**EDA Script for the AIR+WEATHER MERGED DATA SET**

**Import Libraries**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

**Explanation:**

* We use **pandas** to handle data (reading CSV, cleaning, transformations).
* **Matplotlib** and **Seaborn** are for visualization (plots, charts, heatmaps).
* sns.set() and plt.style.use() make plots more readable and visually appealing.

**Interview Tip:**

“I chose Seaborn for statistical plots like boxplots and heatmaps because it’s high-level and integrates well with pandas.”

**2️⃣ Load Dataset**

df = pd.read\_csv("merged\_data.csv")

df.columns = df.columns.str.strip()

df['datetime'] = pd.to\_datetime(df['aqi\_datetime'], dayfirst=True)

df.set\_index('datetime', inplace=True)

**Explanation:**

* Loads your **merged dataset** (Air Quality + Weather).
* Removes any extra spaces in column names to avoid errors.
* Converts AQI datetime into a proper **datetime object** for time-series analysis.
* Sets it as **index** for easy plotting over time.

**Interview Tip:**

“I used the AQI timestamp as the main datetime because air quality is the primary variable I’m analyzing.”

**3️⃣ Data Quality Checks**

df.info()

df.isnull().mean()\*100

df.duplicated().sum()

df.describe()

**Explanation:**

* **info()** → checks data types, non-null counts.
* **isnull().mean()\*100** → % of missing data.
* **duplicated()** → detects duplicate rows.
* **describe()** → basic stats (mean, min, max) to see ranges and outliers.

**Interview Tip:**

“Before analysis, I always check for missing values and duplicates to avoid biased results or errors during modeling.”

**4️⃣ Distribution Analysis**

pollutants = ['pm25', 'pm10', 'no2', 'so2', 'o3', 'co']

df[pollutants].hist(bins=30, figsize=(12,8))

weather = ['temperature','humidity','wind\_speed','pressure']

df[weather].hist(bins=30, figsize=(12,8))

**Explanation:**

* Histograms show the **distribution of pollutants** (PM2.5, PM10, NO2…) and weather features.
* Helps to identify **skewed distributions, outliers, and common ranges**.

**Interview Tip:**

“Histograms help us see if a pollutant exceeds safe limits frequently or if weather factors like wind influence pollutant dispersion.”

**5️⃣ Time-Series Trends**

sns.lineplot(data=df, x=df.index, y='pm25', hue='city')

sns.lineplot(data=df, x=df.index, y='aqi', hue='city')

**Explanation:**

* Plots PM2.5 and AQI over time for **each city**.
* Reveals **seasonal patterns, spikes, or trends** in air quality.

**Interview Tip:**

“Time-series visualization helps policymakers and citizens understand when air pollution peaks, enabling timely interventions.”

**6️⃣ City-wise Comparisons**

sns.boxplot(data=df, x='city', y='pm25')

avg\_pm25 = df.groupby('city')['pm25'].mean()

avg\_pm25.plot(kind='bar')

**Explanation:**

* Boxplots show the **spread of PM2.5 values per city**, including outliers.
* Bar charts show **average PM2.5 per city**.
* Useful for comparing cities’ air quality.

**Interview Tip:**

“Some cities consistently have higher PM2.5, which indicates chronic pollution. Outliers show temporary spikes due to events like fires.”

**7️⃣ Correlation Analysis**

sns.heatmap(df.corr(), annot=True, cmap='coolwarm')

sns.scatterplot(x='wind\_speed', y='pm25', hue='city')

**Explanation:**

* **Heatmap** shows correlations between pollutants and weather features.
* **Scatter plot** explores relationships (e.g., wind speed vs PM2.5).
* Correlation helps identify **factors affecting pollution**.

**Interview Tip:**

“Negative correlation between wind speed and PM2.5 might indicate that wind disperses pollutants.”

**8️⃣ Health Risk Indicators**

df['unhealthy\_pm25'] = df['pm25'] > 25

risk = df.groupby('city')['unhealthy\_pm25'].mean()\*100

risk.plot(kind='bar')

**Explanation:**

* WHO PM2.5 safe limit = 25 µg/m³.
* Creates a boolean column unhealthy\_pm25 for exceedances.
* Computes **% of unhealthy days per city**.
* Important for **public health assessment**.

**Interview Tip:**

“This allows identifying high-risk cities and informing citizens about exposure risks.”

**9️⃣ Rolling Average / Anomaly Detection**

df['pm25\_rolling'] = df.groupby('city')['pm25'].transform(lambda x: x.rolling(3, min\_periods=1).mean())

* Computes a **3-hour rolling average** to smooth spikes.
* Helps detect **anomalous peaks in pollution**.

**Interview Tip:**

“Rolling averages reduce noise and highlight sustained pollution events, useful for alert systems.”

**10️⃣ Export Cleaned Data**

df.to\_csv("merged\_data\_cleaned.csv", index=True)

**Explanation:**

* Saves cleaned and feature-engineered dataset for **dashboarding, predictive modeling, or reporting**.

**Interview Tip:**

“After EDA, I always save a cleaned dataset so downstream tasks like dashboards or ML models can use it reliably.”

**✅ How to Explain in an Interview**

1. **Start with the goal:** “I performed EDA on a merged air quality and weather dataset to understand pollution trends, correlations, and health risks.”
2. **Describe the steps logically:** Data loading → cleaning → distributions → trends → correlations → health risk.
3. **Highlight insights:** e.g., which cities are worst, correlation with weather, spike detection.
4. **Mention feature engineering:** Rolling averages, unhealthy PM2.5 days.
5. **End with purpose:** “This EDA can guide dashboards, alert systems, or predictive modeling.”

**(EDA) of a population dataset, where you clean the data, reshape it, perform analysis, visualize trends, and save the cleaned version for further use (e.g., merging with air quality data or other datasets).**

**Step 1: Import Libraries**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

* **pandas** → for data manipulation and cleaning.
* **matplotlib.pyplot** → for basic plotting.
* **seaborn** → for enhanced visualizations with better aesthetics.

**Interview Tip:** You can say, “I used pandas for cleaning and manipulation, and seaborn/matplotlib for visualizing trends and distributions.”

**Step 2: Load and Clean Dataset**

file\_path = "population\_data.csv"

df = pd.read\_csv(file\_path, skiprows=4)

df = df.dropna(axis=1, how="all")

* **skiprows=4** → The first 4 rows contain metadata, not actual data.
* **dropna(axis=1, how="all")** → Remove columns that are completely empty.

**Interview Tip:** Show that you know about **data cleaning** and handling metadata in real-world datasets.

**Step 3: Reshape Dataset**

df\_long = df.melt(

id\_vars=["Country Name", "Country Code", "Indicator Name", "Indicator Code"],

var\_name="Year",

value\_name="Population"

)

* The original dataset is **wide format** (one column per year).
* **melt()** transforms it into **long format**, which is easier for analysis.
* After this step, each row represents **one country, one year, one population value**.

**Interview Tip:** Emphasize that reshaping to long format is crucial for **grouping, filtering, and plotting trends over time**.

**Step 4: Convert Data Types and Drop Missing Values**

df\_long["Year"] = pd.to\_numeric(df\_long["Year"], errors="coerce")

df\_long["Population"] = pd.to\_numeric(df\_long["Population"], errors="coerce")

df\_long = df\_long.dropna(subset=["Population"])

* Ensure **Year** and **Population** are numeric for plotting and calculations.
* Drop rows where population is missing.

**Interview Tip:** You can explain that **data type consistency** is essential for numeric operations and visualization.

**Step 5: Basic Information**

print("\nNumber of Countries:", df\_long["Country Name"].nunique())

print("\nYears Available:", df\_long["Year"].min(), "to", df\_long["Year"].max())

* Counts **unique countries** in the dataset.
* Finds **range of years** available.
* Helps you understand **coverage and completeness** of your data.

**Interview Tip:** Say, “I first understand the scope of my dataset before analysis.”

**Step 6: Global Population Trend**

global\_trend = df\_long.groupby("Year")["Population"].sum().reset\_index()

* Groups by **Year** and sums population across all countries → gives **global population per year**.
* Then plotted with **seaborn line plot**.

**Interview Tip:** Explain that this gives a **macro view** of global population growth over time.

**Step 7: Top 10 Populated Countries in Latest Year**

latest\_year = df\_long["Year"].max()

latest\_data = df\_long[df\_long["Year"] == latest\_year]

top10 = latest\_data.nlargest(10, "Population")

* Finds the **latest year in data**.
* Filters data for that year.
* Gets **top 10 countries by population**.
* Visualized with a **horizontal bar chart**.

**Interview Tip:** Highlight that this analysis identifies **key countries contributing to global population**, which is insightful for policy, business, and research.

**Step 8: Population Trend for Selected Countries**

countries = ["India", "China", "United States"]

subset = df\_long[df\_long["Country Name"].isin(countries)]

* Filters data for specific countries of interest.
* Plots **line chart** comparing population growth of India, China, and USA over time.

**Interview Tip:** Emphasize that **comparing countries over time** can reveal growth patterns and trends, useful for global health, resource allocation, or environmental studies.

**Step 9: Save Cleaned Dataset**

df\_long.to\_csv("population\_cleaned.csv", index=False)

* Saves the **cleaned and reshaped dataset** for future analysis.
* Ensures reproducibility and **ETL-ready** data for merging with other datasets (like air quality).

**Interview Tip:** You can say, “Saving the cleaned dataset allows me to use it in dashboards, machine learning models, or integrate with other datasets without repeating cleaning steps.”

**How to Explain in an Interview**

1. Start with **dataset understanding** → mention metadata and missing values.
2. Explain **reshaping to long format** → makes analysis easier.
3. Highlight **basic EDA** → total countries, years, global trends.
4. Discuss **specific insights** → top countries, selected country trends.
5. Mention **visualizations** → line plots for trends, bar plots for comparison.
6. Conclude with **saving cleaned data** → ETL/merging readiness.

💡 **Pro Tip:** If asked for improvement suggestions, you can say:

* Could visualize population growth by **continent/region**.
* Could calculate **population growth rates**.
* Could merge with **air quality data** for correlation analysis.

**You don’t *have to* save every cleaned dataset in your ETL pipeline. Whether you save depends on your workflow:**

**🔹 When to Save a Cleaned Dataset**

* **If the dataset is static (like your population data which doesn’t change daily).  
  👉 It makes sense to clean it once and save as population\_cleaned.csv.**
  + **Next time, instead of re-cleaning, you just load the ready file.**
* **If the dataset will be used across multiple scripts/tools (Python, SQL, Power BI).  
  👉 Saved version avoids repeating transformations.**

**🔹 When *Not* to Save**

* **If the dataset is dynamic/real-time (like air quality and weather data).  
  👉 No point in saving intermediate cleaned CSVs, because:**
  + **Data changes hourly/daily.**
  + **Your ETL script will always fetch fresh data, clean it, and directly load into SQL or Power BI.**

**So for Air + Weather data → you don’t save cleaned CSVs. Instead:**

1. **Extract from API.**
2. **Transform (clean, feature engineering).**
3. **Load directly into your database (SQL).**

**🔹 How This Fits Your Project**

* **Population data: clean once, save, and reuse.**
* **Air + Weather data: don’t save; always process live and push into SQL.**
* **Final merged dataset (Air + Weather + Population):**
  + **Usually stored in SQL tables, not as CSV.**
  + **Because your dashboards (Power BI) can connect directly to SQL for live updates.**