relationship with past values
Integrated (I) → differencing to remove trends
MA (Moving Average) → accounts for past prediction errors
Used for forecasting future values based on past time-series patterns.

9. Key components of time series
Trend → long-term movement (up or down pattern)
Seasonality → repetitive pattern over intervals (daily/weekly/monthly)
Cyclic behavior → irregular movements affected by economy or events
Random noise → unpredictable fluctuations

10. Why is model evaluation important in forecasting? Which metrics are used?
Evaluates how accurate the forecast is.
Helps choose the best performing model.
Common metrics:
MAE (Mean Absolute Error) → average absolute difference
MSE (Mean Square Error) → punishes larger errors
RMSE → square root of MSE for interpretation
MAPE → percentage error, easy to explain in business terms