

## **Green Zone: □ Level 4. Full Use of Generative AI Allowed**

**UPDATE:** You may complete this assignment individually or in a team of two students.

**Note:** If you experience issues uploading large files to Canvas, please upload your video to YouTube, OneDrive, or Google Drive. Include the shared link in the final document you submit to Canvas.

### **Problem Description**

In our in-class activity, we practiced “getting to know your data” by loading a dataset, checking its structure, computing descriptive statistics, creating plots, and writing short interpretations about patterns and data quality. This assignment extends that workflow: you will build a Python system that can analyze any dataset a user uploads and automatically produce a clear set of useful data insights.

Your system will accept a CSV file as input (required). It may also accept an optional schema / data dictionary file that describes the columns (e.g., name, type, meaning, units, allowed values, missing-value codes). Your system must still run and produce results even if the schema file is not provided.

You have full access to GenAI tools, including cloud-based models (e.g., ChatGPT, Gemini, Claude) and local/open-source LLMs (e.g., Mistral, DeepSeek, Llama, or similar). You may use GenAI for brainstorming, code assistance, and generating narrative insight. However, you must document your GenAI usage by including a prompt/response log in your repository, and you remain responsible for verifying correctness (i.e., do not present unsupported or hallucinated conclusions as facts).

### **What Your System Must Produce**

When run on a dataset, your system must generate a concise EDA output (not just code) that includes:

1. **Dataset Overview**
  - Rows/columns, column names, inferred types, missing-value summary
  - Basic data quality checks (duplicates, columns with one value, high-missing columns)
2. **Descriptive Statistics**
  - For at least one categorical column: frequency counts + percentages
  - For at least two numeric columns (if available): min/max/mean/median/mode + dispersion (std, IQR) + outlier flagging (1.5×IQR rule)
3. **Visualizations**
  - At least 5 plots total (e.g., histogram, boxplot, bar chart, scatterplot, correlation heatmap)
  - Each plot must have a title and labeled axes
4. **Insights (Human-Readable)**
  - 5–10 bullet insights that summarize key patterns, anomalies, and what the dataset suggests
  - A short note on limitations or potential bias (missingness, sampling, coverage, etc.)

### **Deliverables (Submit on Canvas)**

You must create ONE private GitHub repository for this assignment.  
All deliverables (1–3) must be included inside the repository.  
On Canvas, you will submit only the ZIP exported from the repository.

1. **Code** (GitHub repository, subfolder names "src")
  - Python notebook or script(s) that run end-to-end
  - Include requirements.txt (or environment instructions)
  - Include a short README.md with steps to run
2. **Screencast Video (5–7 minutes)**. Subfolder named "video"
  - Demonstrate running your system on two datasets:
    - one you used during development
    - one “new/unseen” dataset
  - Show the generated outputs (tables/plots/insights) and briefly explain how it works
  - **Repository requirement:**
    - Include the video file (.mp4) **or** a video\_link.txt / README section with the share link.
3. **GenAI Prompt Log (Required if you used GenAI)**. Subfolder names "logs"
  - Provide a single file (genai\_log.md or PDF) containing:
    - tool used (ChatGPT/Gemini/Claude/etc.)
    - your prompts and the key responses
    - what you changed/verified before using it

### Constraints / Rules

- Use public, non-sensitive datasets (e.g., data.gov or state/city open data portals).
- If your dataset includes identifiers or sensitive fields, your system should warn and avoid exposing anything private in the report/output.
- Your work must be reproducible: if we run your instructions, we should get the same results.

### Grading Focus (High Level)

- Works end-to-end and produces meaningful output on two datasets
- Correctness of statistics + quality of plots
- Insight quality (not generic)
- Clear demo video + complete GenAI prompt log (if GenAI used)

### Tips:

- **Push your own limits.**  
Don't aim for the minimum. Use this assignment to stretch what you can build and explain.
- **Let the LLM solve the generic case fast, then go beyond it.**  
GenAI can generate a baseline EDA pipeline in minutes. That's only step 1. Your real value is what you add after the generic output.
- **You must understand your code. You must defend every line.**  
You can use GenAI with no restrictions, but you must be able to explain and justify every part of your solution (statistics, plots, cleaning choices, assumptions, and outputs).
- **Surprise yourself with what you can discover.**  
Combine LLM power with your data science knowledge to extract real insights from *any* dataset I can give you (not only the one you tested).
- **Explore different tools and approaches.**  
Try different models (cloud + local), consider "agent-style" workflows (plan -> analyze -> verify -> report), and test different algorithms for outliers, missingness, and pattern discovery.
- **Design for "any dataset."**  
Handle weird column names, mixed types, missing codes (like -999, "NA", "unknown"), date parsing, and datasets with only categorical or only numeric columns.
- **Make your insights non-generic.**  
Insights should reference *specific columns and specific patterns* (counts, percentages, unusual distributions, strong correlations, outliers). Avoid vague statements like "there are missing values."
- **Build a "verification layer."**  
If you use an LLM to write insights, have your code compute the numbers and pass them to the LLM. The model should *never invent values*.
- **Add a clean, repeatable output format.**  
Save results to an /output/ folder (plots + a short markdown or HTML report).  
Reproducibility is part of trust.
- **Demo two datasets that look different.**  
Example: one numeric-heavy dataset (finance/health), one categorical-heavy dataset (surveys/311 requests). This proves "generalization."

"This is my assignment, where I have to do EDA assisted by LLM, Give me code which uses the Gemini API to do EDA on any CSV, generate plots and stats and follow the constraints, then use Gemini to write human-readable insights, suitable for my assignment."

Response:-

## 1. Overview of the solution

You'll build a Python notebook/script that:

- Accepts a CSV path.
- Performs EDA: overview, quality checks, categorical + numeric stats, ≥5 plots.
- Uses the Gemini API to turn computed stats into narrative insights + limitations.
- Saves outputs (plots + a simple report) in an auto\_output/ folder.
- Is easy to call on two different datasets.

All Gemini usage is in one function, and it receives only numbers computed by your code (so the model cannot invent them).

---

## 2. Install dependencies

bash

```
pip install pandas numpy matplotlib seaborn tabulate google-genai
```

---

## 3. Full notebook-style code

You can paste these cells into a fresh Jupyter notebook and run top to bottom.

### 3.1 Imports and global config

python

```
!pip install google-genai seaborn tabulate --quiet
```

```
import os
```

```
from pathlib import Path
```

```
from datetime import datetime
```

```
import json
```

```
import numpy as np
```

```
import pandas as pd
```

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

```
import seaborn as sns
```

```
from IPython.display import display
```

```
from tabulate import tabulate
```

```
from google import genai
```

```

sns.set(style="whitegrid")

pd.set_option("display.max_columns", 200)

pd.set_option("display.width", 140)

BASE_OUTPUT = Path("auto_output")

BASE_OUTPUT.mkdir(parents=True, exist_ok=True)

MISSING_FLAG_THRESHOLD = 0.5  # >50% missing -> high missing
CAT_MAX_LEVELS = 20          # treat as categorical if <= 20 unique
DATETIME_SUCCESS_RATIO = 0.75 # 75% parseable -> datetime
LLM_MODEL = "gemini-3-flash-preview"

```

---

### 3.2 Data loading and basic cleaning

python

```

DEFAULT_MISSING_TOKENS = {
    "", " ", "na", "n/a", "nan", "null", "none", "unknown", "?",
    "-", "--", "missing", "-999", "-9999", "999", "9999"
}

```

**def** read\_dataset(csv\_path: str) -> pd.DataFrame:

```

"""
Load a CSV, strip column names.
"""

df = pd.read_csv(csv_path, low_memory=False)

df.columns = [str(c).strip() for c in df.columns]

return df

```

**def** normalize\_missing(df: pd.DataFrame) -> pd.DataFrame:

```

"""
Replace common missing-value tokens with NaN and convert

```

object columns that are mostly numeric into real numeric.

```
"""
```

```
tokens = {t.lower() for t in DEFAULT_MISSING_TOKENS}
```

```
for col in df.columns:
```

```
    if df[col].dtype == "object":
```

```
        s = df[col].astype(str).str.strip()
```

```
        df[col] = s.mask(s.str.lower().isin(tokens))
```

```
for col in df.columns:
```

```
    if df[col].dtype == "object":
```

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

```
        if num.notna().mean() > 0.9:
```

```
            df[col] = num
```

```
return df
```

---

### 3.3 Type inference and quality checks

python

```
def infer_roles(df: pd.DataFrame) -> dict:
```

```
    """
```

```
    Infer roles: numeric, datetime, categorical, text, other.
```

```
    """
```

```
    roles = {}
```

```
    n = len(df)
```

```
    for col in df.columns:
```

```
        s = df[col]
```

```
        if pd.api.types.is_numeric_dtype(s):
```

```
            roles[col] = "numeric"
```

```
            continue
```

```

if s.dtype == "object":
    parsed = pd.to_datetime(s, errors="coerce", infer_datetime_format=True)
    if parsed.notna().mean() >= DATETIME_SUCCESS_RATIO:
        df[col] = parsed
        roles[col] = "datetime"
        continue

    nunique = s.nunique(dropna=True)
    if nunique <= CAT_MAX_LEVELS:
        roles[col] = "categorical"
    elif nunique > 0.5 * n:
        roles[col] = "text"
    else:
        roles[col] = "categorical"
else:
    roles[col] = "other"

return roles

```

```

def run_quality_checks(df: pd.DataFrame, roles: dict) -> dict:

```

```

    """

```

```

    Compute row/column counts, missingness, duplicates,
    constant columns, high-missing columns, id-like columns.

```

```

    """

```

```

    n_rows, n_cols = df.shape
    dup_rows = int(df.duplicated().sum())
    miss_cnt = df.isna().sum()
    miss_frac = df.isna().mean()

```

```

const_cols = [c for c in df.columns if df[c].nunique(dropna=True) <= 1]
high_missing = [c for c in df.columns if miss_frac[c] > MISSING_FLAG_THRESHOLD]

id_like = []

for c in df.columns:
    if roles.get(c) != "numeric":
        if df[c].nunique(dropna=True) > 0.9 * n_rows:
            id_like.append(c)

return {
    "n_rows": int(n_rows),
    "n_cols": int(n_cols),
    "duplicate_rows": dup_rows,
    "missing_count": miss_cnt.to_dict(),
    "missing_fraction": miss_frac.to_dict(),
    "constant_cols": const_cols,
    "high_missing_cols": high_missing,
    "id_like_cols": id_like,
    "missing_threshold": MISSING_FLAG_THRESHOLD,
}

```

---

### 3.4 Descriptive statistics

python

**def** pick\_categorical\_column(df: pd.DataFrame, roles: dict) -> str | None:

"""

Choose a 'nice' categorical column (limited unique values).

"""

candidates = [c for c, r in roles.items() if r == "categorical"]

**if not** candidates:

**return** None

candidates = sorted(

```

    candidates,

    key=lambda c: (df[c].nunique(dropna=True), -df[c].notna().sum())
)
return candidates[0]

```

```

def categorical_profile(df: pd.DataFrame, col: str, top_k: int = 15) -> pd.DataFrame:

```

```

    vc = df[col].value_counts(dropna=False)
    total = vc.sum()
    out = (
        vc.to_frame("count")
        .assign(percent=lambda d: (d["count"] / total * 100).round(2))
        .reset_index()
        .rename(columns={"index": col})
    )
    return out.head(top_k)

```

```

def numeric_profiles(df: pd.DataFrame, roles: dict) -> list[dict]:

```

```

    """
    For each numeric column: min/max/mean/median/mode, std, IQR, and outliers via 1.5*IQR.
    """

    stats = []
    for col, r in roles.items():
        if r != "numeric":
            continue

        s = df[col].dropna()
        if s.empty:
            continue

        q1 = s.quantile(0.25)

```

```

q3 = s.quantile(0.75)
iqr = q3 - q1
lower = q1 - 1.5 * iqr
upper = q3 + 1.5 * iqr
outliers = int(((s < lower) | (s > upper)).sum())
mode_vals = s.mode()
mode_val = float(mode_vals.iloc[0]) if not mode_vals.empty else None

```

```

stats.append(
    {
        "column": col,
        "count": int(s.count()),
        "mean": float(s.mean()),
        "std": float(s.std()),
        "min": float(s.min()),
        "q1": float(q1),
        "median": float(s.median()),
        "q3": float(q3),
        "max": float(s.max()),
        "iqr": float(iqr),
        "lower_1_5_iqr": float(lower),
        "upper_1_5_iqr": float(upper),
        "n_outliers_1_5_iqr": outliers,
        "mode": mode_val,
    }
)
return stats

```

---

### 3.5 Plot generation (≥5 plots, with titles and labels)

python

```
def generate_plots(df: pd.DataFrame, roles: dict, dataset_name: str) -> list[Path]:
```

```
"""
```

Create at least 5 plots: histogram, boxplot, bar chart, scatter, correlation heatmap.

Save them under `auto_output/<dataset_name>/plots` and also show inline.

```
"""
```

```
plot_dir = BASE_OUTPUT / dataset_name / "plots"
```

```
plot_dir.mkdir(parents=True, exist_ok=True)
```

```
numeric_cols = [c for c, r in roles.items() if r == "numeric"]
```

```
cat_cols = [c for c, r in roles.items() if r == "categorical"]
```

```
paths: list[Path] = []
```

```
# 1) Histogram
```

```
if numeric_cols:
```

```
    col = numeric_cols[0]
```

```
    fig, ax = plt.subplots(figsize=(6, 4))
```

```
    sns.histplot(df[col], kde=True, ax=ax)
```

```
    ax.set_title(f"Histogram of {col}")
```

```
    ax.set_xlabel(col)
```

```
    ax.set_ylabel("Count")
```

```
    p = plot_dir / f"{dataset_name}_hist_{col}.png"
```

```
    fig.tight_layout()
```

```
    fig.savefig(p, dpi=130)
```

```
    plt.show()
```

```
    paths.append(p)
```

```
# 2) Boxplot
```

```
if numeric_cols:
```

```
    col = numeric_cols[0]
```

```
    fig, ax = plt.subplots(figsize=(4, 4))
```

```
    sns.boxplot(x=df[col], ax=ax)
```

```
ax.set_title(f"Boxplot of {col}")
ax.set_xlabel(col)
p = plot_dir / f"{dataset_name}_box_{col}.png"
fig.tight_layout()
fig.savefig(p, dpi=130)
plt.show()
paths.append(p)
```

### *# 3) Bar chart*

```
if cat_cols:
    col = cat_cols[0]
    vc = df[col].value_counts().head(15)
    fig, ax = plt.subplots(figsize=(7, 4))
    sns.barplot(x=vc.index.astype(str), y=vc.values, ax=ax)
    ax.set_title(f"Top categories in {col}")
    ax.set_xlabel(col)
    ax.set_ylabel("Count")
    ax.tick_params(axis="x", rotation=45)
    p = plot_dir / f"{dataset_name}_bar_{col}.png"
    fig.tight_layout()
    fig.savefig(p, dpi=130)
    plt.show()
    paths.append(p)
```

### *# 4) Scatter*

```
if len(numeric_cols) >= 2:
    xcol, ycol = numeric_cols[:2]
    fig, ax = plt.subplots(figsize=(6, 4))
    sns.scatterplot(x=df[xcol], y=df[ycol], ax=ax, alpha=0.6)
    ax.set_title(f"{xcol} vs {ycol}")
    ax.set_xlabel(xcol)
```

```

ax.set_ylabel(ycol)

p = plot_dir / f"{dataset_name}_scatter_{xcol}_vs_{ycol}.png"

fig.tight_layout()

fig.savefig(p, dpi=130)

plt.show()

paths.append(p)

# 5) Correlation heatmap
if len(numeric_cols) >= 2:
    corr = df[numeric_cols].corr()

    fig, ax = plt.subplots(figsize=(max(6, len(numeric_cols)), 6))

    sns.heatmap(corr, annot=True, fmt=".2f", cmap="coolwarm", ax=ax)

    ax.set_title("Correlation heatmap")

    p = plot_dir / f"{dataset_name}_corr.png"

    fig.tight_layout()

    fig.savefig(p, dpi=130)

    plt.show()

    paths.append(p)

return paths

```

---

### 3.6 Gemini client and insight generation

python

```

def build_gemini_client() -> genai.Client:
    """
    Build a Gemini client. Either set GEMINI_API_KEY in env,
    or hardcode your key here for local testing.
    """
    api_key = os.getenv("GEMINI_API_KEY")

    if not api_key:
        raise RuntimeError("Set GEMINI_API_KEY environment variable first.")

```

```
return genai.Client(api_key=api_key)
```

```
def make_insights_with_llm(
```

```
    client: genai.Client,
```

```
    dataset_name: str,
```

```
    overview: dict,
```

```
    roles: dict,
```

```
    quality: dict,
```

```
    cat_col: str | None,
```

```
    cat_table: pd.DataFrame | None,
```

```
    numeric_stats: list[dict],
```

```
) -> dict:
```

```
    """
```

Call Gemini with a JSON payload of stats and ask for:

- 5–10 bullet insights.

- 1 paragraph of limitations/bias.

The model sees only precomputed numbers and should not invent new ones.

```
    """
```

```
    payload = {
```

```
        "dataset": dataset_name,
```

```
        "overview": overview,
```

```
        "roles": roles,
```

```
        "quality": quality,
```

```
        "categorical": {
```

```
            "column": cat_col,
```

```
            "table": cat_table.to_dict(orient="records") if cat_table is not None else None,
```

```
        },
```

```
        "numeric": numeric_stats,
```

```
    }
```

```
prompt = f"""
```

You are summarizing an exploratory data analysis (EDA).

All numbers are already computed in the JSON below. Do NOT invent new numeric values.

Return:

- 5–10 bullet-point insights referencing specific columns and numbers.
- 1 short paragraph on limitations or potential bias (missingness, sampling, identifiers, etc.).

Here is the JSON:

```
{json.dumps(payload, indent=2)}
```

```
"""
```

```
resp = client.models.generate_content(  
    model=LLM_MODEL,  
    contents=prompt,  
)
```

```
text = resp.text or ""
```

```
lines = [l.strip() for l in text.splitlines() if l.strip()]  
bullets = [l.strip("-*• ") for l in lines if l.startswith(("-", "*", "•"))]  
other = [l for l in lines if not l.startswith(("-", "*", "•"))]
```

```
bullets = bullets[:10]
```

```
while len(bullets) < 5:
```

```
    bullets.append("More detailed insights would require additional numeric or categorical  
features.")
```

```
limitations = " ".join(other) if other else "Results may be limited by missing data, sampling, and  
potential identifier-like columns."
```

```
return {"insights": bullets, "limitations": limitations}
```

---

### 3.7 Orchestrator that runs end-to-end and prints output

python

```
def full_eda_pipeline(csv_path: str, dataset_name: str):  
    print(f"Running EDA for {dataset_name} from {csv_path}")  
  
    # 1. Load and clean  
    df = read_dataset(csv_path)  
    df = normalize_missing(df)  
  
    print("\nData preview:")  
    display(df.head())  
  
    # 2. Roles and quality  
    roles = infer_roles(df)  
    print("\nInferred roles:")  
    print(roles)  
  
    quality = run_quality_checks(df, roles)  
    print("\nQuality checks (missing counts and %):")  
    qual_df = pd.DataFrame({  
        "missing_count": quality["missing_count"],  
        "missing_pct": {k: v * 100 for k, v in quality["missing_fraction"].items()}  
    })  
    display(qual_df)  
  
    # 3. Descriptive stats  
    cat_col = pick_categorical_column(df, roles)  
    if cat_col:  
        cat_table = categorical_profile(df, cat_col)  
        print(f"\nCategorical summary for '{cat_col}':")
```

```

        display(cat_table)
    else:
        cat_table = None

        print("\nNo suitable categorical column found.")

numeric_stats = numeric_profiles(df, roles)
print("\nNumeric summaries:")
display(pd.DataFrame(numeric_stats))

# 4. Plots
print("\nGenerating plots...")
plot_paths = generate_plots(df, roles, dataset_name)
print("Saved plots:")
for p in plot_paths:
    print("-", p)

# 5. Gemini insights
overview = {
    "n_rows": df.shape[0],
    "n_cols": df.shape[1],
    "columns": list(df.columns),
}

print("\nCalling Gemini for insights...")
client = build_gemini_client()
llm_block = make_insights_with_llm(
    client,
    dataset_name=dataset_name,
    overview=overview,
    roles=roles,
    quality=quality,

```

```

        cat_col=cat_col,
        cat_table=cat_table,
        numeric_stats=numeric_stats,
    )

    print("\nInsights:")
    for b in llm_block["insights"]:
        print("-", b)

    print("\nLimitations:")
    print(llm_block["limitations"])

# 6. Save simple text report
ds_dir = BASE_OUTPUT / dataset_name
ds_dir.mkdir(parents=True, exist_ok=True)
report_path = ds_dir / "eda_report.txt"
with report_path.open("w", encoding="utf-8") as f:
    f.write("INSIGHTS\n")
    for b in llm_block["insights"]:
        f.write(f"- {b}\n")
    f.write("\nLIMITATIONS\n")
    f.write(llm_block["limitations"])
    print(f"\nReport saved to: {report_path}")

```

---

### 3.8 Example: run on two datasets

python

*# Development dataset*

```
full_eda_pipeline("india_population.csv", dataset_name="india_population")
```

*# Unseen dataset (put another CSV in the same folder, e.g., chocolatesales.csv)*

```
full_eda_pipeline("other_dataset.csv", dataset_name="other_dataset")
```

---

#### 4. Environment setup for the Gemini key

Before calling the pipeline, set your Gemini key either:

In the shell:

```
bash
```

```
export GEMINI_API_KEY="YOUR_REAL_GEMINI_API_KEY"
```

Or inside the notebook (one time per runtime):

```
python
```

```
import os
```

```
os.environ["GEMINI_API_KEY"] = "YOUR_REAL_GEMINI_API_KEY"
```