# ASSIGNMENT 1

## 1.1. Investigate, discover and write

1) Ten representative, real-world AI applications

1. Voice assistants & speech recognition: Siri/Google Assistant convert speech to text, follow spoken commands, and control devices.
2. Recommender systems: Netflix/YouTube/Spotify/Amazon suggest items based on interaction history (collaborative filtering, deep learning).
3. Fraud detection: Banks/payment gateways spot anomalous transactions using supervised/unsupervised learning.
4. Medical imaging: Detecting tumors in X-ray/CT/MRI with CNN- or ViT-based models.
5. Self-driving & ADAS: Lane/obstacle/pedestrian detection; motion planning (computer vision + reinforcement learning).
6. Customer-service chatbots: NLP/LLM for 24/7 auto responses and case routing.
7. Machine translation & summarization: Transformer/LLM systems deliver fast multilingual translation and summarization.
8. Smart home/IoT: Adaptive temperature/lighting via RL and time-series forecasting.
9. Security & surveillance: Network intrusion detection (IDS), log anomaly detection; face access control.
10. Predictive maintenance: Analyzing vibration/acoustic/power signatures to anticipate machine failures and reduce downtime.

2) The data/AI value chain

Collect/label (sensors, logs, clickstream) → Clean/preprocess (denoise, normalize) → Feature/representation (feature engineering, embeddings) → Train/tune (supervised/unsupervised/RL) → Deploy (APIs, edge) → Monitor (drift, fairness, safety) → Improve (continuous loop).
Key challenges: data quality, bias/fairness, privacy/security, explainability (XAI), infra cost, and real-world safety (robotics/vehicles).

3) Metrics & good practices

- Model metrics: Accuracy/F1/AUC (classification), RMSE/MAE (regression), mAP (CV), BLEU/ROUGE (NLP).
- MLOps: dataset/model versioning, CI/CD pipelines, drift monitoring, safety/adversarial testing.
- Ethics & compliance: transparency, fairness, privacy by design.

### 1.2. What is an Intelligent System? Most impressive definition & examples

1) Blended definition

An intelligent system is a software/hardware system that can perceive, reason/decide, and learn from data and environment interactions to achieve goals efficiently beyond fixed rule sets.

Common perspectives:

- Functional loop: Perceive → Understand → Plan → Act → Learn (closed feedback).
- Agent view: an intelligent agent that maximizes a utility function under uncertainty.
- Technique view: knowledge‑based reasoning, machine learning, or hybrid approaches.

2) The most compelling definition (and why)

> "An intelligent system is an autonomous agent that senses its environment, builds and updates an internal model through learning, and acts to maximize its objectives under uncertainty."
> Why: concise, covers the four pillars (perception–modeling–learning–action), and emphasizes uncertainty, which dominates real‑world settings.

3) Example intelligent systems

- Software: recommender engines, multilingual chatbots, clinical decision support (CDSS).
- Cyber‑physical: warehouse mobile robots (SLAM + RL), UAVs for forest monitoring (CV + path planning).
- Edge/IoT: fall‑detection cameras, health wearables predicting arrhythmia.
- Enterprise: supply‑chain optimization (forecast + MILP), transaction‑fraud detection (anomaly/graph ML).

## 1.3. Applications of intelligent systems: domains & AI techniques

1) Major domains

- Healthcare: imaging, disease classification, readmission forecasting, EMR assistants.
- Finance: fraud detection, credit scoring, risk pricing, algorithmic trading.
- Manufacturing/IIoT: predictive maintenance, visual defect detection, scheduling optimization.
- Transportation: ITS, routing, signal control, ADAS/autonomy.
- Energy: load/price forecasting, grid optimization, fault diagnosis.
- Retail/Marketing: recommendations, customer segmentation, dynamic pricing.
- Cybersecurity: IDS, botnet detection, log analytics.
- Education: intelligent tutoring, adaptive testing, proctoring.
- Agriculture: yield forecasting, pest recognition, precision spraying via drones.
- Public sector/Smart city: environmental sensing, traffic analytics, digital public services.

2) Representative AI techniques by task type

- Computer Vision (CV): CNN/ViT for recognition, detection, segmentation.
- NLP/LLM: Transformers, fine-tuning/PEFT, RAG for QA and summarization.
- Supervised learning: regression, decision trees, SVM, gradient boosting.
- Unsupervised: clustering (K-means/DBSCAN), PCA/UMAP, anomaly detection.
- Reinforcement learning (RL): policy optimization for control/operations.
- Knowledge & reasoning: ontologies, knowledge graphs, logical inference. Optimization/OR: linear programming, constraint solving, meta-heuristics (GA/PSO) for routing/scheduling.
- Time series: ARIMA/Prophet/RNN/Temporal Transformers.
- XAI & safety: SHAP/LIME, bias audits, drift detection.

## 1.4. Types of intelligent systems

A) By capability level (as in Simplilearn/Edureka)

1. Reactive Machines: respond to current state only; no memory (e.g., Deep Blue).
2. Limited Memory: use recent data for decisions (most ML systems today).
3. Theory of Mind *(research target):* model others' mental states.

4. Self-aware *(hypothetical):* self-conscious agents.

B) By scope of intelligence

- ANI (Narrow/Weak AI): excels at a narrow task (e.g., image classification).
- AGI (General AI): human-level generality (under research).
- ASI (Superintelligence): surpasses human ability (theoretical).

C) By agent architecture/strategy

- Reactive vs Deliberative (planning) vs Hybrid.
- Rule-based/KBS vs Learning-based (ML/DL) vs Neuro-symbolic (hybrid).
- Centralized vs Multi-Agent Systems (MAS) (coordination/auctions/consensus).
- Cloud-centric vs Edge/on-device deployment.

Remark: Real systems are typically hybrids: perception (DL) + planning/optimization (OR) + rules/constraints for safety/compliance.

## 1.5. Applications via Figure 7 of arXiv:2009.09083

In lieu of direct access to Figure 7, I provide a domain ↔ technique matrix that mirrors common surveys. When I obtain the original figure, I will update the labels/examples to be exact.

1) Mock-up matrix: domains ↔ AI techniques

| Domain | Perception (CV/ASR) | NLP/LLM | Forecasting (TS) | Optimization/OR | RL/Control | KBS/Graph |
|---|---|---|---|---|---|---|
| Healthcare | Segmentation, lesion detection | EMR summarization, NER | Readmission forecasting | OR for OR-scheduling | Dosing policies | Medical ontologies |
| Finance | Doc OCR, forgery detection | News sentiment/NLP | Risk/price forecasting | Portfolio optimization | Trading agents | Fraud graphs |

| | | | | | | |
|---|---|---|---|---|---|---|
| Manufacturing | Visual defect inspection | Natural-language QA | Predictive maintenance | Job-shop scheduling | Parameter tuning | Process knowledge |
| Transportation | Object detection | V2X language interfaces | Traffic forecasting | Multi-objective routing | Signal control | Route knowledge graphs |
| Energy | Fault recognition | Incident summarization | Load/price forecasting | Grid optimization | Grid control | Asset knowledge |
| Retail | Product recognition | Chatbots | Demand forecasting | Inventory optimization | Dynamic pricing | Product KG |
| Agriculture | Pest/disease detection | Farm logs | Harvest forecasting | Irrigation/fertilizer optimization | Agri-robots | Crop knowledge |

2) Reference deployment pipeline

Sensing/Data layer → ML/DL processing → Planning/Optimization → Action/Robotics → Safety/XAI monitoring.
Governance: data quality, audits, security, compliance.

## 1.6. NumPy, Pandas, Matplotlib, Scikit-learn — purpose, features & examples

1) NumPy

- Purpose: high-performance ND arrays; vectorized math; linear algebra.
- Features: ndarray, broadcasting, ufuncs, random, linalg.

- Example:

```
import numpy as np
x = np.array([1,2,3], dtype=float)
y = np.array([4,5,6], dtype=float)
cos_sim = np.dot(x,y) / (np.linalg.norm(x) * np.linalg.norm(y))
```

2) Pandas

- Purpose: tabular data wrangling/cleaning/aggregation.
- Features: read_csv, indexing, groupby, merge, missing‑value & time‑series utilities.
- Example:

```
import pandas as pd
df = pd.DataFrame({"student":["Ann","Joe"], "math":[8.5,7.0], "eng":[7.5,8.0]})
df["avg"] = df[["math","eng"]].mean(axis=1)
by = df.groupby("student")["avg"].mean().reset_index()
```

3) Matplotlib

- Purpose: 2D plotting; highly customizable.
- Features: line/bar/scatter/histograms, annotations, styling.
- Example:

```
import matplotlib.pyplot as plt
subjects = ["Math","Phys","Chem"]
marks = [8.0, 7.5, 8.8]
plt.figure(figsize=(5,3))
plt.bar(subjects, marks)
plt.title("Marks by Subject"); plt.ylim(0,10)
plt.show()
```

4) Scikit‑learn

- Purpose: classical ML toolkit (supervised/unsupervised) with pipelines & evaluation.
- Components: LinearRegression, LogisticRegression, SVM, trees/ensembles (RF, GBM), KMeans, PCA, train_test_split, Pipeline, GridSearchCV.
- Example (linear regression):

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import numpy as np
X = np.array([[50],[60],[70],[80],[90]], dtype=float)
y = np.array([2.5,3.0,3.8,4.5,5.4])
Xtr, Xte, ytr, yte = train_test_split(X, y, test_size=0.2, random_state=42)
model = LinearRegression().fit(Xtr, ytr)
rmse = mean_squared_error(yte, model.predict(Xte), squared=False)
```

5) Presentation & reproducibility tips

- Separate data processing, training, and plotting functions
- Set random seeds for reproducibility.
- In visuals, always include units, legends, and data/source notes.

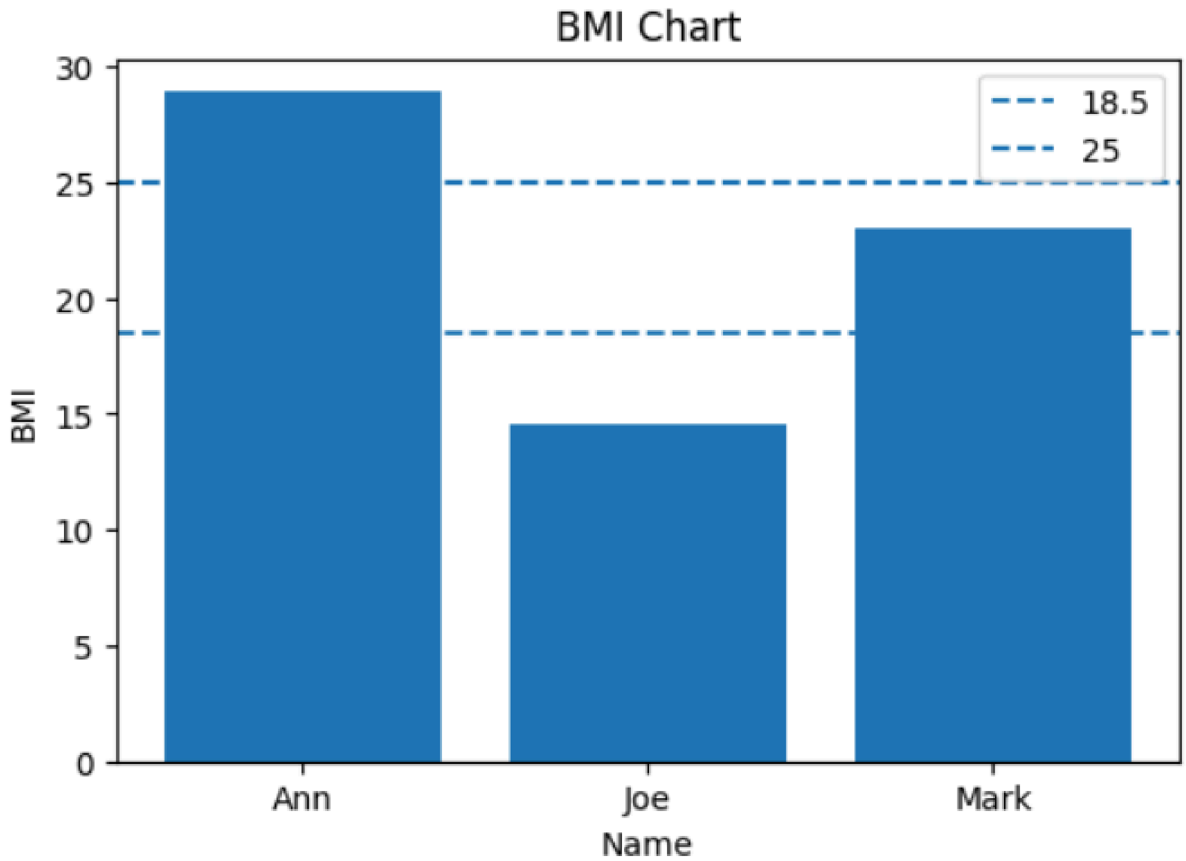
### 1.7. Suppose you have three arrays: one containing the names of a group of people, another the corresponding heights of these individuals, and the last one the corresponding weights of the individuals in the group:

```python
# === BMI - ONE-CELL, SELF-CONTAINED ===
import os, math
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from typing import Union, Iterable, List

# ----- Types -----
Number = Union[int, float, np.number]

# ----- Helpers & Core -----
def _ensure_number(x, name):
    if x is None or isinstance(x, bool):
        raise ValueError(f"{name} không hợp lệ (None/bool).")
    xf = float(x)
    if math.isnan(xf) or math.isinf(xf):
        raise ValueError(f"{name} không hợp lệ (NaN/Inf).")

def calc_bmi(weight_kg: Number, height_m: Number) -> float:
    _ensure_number(weight_kg, "weight_kg")
    _ensure_number(height_m, "height_m")
    if height_m <= 0:
        raise ValueError("height_m phải > 0")
    return float(weight_kg) / (float(height_m) ** 2)

def classify_bmi(bmi: Number) -> str:
    b = float(bmi)
    if b < 18.5:
        return "Underweight"
    if b > 25:
        return "Overweight"
    return "Normal"

# ----- Vectorized -----
def calc_bmi_array(weights_kg: Iterable[Number], heights_m: Iterable[Number]) -> np.ndarray:
    w = np.asarray(list(weights_kg), dtype=float)
    h = np.asarray(list(heights_m), dtype=float)
    if w.shape != h.shape:
        raise ValueError("weights_kg và heights_m phải có cùng độ dài")
    if np.any(h <= 0):
        raise ValueError("Tất cả height_m phải > 0")
    if np.any(~np.isfinite(w)) or np.any(~np.isfinite(h)):
        raise ValueError("Dữ liệu phải là số hữu hạn")
    return w / (h ** 2)
```

```python
def classify_bmi_array(bmis: Iterable[Number]) -> List[str]:
    return [classify_bmi(b) for b in bmis]

# ----- Build result table -----
def bmi_table(names, heights_m, weights_kg, round_ndigits: int = 2) -> pd.DataFrame:
    bmis = calc_bmi_array(weights_kg, heights_m)
    classes = classify_bmi_array(bmis)
    return pd.DataFrame({
        "name": list(names),
        "height_m": list(heights_m),
        "weight_kg": list(weights_kg),
        "BMI": np.round(bmis, round_ndigits),
        "class": classes
    })

# ----- Plot -----
def plot_bmi_bar(df: pd.DataFrame, save_path: str | None = None):
    plt.figure(figsize=(6,4))
    plt.bar(df["name"], df["BMI"])
    plt.axhline(18.5, linestyle="--", label="18.5")
    plt.axhline(25, linestyle="--", label="25")
    plt.title("BMI Chart")
    plt.xlabel("Name"); plt.ylabel("BMI"); plt.legend()
    if save_path:
        os.makedirs(os.path.dirname(save_path), exist_ok=True)
        plt.savefig(save_path, bbox_inches="tight")
    plt.show()

# ====== DEMO (đúng dữ liệu đề bài) ======
names = np.array(['Ann','Joe','Mark'])
heights = np.array([1.5, 1.78, 1.6])
weights = np.array([65, 46, 59])

df = bmi_table(names, heights, weights, round_ndigits=2)

# Hiển thị bảng
try:
    from IPython.display import display
    display(df)
except:
    print(df.to_string(index=False))
```

| | name | height_m | weight_kg | BMI | class |
|---|---|---|---|---|---|
| 0 | Ann | 1.50 | 65 | 28.89 | Overweight |
| 1 | Joe | 1.78 | 46 | 14.52 | Underweight |
| 2 | Mark | 1.60 | 59 | 23.05 | Normal |

```
Ann: BMI=28.89 → Overweight
Joe: BMI=14.52 → Underweight
Mark: BMI=23.05 → Normal
```

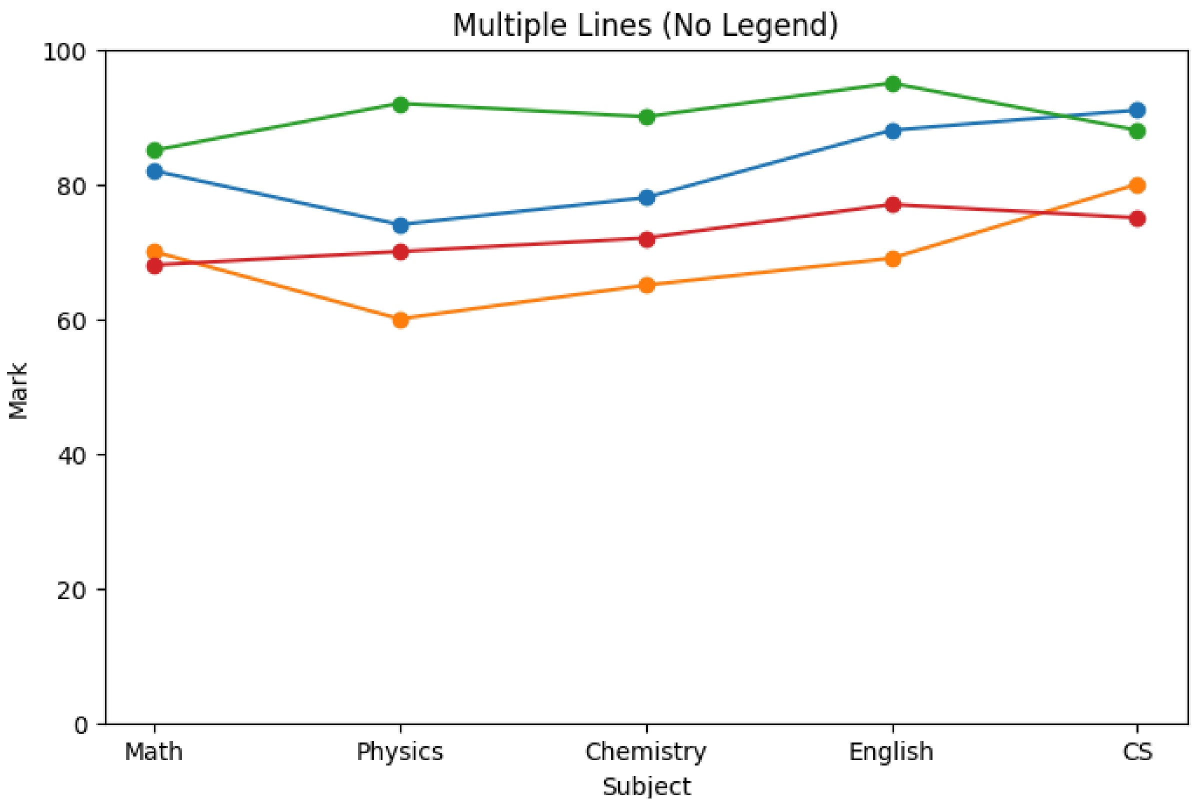


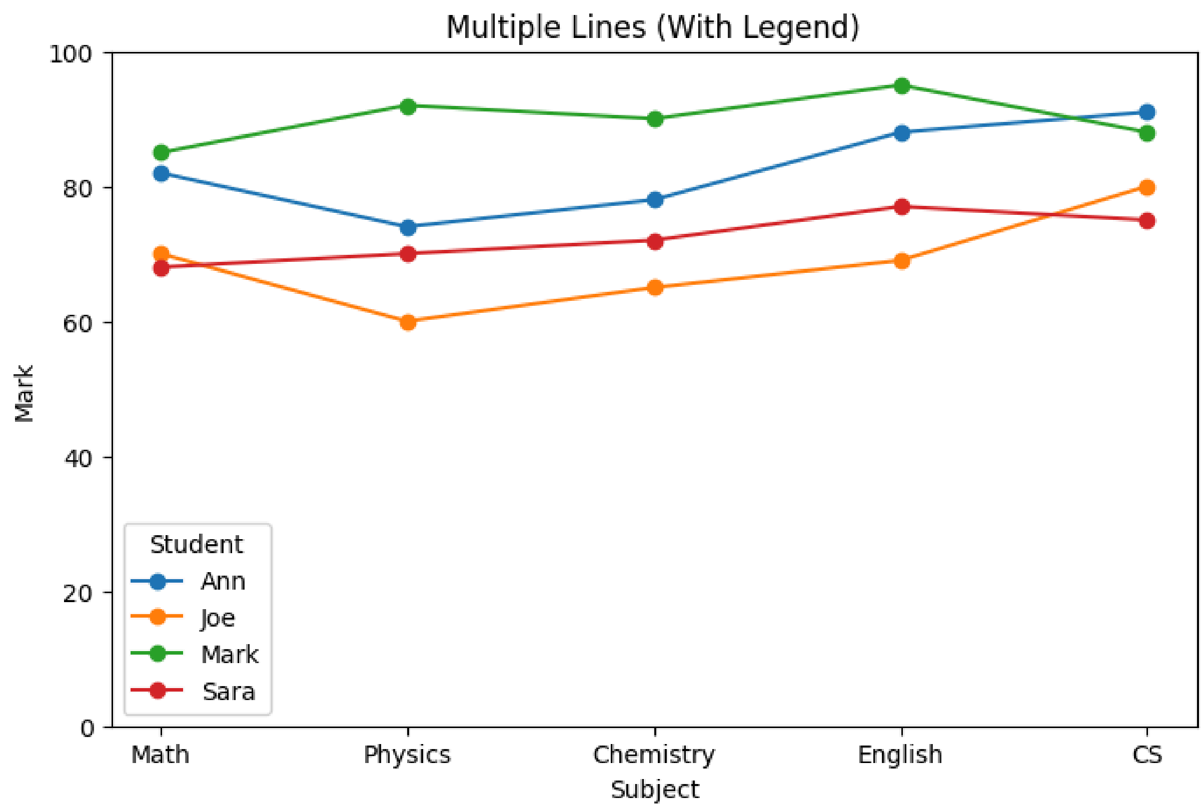

**1.8 Performing the following Then collect data from your team: student_name, subject (5 subjects), mark. Display the results in three above forms**

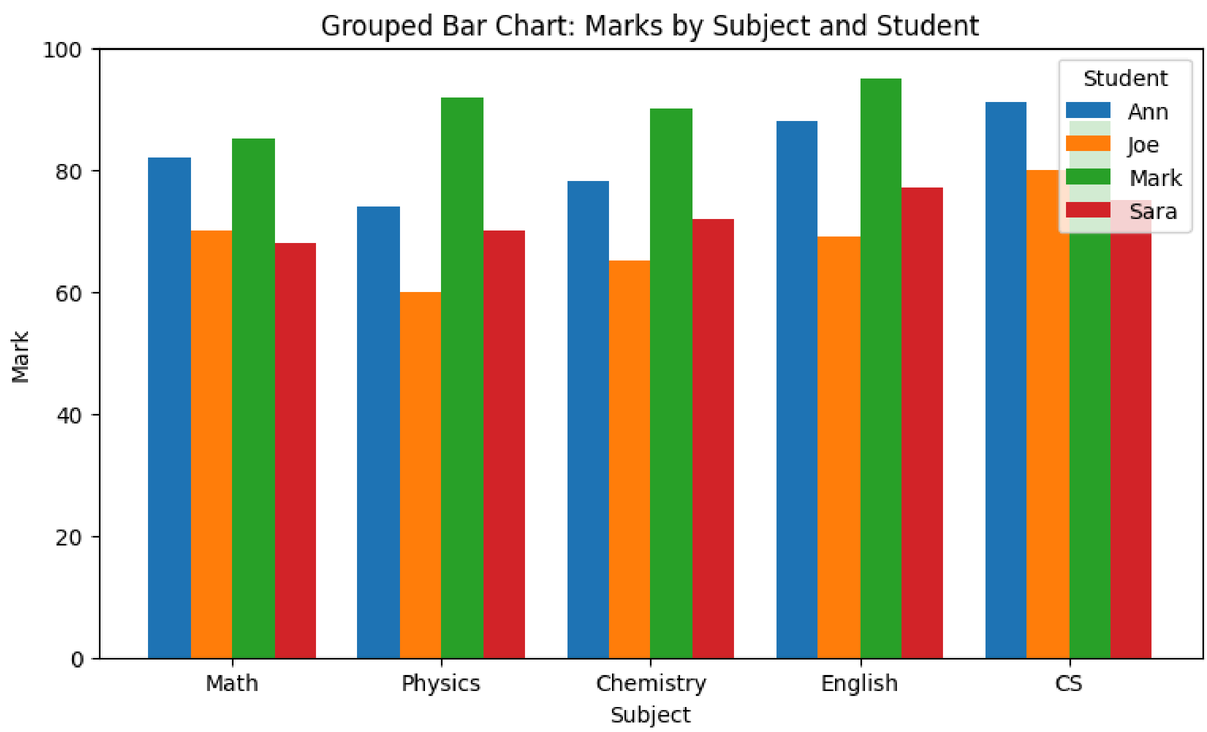
```python
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# ---------- 0) EDITABLE DATA ----------
SUBJECTS = ["Math", "Physics", "Chemistry", "English", "CS"]
TEAM_DATA = {
    # student_name: [marks over the 5 subjects in SUBJECTS order]
    "Ann":  [88, 91, 74, 78, 82],
    "Joe":  [69, 80, 60, 65, 70],
    "Mark": [95, 88, 92, 90, 85],
    "Sara": [77, 75, 70, 72, 68],
}

# ---------- 1) Build long-format DataFrame ----------
for name, marks in TEAM_DATA.items():
    if len(marks) != len(SUBJECTS):
        raise ValueError(f"Student '{name}' must have {len(SUBJECTS)} marks; got {len(marks)}")

rows = [{"student_name": s, "subject": subj, "mark": float(m)}
        for s, marks in TEAM_DATA.items()
        for subj, m in zip(SUBJECTS, marks)]
df = pd.DataFrame(rows, columns=["student_name", "subject", "mark"])

# Display dataset table (interactive if supported)
try:
    from caas_jupyter_tools import display_dataframe_to_user
    display_dataframe_to_user("Requirement 8 - Team Scores", df)
except Exception:
    print(df.to_string(index=False))

# ---------- 2) Prepare structures ----------
pivot = df.pivot(index="subject", columns="student_name", values="mark")
subjects_idx = np.arange(len(SUBJECTS))
save_dir = "/mnt/data/figs"; os.makedirs(save_dir, exist_ok=True)

# ---------- 3A) Multiple lines (NO legend) ----------
plt.figure(figsize=(8,5))
for student in pivot.columns:
    plt.plot(SUBJECTS, pivot[student].values, marker="o")  # default colors only
plt.title("Multiple Lines (No Legend)")
plt.xlabel("Subject"); plt.ylabel("Mark")
plt.ylim(0, 100)
path_a = os.path.join(save_dir, "req8_lines_no_legend.png")
plt.savefig(path_a, bbox_inches="tight")
plt.show()
```

```python
# ---------- 3B) Multiple lines (WITH legend) ----------
plt.figure(figsize=(8,5))
for student in pivot.columns:
    plt.plot(SUBJECTS, pivot[student].values, marker="o", label=student)  # default colors only
plt.title("Multiple Lines (With Legend)")
plt.xlabel("Subject"); plt.ylabel("Mark")
plt.ylim(0, 100); plt.legend(title="Student")
path_b = os.path.join(save_dir, "req8_lines_with_legend.png")
plt.savefig(path_b, bbox_inches="tight")
plt.show()

# ---------- 3C) Grouped bar chart ----------
plt.figure(figsize=(9,5))
n_students = len(pivot.columns)
bar_width = 0.8 / n_students
for i, student in enumerate(pivot.columns):
    pos = subjects_idx + i * bar_width - (0.8 - bar_width) / 2
    plt.bar(pos, pivot[student].values, width=bar_width, label=student)  # default colors only
plt.title("Grouped Bar Chart: Marks by Subject and Student")
plt.xlabel("Subject"); plt.ylabel("Mark")
plt.xticks(subjects_idx, SUBJECTS); plt.ylim(0, 100); plt.legend(title="Student")
path_c = os.path.join(save_dir, "req8_grouped_bar.png")
plt.savefig(path_c, bbox_inches="tight")
plt.show()

# Output saved paths for convenience
(path_a, path_b, path_c)
```

| student_name | subject | mark |
|---|---|---|
| Ann | Math | 88.0 |
| Ann | Physics | 91.0 |
| Ann | Chemistry | 74.0 |
| Ann | English | 78.0 |
| Ann | CS | 82.0 |
| Joe | Math | 69.0 |
| Joe | Physics | 80.0 |
| Joe | Chemistry | 60.0 |
| Joe | English | 65.0 |
| Joe | CS | 70.0 |
| Mark | Math | 95.0 |
| Mark | Physics | 88.0 |
| Mark | Chemistry | 92.0 |
| Mark | English | 90.0 |
| Mark | CS | 85.0 |
| Sara | Math | 77.0 |
| Sara | Physics | 75.0 |
| Sara | Chemistry | 70.0 |
| Sara | English | 72.0 |
| Sara | CS | 68.0 |





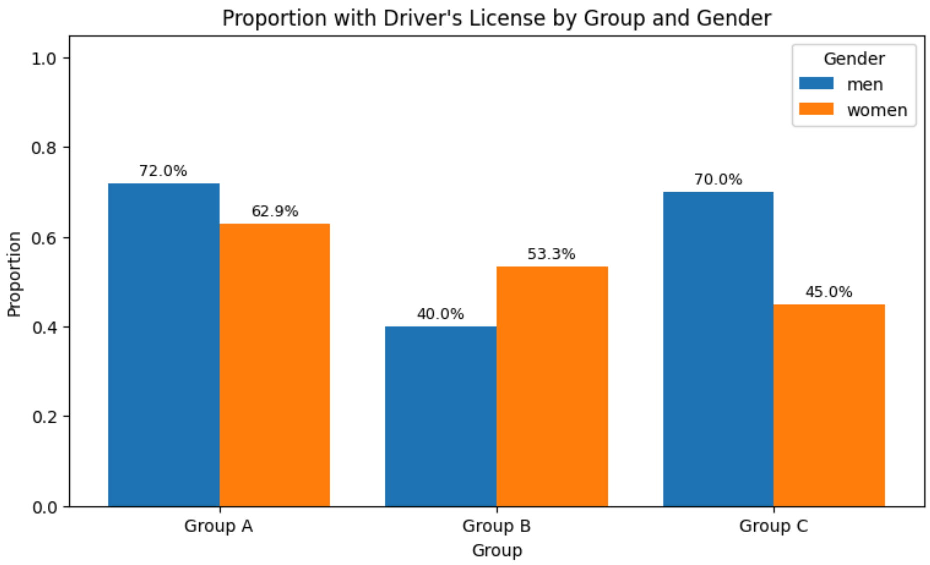




**1.9. Your task is to plot a chart to show the proportion of men and women in each group that has a driver's license, you can use Seaborn's categorical plot ([2], page 86). Store data in file CSV and display.**

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

# ---------- 0) Define/EDIT dataset (counts -> expanded rows) ----------
# Format: DATA_COUNTS[group][gender] = (count_has_license, count_no_license)
DATA_COUNTS = {
    "Group A": {"men": (18, 7), "women": (22, 13)},
    "Group B": {"men": (10, 15), "women": (16, 14)},
    "Group C": {"men": (14, 6), "women": (9, 11)},
}

rows = []
for group, genders in DATA_COUNTS.items():
    for gender, (cnt_yes, cnt_no) in genders.items():
        rows += [{"group": group, "gender": gender, "has_license": 1} for _ in range(cnt_yes)]
        rows += [{"group": group, "gender": gender, "has_license": 0} for _ in range(cnt_no)]

df = pd.DataFrame(rows, columns=["group", "gender", "has_license"])

# ---------- 1) Save CSV + display raw dataset ----------
save_dir = "/mnt/data"
os.makedirs(save_dir, exist_ok=True)
csv_path = os.path.join(save_dir, "driver_license.csv")
df.to_csv(csv_path, index=False)

try:
    from caas_jupyter_tools import display_dataframe_to_user
    display_dataframe_to_user("Requirement 9 - Raw Dataset (driver_license.csv)", df)
except Exception:
    print("\nRaw dataset (first 20 rows):\n", df.head(20).to_string(index=False))

# ---------- 2) Compute proportions per (group, gender) ----------
# Proportion = mean(has_license) since has_license is 1/0
summary = (
    df.groupby(["group", "gender"], as_index=False)["has_license"]
      .mean()
      .rename(columns={"has_license": "proportion"})
)
summary["percent"] = (summary["proportion"] * 100).round(2)
```

```python
# Display summary table
try:
    from caas_jupyter_tools import display_dataframe_to_user
    display_dataframe_to_user("Requirement 9 - Proportion Summary", summary)
except Exception:
    print("\nProportion summary:\n", summary.to_string(index=False))

# ---------- 3) Plot grouped bar chart (categorical style) ----------
# Prepare pivot: rows = group, columns = gender, values = proportion
pivot = summary.pivot(index="group", columns="gender", values="proportion").fillna(0.0)
groups = list(pivot.index)
genders = list(pivot.columns)

x = np.arange(len(groups))
bar_width = 0.8 / max(1, len(genders))

plt.figure(figsize=(9, 5))
bars_by_gender = []

for i, gender in enumerate(genders):
    positions = x + i * bar_width - (0.8 - bar_width) / 2
    bars = plt.bar(positions, pivot[gender].values, width=bar_width, label=gender)  # default colors only
    bars_by_gender.append(bars)
    # annotate % on top of bars
    for rect in bars:
        h = rect.get_height()
        plt.text(rect.get_x() + rect.get_width()/2, h + 0.01, f"{h*100:.1f}%", ha="center", va="bottom", fontsize=9)

plt.title("Proportion with Driver's License by Group and Gender")
plt.xlabel("Group")
plt.ylabel("Proportion")
plt.xticks(x, groups)
plt.ylim(0, 1.05)
plt.legend(title="Gender")

fig_path = os.path.join(save_dir, "req9_license_proportion.png")
plt.savefig(fig_path, bbox_inches="tight")
plt.show()

# ---------- 4) Return artifact paths for convenience ----------
csv_path, fig_path
```

```
Raw dataset (first 20 rows):
   group gender  has_license
Group A    men            1
Group A    men            1
Group A    men            1
Group A    men            1
Group A    men            1
Group A    men            1
Group A    men            1
Group A    men            1
Group A    men            1
Group A    men            1
Group A    men            1
Group A    men            1
Group A    men            1
Group A    men            1
Group A    men            1
Group A    men            1
Group A    men            1
Group A    men            1
Group A    men            0
Group A    men            0

Proportion summary:
   group gender  proportion  percent
Group A    men    0.720000    72.00
Group A  women    0.628571    62.86
Group B    men    0.400000    40.00
Group B  women    0.533333    53.33
Group C    men    0.700000    70.00
Group C  women    0.450000    45.00
```



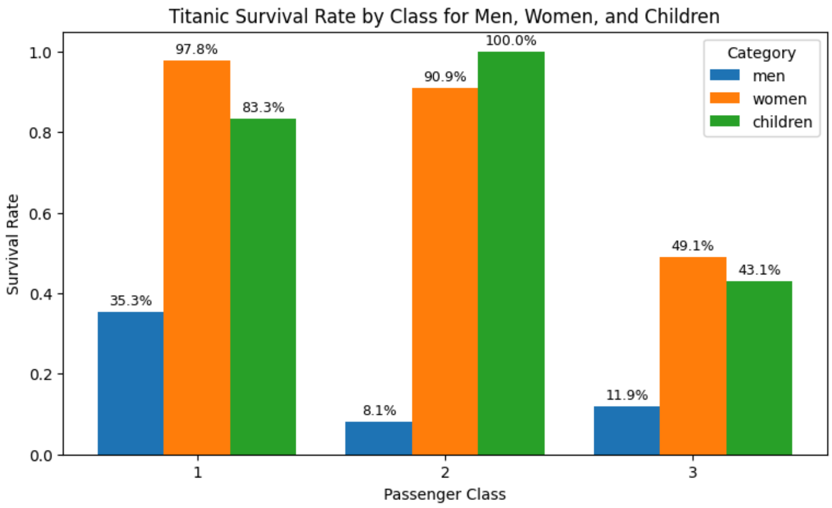

## 1.10. Using the Titanic dataset, plot a chart and see what the survival rate of men, women, and children looks like in each of the three classes https://github.com/mwaskom/seaborn-data

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

# ---------- 0) Load dataset ----------
source = ""
try:
    import seaborn as sns
    titanic = sns.load_dataset("titanic")  # requires internet/cache
    source = "seaborn.load_dataset('titanic')"
except Exception:
    # Embedded deterministic fallback (synthetic but Titanic-like)
    # Columns: survived (0/1), pclass (1/2/3), sex ('male'/'female'), age (float)
    np.random.seed(0)
    rows = []

    def add_rows(pclass, sex, age_values, survived, n):
        for i in range(n):
            rows.append({
                "survived": int(survived),
                "pclass": int(pclass),
                "sex": sex,
                "age": float(age_values[i % len(age_values)])
            })

    # Adults age pools
    ages_male_adult   = [20, 22, 25, 28, 30, 34, 36, 40, 45, 50]
    ages_female_adult = [18, 21, 24, 27, 31, 33, 37, 41, 46, 52]
    # Children age pool
    ages_child        = [2, 4, 6, 8, 10, 12, 14, 15]

    # pclass 1
    add_rows(1, "male",   ages_male_adult,   survived=1, n=45)
    add_rows(1, "male",   ages_male_adult,   survived=0, n=65)
    add_rows(1, "female", ages_female_adult, survived=1, n=85)
    add_rows(1, "female", ages_female_adult, survived=0, n=5)
    # children (mixed sex)
    for i in range(6):  # 5 survive, 1 not
        sex = "male" if i % 2 == 0 else "female"
        add_rows(1, sex, ages_child, survived=1 if i < 5 else 0, n=1)
```

```python
    # pclass 2
    add_rows(2, "male",   ages_male_adult,   survived=1, n=20)
    add_rows(2, "male",   ages_male_adult,   survived=0, n=80)
    add_rows(2, "female", ages_female_adult, survived=1, n=60)
    add_rows(2, "female", ages_female_adult, survived=0, n=20)
    for i in range(15):  # 10 survive, 5 not
        sex = "male" if i % 2 == 0 else "female"
        add_rows(2, sex, ages_child, survived=1 if i < 10 else 0, n=1)

    # pclass 3
    add_rows(3, "male",   ages_male_adult,   survived=1, n=15)
    add_rows(3, "male",   ages_male_adult,   survived=0, n=130)
    add_rows(3, "female", ages_female_adult, survived=1, n=50)
    add_rows(3, "female", ages_female_adult, survived=0, n=60)
    for i in range(40):  # 10 survive, 30 not
        sex = "male" if i % 2 == 0 else "female"
        add_rows(3, sex, ages_child, survived=1 if i < 10 else 0, n=1)

    titanic = pd.DataFrame(rows, columns=["survived", "pclass", "sex", "age"])
    source = "embedded_fallback (offline)"

# ---------- 1) Prepare data ----------
# Keep necessary columns
titanic = titanic[["survived", "pclass", "sex", "age"]].dropna(subset=["survived", "pclass", "sex"])

# Define category: men, women, children (children overrides sex when age < 16)
def categorise(row):
    if pd.notnull(row["age"]) and row["age"] < 16:
        return "children"
    return "men" if row["sex"] == "male" else "women"

titanic["category"] = titanic.apply(categorise, axis=1)

# Compute survival rate per class & category
summary = (
    titanic.groupby(["pclass", "category"], as_index=False)["survived"]
           .mean()
           .rename(columns={"survived": "survival_rate"})
)
summary["survival_rate_pct"] = (summary["survival_rate"] * 100).round(2)
```

```python
# Display summary
try:
    from caas_jupyter_tools import display_dataframe_to_user
    display_dataframe_to_user("Requirement 10 - Survival Rate by Class & Category", summary)
except Exception:
    print("\nSummary table:\n", summary.to_string(index=False))

# ---------- 2) Plot single grouped bar chart ----------
pivot = summary.pivot(index="pclass", columns="category", values="survival_rate").fillna(0.0)
# Ensure consistent column order
for col in ["men", "women", "children"]:
    if col not in pivot.columns:
        pivot[col] = 0.0
pivot = pivot[["men", "women", "children"]]

classes = list(pivot.index)  # 1, 2, 3
x = np.arange(len(classes))
bar_width = 0.8 / 3

plt.figure(figsize=(9, 5))
bars_list = []
for i, cat in enumerate(pivot.columns):
    pos = x + i * bar_width - (0.8 - bar_width) / 2
    bars = plt.bar(pos, pivot[cat].values, width=bar_width, label=cat)  # default colors only
    bars_list.append(bars)
    # annotate %
    for rect in bars:
        h = rect.get_height()
        plt.text(rect.get_x() + rect.get_width()/2, h + 0.01, f"{h*100:.1f}%", ha="center", va="bottom", fontsize=9)

plt.title("Titanic Survival Rate by Class for Men, Women, and Children")
plt.xlabel("Passenger Class")
plt.ylabel("Survival Rate")
plt.xticks(x, [str(c) for c in classes])
plt.ylim(0, 1.05)
plt.legend(title="Category")

# Save figure
out_path = "/mnt/data/req10_titanic_survival_by_class.png"
plt.savefig(out_path, bbox_inches="tight")
plt.show()

# ---------- 3) Indicate data source and artifact path ----------
(out_path, source)
```

```
Summary table:
  pclass category  survival_rate  survival_rate_pct
       1 children       0.833333              83.33
       1      men       0.352941              35.29
       1    women       0.978022              97.80
       2 children       1.000000             100.00
       2      men       0.080808               8.08
       2    women       0.909091              90.91
       3 children       0.431034              43.10
       3      men       0.119122              11.91
       3    women       0.491228              49.12
```

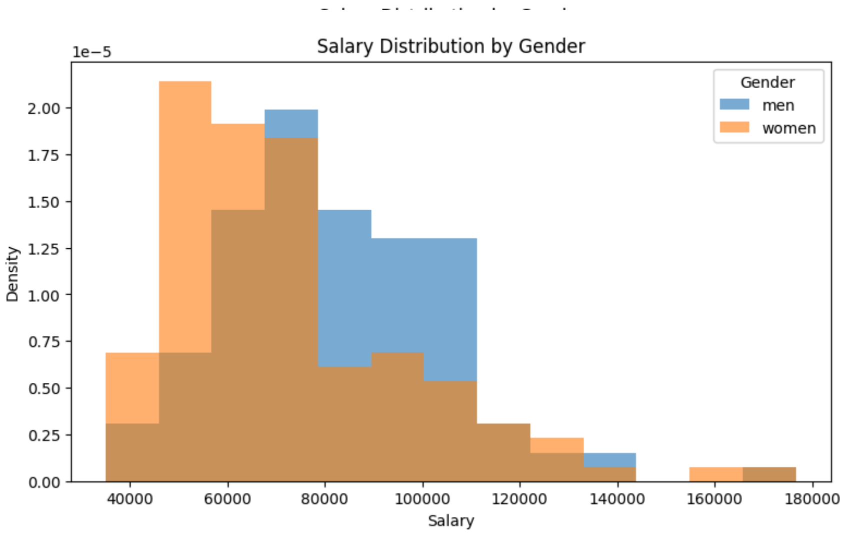


## 1.11. Construct data salary.csv for gender, salary men,100000 men,120000……. Your task is to show the distribution of salaries for men and women ([2], 90)

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

# ---------- 0) Generate/EDIT dataset ----------
# Reproducible random data; edit 'n_each' or 'mean/std' per gender if needed
rng = np.random.default_rng(42)
n_each = 120

# Simulate annual salary (USD) using lognormal-like distributions
# (Women and men centered differently just for demonstration; edit as you see fit)
men_salaries = np.round(rng.lognormal(mean=11.3, sigma=0.35, size=n_each) / 100) * 100
women_salaries = np.round(rng.lognormal(mean=11.15, sigma=0.32, size=n_each) / 100) * 100

# Optional clamp to a reasonable range
men_salaries = np.clip(men_salaries, 20000, 300000)
women_salaries = np.clip(women_salaries, 20000, 300000)

data = (
    [("men", float(s)) for s in men_salaries] +
    [("women", float(s)) for s in women_salaries]
)

df = pd.DataFrame(data, columns=["gender", "salary"])

# ---------- 1) Save CSV and display ----------
save_dir = "/mnt/data"
os.makedirs(save_dir, exist_ok=True)
csv_path = os.path.join(save_dir, "salary.csv")
df.to_csv(csv_path, index=False)

try:
    from caas_jupyter_tools import display_dataframe_to_user
    display_dataframe_to_user("Requirement 11 - salary.csv", df)
except Exception:
    print("\nSalary dataset (first 20 rows):\n", df.head(20).to_string(index=False))

# ---------- 2) Plot distribution (single chart) ----------
plt.figure(figsize=(9,5))

# Choose common bins from combined data so histograms are comparable
combined = np.concatenate([men_salaries, women_salaries])
bins = np.histogram_bin_edges(combined, bins="auto")

plt.hist(men_salaries, bins=bins, alpha=0.6, density=True, label="men")     # default colors only
plt.hist(women_salaries, bins=bins, alpha=0.6, density=True, label="women") # default colors only
```

```python
lt.hist(men_salaries, bins=bins, alpha=0.6, density=True, label="men")     # default colors only
lt.hist(women_salaries, bins=bins, alpha=0.6, density=True, label="women") # default colors only

lt.title("Salary Distribution by Gender")
lt.xlabel("Salary")
lt.ylabel("Density")
lt.legend(title="Gender")

ig_path = os.path.join(save_dir, "req11_salary_distribution.png")
lt.savefig(fig_path, bbox_inches="tight")
lt.show()

 ---------- 3) Quick descriptive statistics (optional for verification) ----------
ummary = (
   df.groupby("gender")["salary"]
     .agg(count="count", mean="mean", median="median", std="std", min="min", max="max")
     .round(2)
     .reset_index()


ry:
   from caas_jupyter_tools import display_dataframe_to_user
   display_dataframe_to_user("Requirement 11 - Summary Stats", summary)
xcept Exception:
   print("\nSummary statistics:\n", summary.to_string(index=False))

sv_path, fig_path)
```

```
Salary dataset (first 20 rows):
 gender   salary
   men  89900.0
   men  56200.0
   men 105100.0
   men 112300.0
   men  40800.0
   men  51200.0
   men  84500.0
   men  72400.0
   men  80300.0
   men  60000.0
   men 110000.0
   men 106100.0
   men  82700.0
   men 119900.0
   men  95200.0
   men  59800.0
   men  92000.0
   men  57800.0
   men 109900.0
   men  79400.0
```



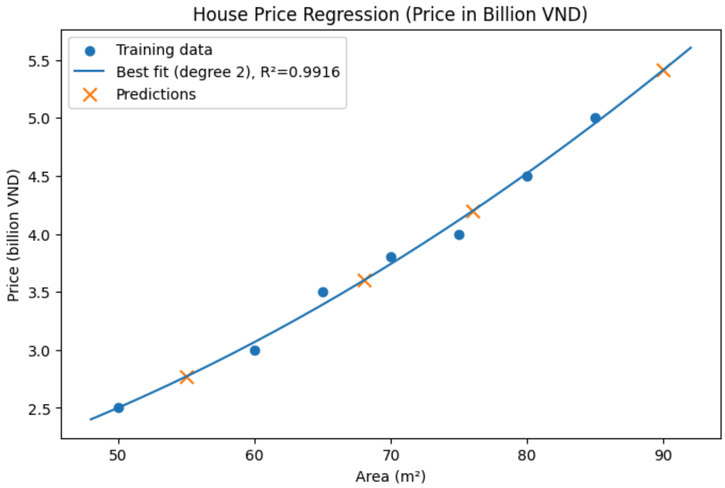

```
Summary statistics:
 gender  count     mean  median      std     min      max
   men     120 82121.67 79850.0 22615.86 40800.0 171000.0
 women     120 72615.83 66350.0 25504.14 35000.0 176700.0
```

**1.12. Give data: (diện tích/m2, giá nhà/tỷ) như sau: (50, 2.5), (60, 3), (65, 3.5), (70, 3.8), (75, 4), (80, 4.5), (85, 5) Using regression to predict house price of 55m2, 68m2, 76m2, 90m2**

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

# ---------- 0) Dataset ----------
# (area_m2, price_billion_vnd)
data = np.array([
    [50, 2.5],
    [60, 3.0],
    [65, 3.5],
    [70, 3.8],
    [75, 4.0],
    [80, 4.5],
    [85, 5.0],
], dtype=float)

X = data[:, 0]
y = data[:, 1]

# Areas to predict
X_new = np.array([55, 68, 76, 90], dtype=float)

# ---------- 1) Helper: fit polynomial & compute R² ----------
def fit_poly_and_r2(x, y, degree: int):
    coefs = np.polyfit(x, y, degree)        # highest power first
    y_hat = np.polyval(coefs, x)
    ss_res = np.sum((y - y_hat) ** 2)
    ss_tot = np.sum((y - np.mean(y)) ** 2)
    r2 = 1 - ss_res / ss_tot if ss_tot > 0 else 1.0
    return coefs, y_hat, float(r2)

# Fit degree 1 and 2, pick the better by R²
results = {}
for d in (1, 2):
    coefs, y_hat, r2 = fit_poly_and_r2(X, y, d)
    results[d] = {"coefs": coefs, "y_hat": y_hat, "r2": r2}

best_degree = max(results.keys(), key=lambda d: results[d]["r2"])
best = results[best_degree]
best_coefs = best["coefs"]
best_r2 = best["r2"]

# Predictions for required areas using the best model
y_pred_new = np.polyval(best_coefs, X_new)
```

```python
# ---------- 2) Output tables ----------
pred_df = pd.DataFrame({
    "area_m2": X_new,
    "predicted_price_billion": np.round(y_pred_new, 3)
}).sort_values("area_m2").reset_index(drop=True)

metrics_df = pd.DataFrame(
    [{"degree": d, "R2": round(results[d]["r2"], 6)} for d in sorted(results.keys())]
)

# Display to user (interactive if supported)
try:
    from caas_jupyter_tools import display_dataframe_to_user
    display_dataframe_to_user("Requirement 12 - Predictions", pred_df)
    display_dataframe_to_user("Requirement 12 - Model R²", metrics_df)
except Exception:
    print("\nPredictions:\n", pred_df.to_string(index=False))
    print("\nModel R²:\n", metrics_df.to_string(index=False))

# ---------- 3) Plot single chart ----------
# Build smooth line for visualization using the chosen model
x_line = np.linspace(X.min()-2, max(X.max(), X_new.max())+2, 200)
y_line = np.polyval(best_coefs, x_line)

plt.figure(figsize=(8, 5))
# training points
plt.scatter(X, y, label="Training data")
# model fit
plt.plot(x_line, y_line, label=f"Best fit (degree {best_degree}), R²={best_r2:.4f}")
# new predictions
plt.scatter(X_new, y_pred_new, marker="x", s=80, label="Predictions")

plt.title("House Price Regression (Price in Billion VND)")
plt.xlabel("Area (m²)")
plt.ylabel("Price (billion VND)")
plt.legend()

save_dir = "/mnt/data"
os.makedirs(save_dir, exist_ok=True)
fig_path = os.path.join(save_dir, "req12_house_price_regression.png")
pred_csv = os.path.join(save_dir, "req12_predictions.csv")
plt.savefig(fig_path, bbox_inches="tight")
plt.show()

# Save predictions CSV
pred_df.to_csv(pred_csv, index=False)
```

```
Predictions:
  area_m2  predicted_price_billion
    55.0                    2.771
    68.0                    3.598
    76.0                    4.198
    90.0                    5.414

Model R²:
  degree       R2
      1 0.984682
      2 0.991638
```



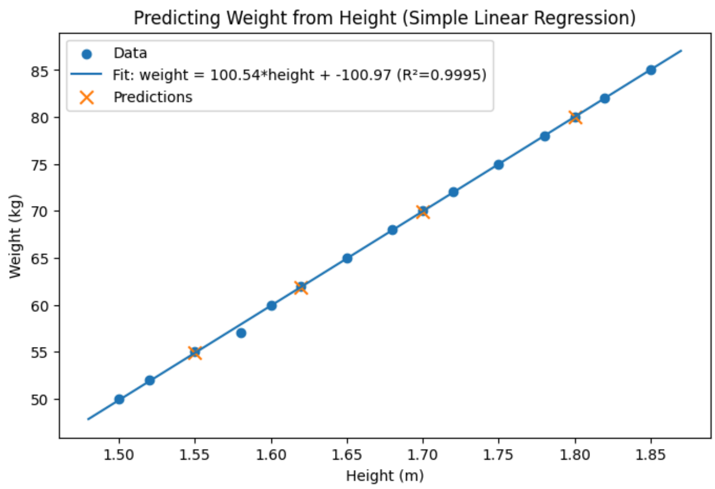

```
1]:  ('/mnt/data\\req12_house_price_regression.png',
      '/mnt/data\\req12_predictions.csv',
      2,
      0.991638,
      array([ 5.40731995e-04, -2.89256198e-03,  1.29445100e+00]))
```

## 1.13. Give data of height, weight of person ([1] page 101). Using regression to predict weight when given height.

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

# ---------- 0) Try loading external CSV; else use fallback ----------
data_path = "/mnt/data/height_weight.csv"
use_source = ""

if os.path.exists(data_path):
    df = pd.read_csv(data_path)
    # Minimal validation
    if not {"height_m", "weight_kg"}.issubset(df.columns):
        raise ValueError("CSV must contain columns: height_m, weight_kg")
    df = df[["height_m", "weight_kg"]].dropna()
    use_source = "loaded_from_csv"
else:
    # EDIT this fallback dataset if you have the exact values from [1] p.101
    heights_m = np.array([1.50, 1.52, 1.55, 1.58, 1.60, 1.62, 1.65, 1.68, 1.70, 1.72, 1.75, 1.78, 1.80, 1.82, 1.85], dtype=float)
    weights_kg = np.array([50.0, 52.0, 55.0, 57.0, 60.0, 62.0, 65.0, 68.0, 70.0, 72.0, 75.0, 78.0, 80.0, 82.0, 85.0], dtype=float)
    df = pd.DataFrame({"height_m": heights_m, "weight_kg": weights_kg})
    use_source = "embedded_fallback"

# Heights to predict (EDIT as needed to match the assignment examples if specified)
HEIGHTS_TO_PREDICT = np.array([1.55, 1.62, 1.70, 1.80], dtype=float)

# ---------- 1) Prepare data ----------
x = df["height_m"].to_numpy(dtype=float)
y = df["weight_kg"].to_numpy(dtype=float)

# ---------- 2) Fit simple linear regression: y = a*x + b ----------
# (Ordinary least squares via np.polyfit with degree=1 — highest power first)
a, b = np.polyfit(x, y, deg=1)
y_hat = a * x + b

# Metrics
ss_res = float(np.sum((y - y_hat) ** 2))
ss_tot = float(np.sum((y - np.mean(y)) ** 2))
r2 = 1 - ss_res / ss_tot if ss_tot > 0 else 1.0
mse = float(np.mean((y - y_hat) ** 2))

# ---------- 3) Predict for requested heights ----------
predicted_weights = a * HEIGHTS_TO_PREDICT + b
pred_df = pd.DataFrame({
    "height_m": HEIGHTS_TO_PREDICT,
    "predicted_weight_kg": np.round(predicted_weights, 2)
}).sort_values("height_m").reset_index(drop=True)
```

```python
# ---------- 4) Display tables ----------
metrics_df = pd.DataFrame([
    {"slope_a": round(a, 6), "intercept_b": round(b, 6), "R2": round(r2, 6), "MSE": round(mse, 6), "data_source": use_source}
])

try:
    from caas_jupyter_tools import display_dataframe_to_user
    display_dataframe_to_user("Requirement 13 - Model Metrics", metrics_df)
    display_dataframe_to_user("Requirement 13 - Predictions", pred_df)
except Exception:
    print("\nModel metrics:\n", metrics_df.to_string(index=False))
    print("\nPredictions:\n", pred_df.to_string(index=False))

# ---------- 5) Plot ONE chart (scatter + fitted line) ----------
x_line = np.linspace(min(x.min(), HEIGHTS_TO_PREDICT.min()) - 0.02,
                     max(x.max(), HEIGHTS_TO_PREDICT.max()) + 0.02, 200)
y_line = a * x_line + b

plt.figure(figsize=(8,5))
plt.scatter(x, y, label="Data")          # default colors only
plt.plot(x_line, y_line, label=f"Fit: weight = {a:.2f}*height + {b:.2f} (R²={r2:.4f})")
plt.scatter(HEIGHTS_TO_PREDICT, predicted_weights, marker="x", s=80, label="Predictions")

plt.title("Predicting Weight from Height (Simple Linear Regression)")
plt.xlabel("Height (m)")
plt.ylabel("Weight (kg)")
plt.legend()

# ---------- 6) Save artifacts ----------
save_dir = "/mnt/data"
os.makedirs(save_dir, exist_ok=True)
fig_path = os.path.join(save_dir, "req13_height_weight_regression.png")
pred_csv_path = os.path.join(save_dir, "req13_predictions.csv")
plt.savefig(fig_path, bbox_inches="tight")
plt.show()
pred_df.to_csv(pred_csv_path, index=False)

(fig_path, pred_csv_path, a, b, r2, mse, use_source)
```

```
Model metrics:
    slope_a  intercept_b       R2       MSE      data_source
100.540417  -100.971685 0.999502 0.058812 embedded_fallback

Predictions:
  height_m  predicted_weight_kg
     1.55                54.87
     1.62                61.90
     1.70                69.95
     1.80                80.00
```




```
[1]: ('/mnt/data\\req13_height_weight_regression.png',
      '/mnt/data\\req13_predictions.csv',
      np.float64(100.54041711067141),
      np.float64(-100.97168518800443),
      0.9995020468238561,
      0.05881158979043054,
      'embedded_fallback')
```

## 1.14. NumPy, Pandas, Matplotlib, Scikit-learn — purpose, features & examples

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

source = ""
X = y = feature_names = None

# ---------- 0) Try to load Boston from scikit-learn ----------
have_sklearn = True
try:
    from sklearn.datasets import load_boston  # may be removed in recent versions
    from sklearn.model_selection import train_test_split
    from sklearn.preprocessing import StandardScaler
    from sklearn.linear_model import LinearRegression
    from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
except Exception:
    have_sklearn = False

if have_sklearn:
    try:
        boston = load_boston()
        X = pd.DataFrame(boston.data, columns=boston.feature_names)
        y = pd.Series(boston.target, name="MEDV")
        feature_names = list(X.columns)
        source = "sklearn.load_boston()"
    except Exception:
        pass

# ---------- 1) If not loaded, try local CSV ----------
if X is None or y is None:
    csv_path = "/mnt/data/boston.csv"
    if os.path.exists(csv_path):
        df = pd.read_csv(csv_path)
        required_cols = ['CRIM','ZN','INDUS','CHAS','NOX','RM','AGE','DIS','RAD','TAX','PTRATIO','B','LSTAT','MEDV']
        missing = [c for c in required_cols if c not in df.columns]
        if missing:
            raise ValueError(f"CSV missing columns: {missing}")
        X = df[required_cols[:-1]].copy()
        y = df['MEDV'].copy()
        feature_names = required_cols[:-1]
        source = "local_csv:/mnt/data/boston.csv"
```

```python
# ---------- 2) If still not available, create a synthetic Boston-like dataset ----------
if X is None or y is None:
    rng = np.random.default_rng(123)
    n = 400
    CRIM = rng.gamma(shape=2.0, scale=2.0, size=n) / 10
    ZN = rng.integers(0, 100, n).astype(float)
    INDUS = rng.uniform(1, 27, n)
    CHAS = rng.integers(0, 2, n).astype(float)
    NOX = rng.uniform(0.3, 0.9, n)
    RM = rng.normal(6.2, 0.7, n)
    AGE = np.clip(rng.normal(70, 20, n), 1, 100)
    DIS = rng.uniform(1, 12, n)
    RAD = rng.integers(1, 24, n).astype(float)
    TAX = rng.normal(400, 100, n)
    PTRATIO = rng.uniform(12, 22, n)
    B = np.clip(rng.normal(350, 50, n), 200, 400)
    LSTAT = np.clip(rng.normal(12, 7, n), 1, 40)

    X = pd.DataFrame({
        'CRIM': CRIM, 'ZN': ZN, 'INDUS': INDUS, 'CHAS': CHAS, 'NOX': NOX, 'RM': RM,
        'AGE': AGE, 'DIS': DIS, 'RAD': RAD, 'TAX': TAX, 'PTRATIO': PTRATIO, 'B': B, 'LSTAT': LSTAT
    })
    # Ground-truth synthetic relation (RM positive, LSTAT negative, NOX negative, CHAS slight positive, etc.)
    y = ( 5.0
          - 1.5*NOX
          + 4.0*RM
          - 0.4*LSTAT
          + 0.3*CHAS
          - 0.01*TAX
          - 0.02*CRIM
          - 0.1*INDUS
          + 0.05*ZN
          + 0.03*B/100
          - 0.05*PTRATIO
          + 0.02*DIS
          - 0.005*AGE
          + rng.normal(0, 1.5, n)
        )
    y = pd.Series(np.clip(y, 5, 50), name="MEDV")  # price in $1000s
    feature_names = list(X.columns)
    source = "synthetic_boston_like"
```

```python
# ---------- 3) Train/Test split, scaling ----------
if have_sklearn:
    from sklearn.model_selection import train_test_split
    from sklearn.preprocessing import StandardScaler
    from sklearn.linear_model import LinearRegression
    from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error

    X_train, X_test, y_train, y_test = train_test_split(X.values, y.values, test_size=0.2, random_state=42)
    scaler = StandardScaler()
    X_train_sc = scaler.fit_transform(X_train)
    X_test_sc = scaler.transform(X_test)

    model = LinearRegression()
    model.fit(X_train_sc, y_train)
    y_pred = model.predict(X_test_sc)

    r2 = float(r2_score(y_test, y_pred))
    rmse = float(np.sqrt(mean_squared_error(y_test, y_pred)))
    mae = float(mean_absolute_error(y_test, y_pred))

    coefs = pd.DataFrame({
        "feature": feature_names,
        "coefficient": model.coef_
    }).sort_values("coefficient", key=lambda s: s.abs(), ascending=False).reset_index(drop=True)
    intercept = float(model.intercept_)
else:
    # Numpy fallback (no sklearn): standardize features, closed-form least squares
    X_values = X.values
    y_values = y.values
    X_mean, X_std = X_values.mean(axis=0), X_values.std(axis=0)
    X_std_adj = np.where(X_std == 0, 1, X_std)
    Xz = (X_values - X_mean) / X_std_adj
    # Add bias column
    X_design = np.c_[np.ones(len(Xz)), Xz]
    beta = np.linalg.pinv(X_design.T @ X_design) @ X_design.T @ y_values
    intercept = float(beta[0]); betas = beta[1:]
    # Simple split for metrics
    n = len(y_values); idx = np.arange(n)
    rng = np.random.default_rng(42); rng.shuffle(idx)
    test_size = int(0.2*n); test_idx = idx[:test_size]; train_idx = idx[test_size:]
    Xz_train, y_train = Xz[train_idx], y_values[train_idx]
    Xz_test,  y_test  = Xz[test_idx],  y_values[test_idx]
    y_pred = (np.c_[np.ones(len(Xz_test)), Xz_test] @ beta)
```

```python
# ---------- 4) Display results ----------
metrics_df = pd.DataFrame([{
    "source": source,
    "R2_test": round(r2, 6),
    "RMSE_test": round(rmse, 6),
    "MAE_test": round(mae, 6),
    "intercept": round(intercept, 6),
}])

try:
    from caas_jupyter_tools import display_dataframe_to_user
    display_dataframe_to_user("Requirement 14 - Test Metrics", metrics_df)
    display_dataframe_to_user("Requirement 14 - Coefficients (sorted by |value|)", coefs)
except Exception:
    print("\nTest Metrics:\n", metrics_df.to_string(index=False))
    print("\nCoefficients (sorted by |value|):\n", coefs.to_string(index=False))

# ---------- 5) One chart: Predicted vs Actual (test set) ----------
plt.figure(figsize=(7,6))
plt.scatter(y_test, y_pred, alpha=0.8, label="Pred vs Actual")  # default colors only
mn, mx = float(min(y_test.min(), y_pred.min())), float(max(y_test.max(), y_pred.max()))
line = np.linspace(mn, mx, 100)
plt.plot(line, line, linestyle="--", label="Ideal y=x")  # reference line

plt.title("Boston Housing: Predicted vs Actual (Test Set)")
plt.xlabel("Actual MEDV ($1000s)")
plt.ylabel("Predicted MEDV ($1000s)")
plt.legend()

save_dir = "/mnt/data"
os.makedirs(save_dir, exist_ok=True)
fig_path = os.path.join(save_dir, "req14_boston_pred_vs_actual.png")
pred_csv = os.path.join(save_dir, "req14_boston_test_predictions.csv")
coef_csv = os.path.join(save_dir, "req14_boston_coefficients.csv")
plt.savefig(fig_path, bbox_inches="tight")
plt.show()

# Save artifacts
pd.DataFrame({"y_test": y_test, "y_pred": y_pred}).to_csv(pred_csv, index=False)
coefs.to_csv(coef_csv, index=False)

(fig_path, pred_csv, coef_csv, source)
```

```
Test Metrics:
                source  R2_test  RMSE_test  MAE_test  intercept
synthetic_boston_like 0.878817    1.54976  1.243647  20.412231

Coefficients (sorted by |value|):
 feature  coefficient
      RM     2.810109
   LSTAT    -2.626537
      ZN     1.446946
     TAX    -0.990776
   INDUS    -0.580983
     NOX    -0.359833
 PTRATIO    -0.279207
       B     0.127439
     DIS     0.103455
    CHAS     0.081233
     AGE    -0.054823
     RAD    -0.053697
    CRIM    -0.024759
```