# Setup (paste once)

import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
from sklearn.linear\_model import LinearRegression  
from sklearn.model\_selection import train\_test\_split, GridSearchCV  
from sklearn.preprocessing import MinMaxScaler  
from sklearn.svm import SVC  
from sklearn.metrics import accuracy\_score, r2\_score, mean\_squared\_error  
np.random.seed(2025) # exam uses this seed in Q21–23

# Q21–Q23 (theory mini‑keys)

* **Q21 (array x):** created by np.random.normal(loc=3, scale=1, size=100) → *1‑D array*, 100 draws, mean≈3, std≈1 (sample mean/std won’t be exactly 3/1).
* **LinearRegression().score(X, y)** returns **R²**.
* **R² (coefficient of determination):** 1 − SSR/SST = proportion of variance in y explained by the model.
* **MSE:** average squared residual. *Lower is better* on both train and test (on test, “lower MSE” means **better** generalization).

# Task 1 — Logistic Regression (manual + code)

**Given** f(x) = w0 + w1 \* income with w0 = -1.5, w1 = 0.15. - **Sigmoid (probability):** p = 1 / (1 + exp(-f(x))). - **Decision rule:** predict 1 if p >= 0.5, else 0.

**Template (income=20):**

w0, w1 = -1.5, 0.15  
x = 20  
log\_odds = w0 + w1 \* x # = 1.5  
p = 1 / (1 + np.exp(-log\_odds)) # 0.8176 -> 0.82  
pred = int(p >= 0.5) # 1  
print(round(p, 2), pred)

**Round to 2 decimals** when asked.

# Task 2 — Data Generation & Visualization

**Goal**: build x1, x2, df, means, and a gold scatter.

x1 = np.arange(0, 10, 0.1) # 0..9.9 (10 not included)  
x2 = np.exp(x1)  
df = pd.DataFrame({'x1': x1, 'x2': x2})  
print(df.shape) # (100, 2)  
print(df['x1'].mean()) # 4.95  
print(round(df['x2'].mean(), 2))  
  
plt.scatter(x1, x2, color='gold')  
plt.xlabel('x1'); plt.ylabel('x2'); plt.title('Scatter Plot of x1 vs x2')  
plt.show()

**Quick checks:** length 100; x1 mean ≈ 4.95; color string exactly 'gold'.

# Task 3 — SVM Programming (Linear SVC, scaling, grid search)

## (A) Load, select 5 features, split 80/20

from sklearn.datasets import load\_breast\_cancer  
X, y = load\_breast\_cancer(return\_X\_y=True, as\_frame=True)  
X = X[['mean radius','mean texture','mean perimeter','mean area','mean smoothness']]  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(  
 X, y, test\_size=0.2, random\_state=2025, stratify=y)  
print(X\_train.shape[0], X\_test.shape[0]) # train count, test count

## (B) Scale with MinMaxScaler (fit on train only)

scaler = MinMaxScaler()  
X\_train\_scaled = scaler.fit\_transform(X\_train)  
X\_test\_scaled = scaler.transform(X\_test)  
print(round(X\_test\_scaled.min(), 2), round(X\_test\_scaled.max(), 2)) # report mins/maxes  
# Why not 0/1 on test? Because scaler uses train min/max only; test can fall outside that range.

## (C) Train two linear SVMs (C=0.1 and 100) and report accuracies

svm01 = SVC(kernel='linear', C=0.1, random\_state=2025)  
svm100 = SVC(kernel='linear', C=100, random\_state=2025)  
svm01.fit(X\_train\_scaled, y\_train); svm100.fit(X\_train\_scaled, y\_train)  
acc\_tr\_01 = round(accuracy\_score(y\_train, svm01.predict(X\_train\_scaled)), 2)  
acc\_te\_01 = round(accuracy\_score(y\_test, svm01.predict(X\_test\_scaled)), 2)  
acc\_tr\_100 = round(accuracy\_score(y\_train, svm100.predict(X\_train\_scaled)), 2)  
acc\_te\_100 = round(accuracy\_score(y\_test, svm100.predict(X\_test\_scaled)), 2)  
print(acc\_tr\_01, acc\_te\_01, acc\_tr\_100, acc\_te\_100)  
# Pick the model with the higher \*\*test\*\* accuracy; explain: larger C = weaker regularization (tighter fit), smaller C = stronger regularization.

## (D) Grid search for best C (5‑fold CV), then refit and test

param\_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 50, 100, 150, 200, 500]}  
svc = SVC(kernel='linear')  
grid = GridSearchCV(svc, param\_grid=param\_grid, cv=5)  
grid.fit(X\_train\_scaled, y\_train)  
print('Best C =', grid.best\_params\_['C'])  
print('CV mean score =', round(grid.best\_score\_, 2))  
# Evaluate best model on test  
y\_pred\_best = grid.best\_estimator\_.predict(X\_test\_scaled)  
print('Test accuracy =', round(accuracy\_score(y\_test, y\_pred\_best), 2))

# Q21–Q23 — tiny reusable code blocks

## Build the exact Q21 objects & fit LinearRegression

np.random.seed(2025)  
x = np.random.normal(loc=3, scale=1, size=100)  
e = np.random.normal(loc=1, scale=1, size=100)  
y = 1 + 2\*x + e  
X\_df = pd.DataFrame({'X': x})  
lin = LinearRegression().fit(X\_df, y)  
print('shape:', X\_df.shape, 'R2 (score):', round(lin.score(X\_df, y), 3))  
print('MSE:', round(mean\_squared\_error(y, lin.predict(X\_df)), 3))

# One‑liners you may need

* **Sigmoid:** lambda z: 1/(1+np.exp(-z))
* **R²:** r2\_score(y\_true, y\_pred) (or model.score(X, y))
* **MSE:** mean\_squared\_error(y\_true, y\_pred)
* **Split:** train\_test\_split(X, y, test\_size=0.2, random\_state=2025, stratify=y)
* **Scaling rule:** fit on **train only**, transform both; test min/max won’t be 0/1.

# Common pitfalls checklist (fast self‑audit)

* ✅ *Used the 5 required features* exactly (spelling matters).
* ✅ random\_state=2025 and test\_size=0.2.
* ✅ **MinMaxScaler:** fit\_transform on train, only transform on test.
* ✅ For Task 1, **probability first**, then threshold at 0.5, **round to 2 decimals**.
* ✅ For plots: color='gold', labeled axes, title present.
* ✅ Report **counts**, **min/max**, and **accuracies** rounded to 2 decimals where asked.

# Tiny index (Ctrl/Cmd‑F these)

* “**Sigmoid**”, “**R²**”, “**MSE**”
* “**MinMaxScaler (fit on train)**”
* “**GridSearchCV**”
* “**SVC linear C**”
* “**x1/x2 scatter gold**”