CMSC 436 — Project 1

Hudson, Isabella, Jacob, and Hayat

Team & Certification

- Team member 1: *Isabella Darko / Normalized and plotted data set B*
- Team member 2: *Hayat Bounite / Normalized and plotted data set A*
- Team member 3: *Hudson Finnochio / Normalized and plotted data set C*
- Team member 4: *Jacob Alexander / Did Part 2, the McCulloch–Pitts Neuron*

Part 1 — Dataset Exploration and Linear Separation

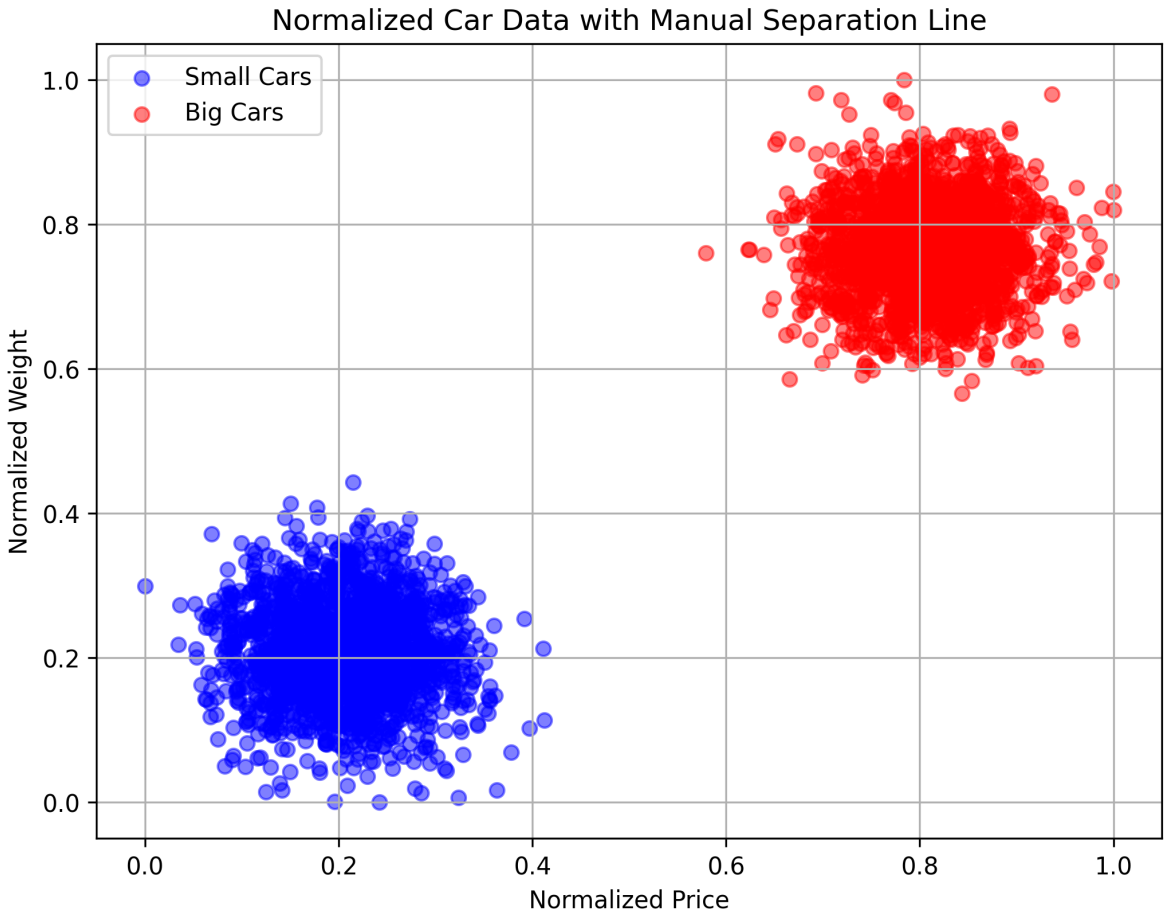
Dataset A

Q1 — Normalize and visualize Dataset A

```
%matplotlib inline

import pandas as pd
import matplotlib.pyplot as plt

# Loading dataset
df = pd.read_csv("data/groupA.txt", sep=",", header=None, names=["Price",
 ↪  "Weight", "Type"])

# Price and Weight normalization
df["Price_norm"] = (df["Price"] - df["Price"].min()) / (df["Price"].max() -
 ↪  df["Price"].min())
df["Weight_norm"] = (df["Weight"] - df["Weight"].min()) / (df["Weight"].max()
 ↪  - df["Weight"].min())

print(df.head())
```

```
# visualization
small_cars = df[df["Type"] == 0]
big_cars = df[df["Type"] == 1]

plt.figure(figsize=(8,6))
plt.scatter(small_cars["Price_norm"], small_cars["Weight_norm"],
 ↪  color='blue', label='Small Cars', alpha=0.5)
plt.scatter(big_cars["Price_norm"], big_cars["Weight_norm"], color='red',
 ↪  label='Big Cars', alpha=0.5)


plt.xlabel("Normalized Price")
plt.ylabel("Normalized Weight")
plt.title("Normalized Car Data with Manual Separation Line")
plt.legend()
plt.grid(True)
plt.show()
```

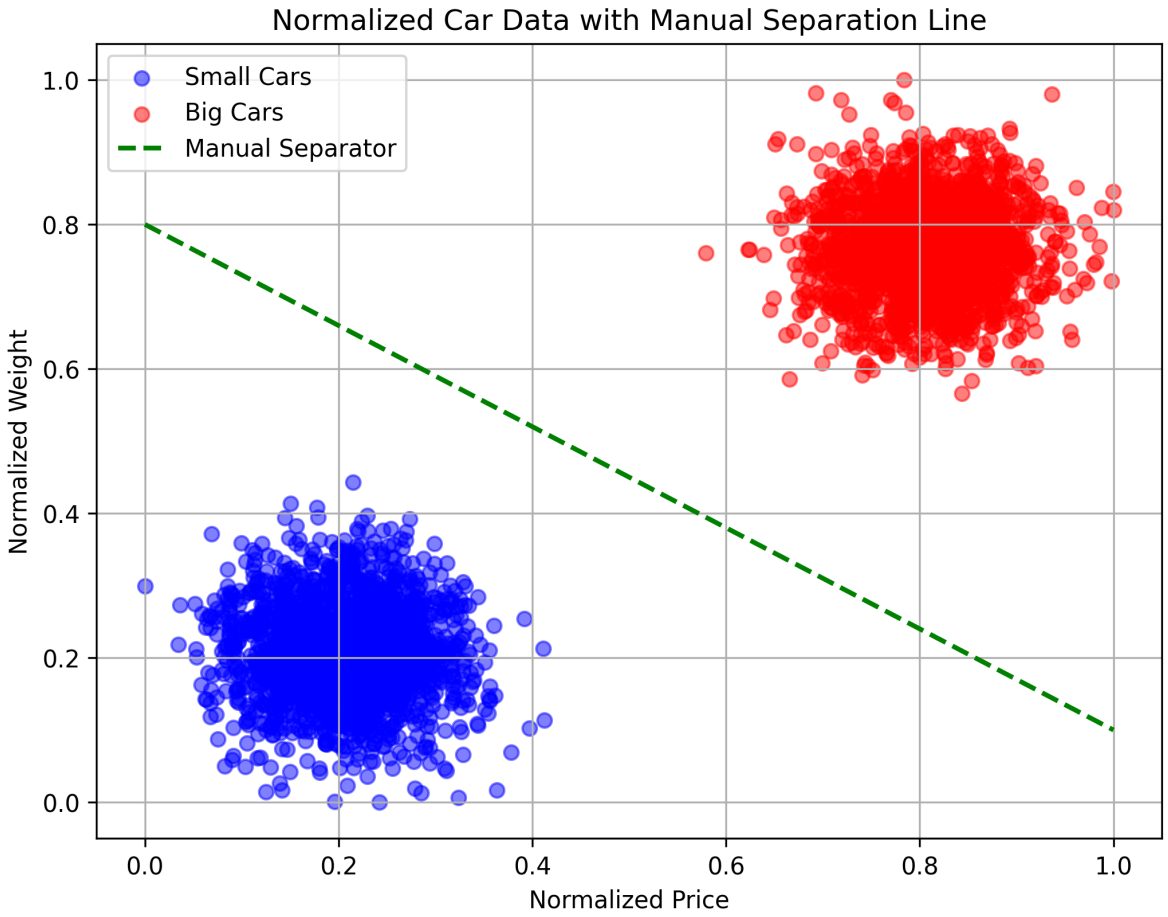
	Price	Weight	Type	Price_norm	Weight_norm
0	23615.75928	61360.78729	1	0.748528	0.727693
1	24310.65894	59815.21582	1	0.852503	0.642088
2	23421.51716	60361.49124	1	0.719464	0.672345
3	24301.90094	61630.95458	1	0.851192	0.742657
4	23907.95918	63305.60879	1	0.792248	0.835412



Q2 — Drawing my line on top of the scatter plot

```
%matplotlib inline

import pandas as pd
import matplotlib.pyplot as plt

# Loading dataset
df = pd.read_csv("data/groupA.txt", sep=",", header=None, names=["Price",
 ↪  "Weight", "Type"])

# Price and Weight normalization
df["Price_norm"] = (df["Price"] - df["Price"].min()) / (df["Price"].max() -
 ↪  df["Price"].min())
```

```
df["Weight_norm"] = (df["Weight"] - df["Weight"].min()) / (df["Weight"].max()
 ↪  - df["Weight"].min())

# visualization
small_cars = df[df["Type"] == 0]
big_cars = df[df["Type"] == 1]

plt.figure(figsize=(8,6))
plt.scatter(small_cars["Price_norm"], small_cars["Weight_norm"],
 ↪  color='blue', label='Small Cars', alpha=0.5)
plt.scatter(big_cars["Price_norm"], big_cars["Weight_norm"], color='red',
 ↪  label='Big Cars', alpha=0.5)

# Manual separation line
x = [0, 1]
y = [0.8, 0.1]
plt.plot(x, y, color='green', linestyle='--', linewidth=2, label='Manual
 ↪  Separator')

plt.xlabel("Normalized Price")
plt.ylabel("Normalized Weight")
plt.title("Normalized Car Data with Manual Separation Line")
plt.legend()
plt.grid(True)
plt.show()
```



Q3 — Turning line into the neuron inequality

We start with the estimated linear function:

$$\text{Weight}_{\text{norm}} = -1.1277 \cdot \text{Price}_{\text{norm}} + 1.0349$$

Rearranging into linear form:

$$7.6233 \cdot \text{Weight}_{\text{norm}} + 8.5968 \cdot \text{Price}_{\text{norm}} - 7.8896 = 0$$

which is equivalent to:

$$f(x) = 8.5968 \cdot \text{Price}_{\text{norm}} + 7.6233 \cdot \text{Weight}_{\text{norm}} - 7.8896$$

The activation is the weighted sum:

$$f(x) \geq 0 \quad \Longrightarrow \quad \text{Predict Type} = 1 \text{ (big car)}$$

Classifier training accuracy:

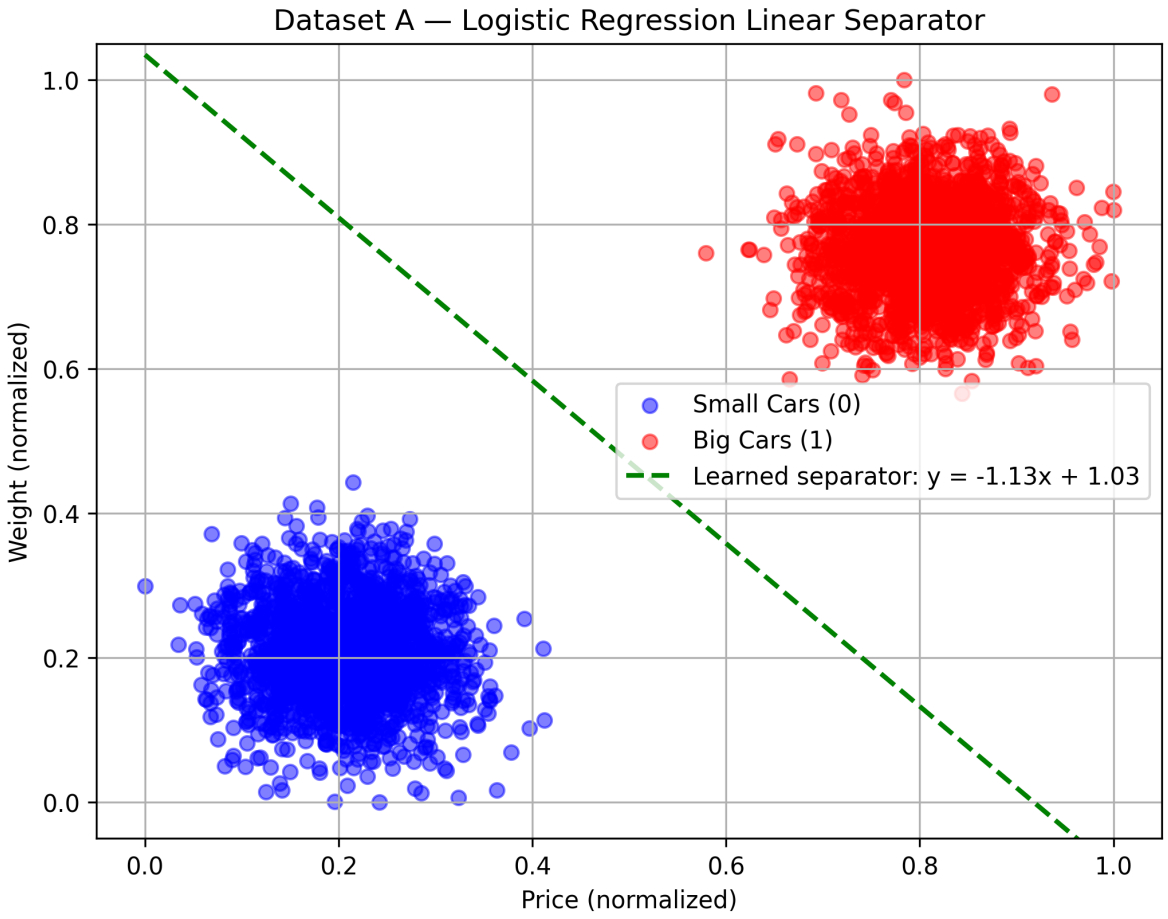
1.0000

Q3a — Logistic regression separator (fit + plot)

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression

# --- Ensure normalized df exists (load if this cell runs standalone) ---
try:
    assert {"Price_norm","Weight_norm","Type"}.issubset(df.columns)
except Exception:
    df = pd.read_csv("data/groupA.txt", sep=",", header=None, names=["Price",
 ↪  "Weight", "Type"])
    df["Price_norm"]  = (df["Price"]  - df["Price"].min())  /
 ↪  (df["Price"].max()  - df["Price"].min())
    df["Weight_norm"] = (df["Weight"] - df["Weight"].min()) /
 ↪  (df["Weight"].max() - df["Weight"].min())

# --- Train logistic regression on normalized features ---
X = df[["Price_norm", "Weight_norm"]].to_numpy()
y = df["Type"].to_numpy()

clf = LogisticRegression(solver="liblinear", random_state=0)
clf.fit(X, y)

# Weights, bias, slope-intercept form
w1, w2 = clf.coef_[0]
b = clf.intercept_[0]

# Avoid divide-by-zero if w2 ~ 0
if abs(w2) < 1e-12:
    slope = np.nan
```

```
    intercept = np.nan
else:
    slope = -w1 / w2
    intercept = -b / w2

acc = clf.score(X, y)

print(f"weights (w1, w2) = ({w1:.6f}, {w2:.6f})")
print(f"bias (b) = {b:.6f}")
print(f"Classifier training accuracy: {acc:.4f}")

# --- Plot data + learned decision boundary (like your classmate's) ---
small_cars = df[df["Type"] == 0]
big_cars   = df[df["Type"] == 1]

xx = np.linspace(0, 1, 200)

plt.figure(figsize=(8,6))
plt.scatter(small_cars["Price_norm"], small_cars["Weight_norm"],
            color="blue", alpha=0.5, label="Small Cars (0)")
plt.scatter(big_cars["Price_norm"], big_cars["Weight_norm"],
            color="red", alpha=0.5, label="Big Cars (1)")

if not np.isnan(slope):
    yy = slope * xx + intercept
    plt.plot(xx, yy, color="green", linestyle="--", linewidth=2,
             label=f"Learned separator: y = {slope:.2f}x + {intercept:.2f}")
else:
    # vertical boundary if w2 ~ 0
    x_const = -b / w1
    plt.axvline(x=x_const, color="green", linestyle="--", linewidth=2,
                label=f"Learned separator: x = {x_const:.2f}")

plt.xlim(-0.05, 1.05)
plt.ylim(-0.05, 1.05)
plt.xlabel("Price (normalized)")
plt.ylabel("Weight (normalized)")
plt.title("Dataset A - Logistic Regression Linear Separator")
plt.legend()
plt.grid(True)
plt.show()
```

```
weights (w1, w2) = (8.596780, 7.623308)
bias (b) = -7.889609
Classifier training accuracy: 1.0000
```



Q4 - Confusion matrix (TP, TN, FP, FN)

```python
import pandas as pd
from sklearn.metrics import confusion_matrix

# predictions
y_pred = clf.predict(X)
y_true = y
```

```
# compute confusion matrix (tn, fp, fn, tp)
tn, fp, fn, tp = confusion_matrix(y_true, y_pred).ravel()

# assign to match your style
TP = int(tp)
TN = int(tn)
FP = int(fp)
FN = int(fn)

print("confusion matrix counts:")
print("TP:", TP, "  FN:", FN)
print("FP:", FP, "  TN:", TN)

# table
pd.DataFrame(
    [[TP, FN],
     [FP, TN]],
    index=["Actual 1 (Big)", "Actual 0 (Small)"],
    columns=["Pred 1 (Big)", "Pred 0 (Small)"]
)
```

```
confusion matrix counts:
TP: 2000   FN: 0
FP: 0   TN: 2000
```

	Pred 1 (Big)	Pred 0 (Small)
Actual 1 (Big)	2000	0
Actual 0 (Small)	0	2000

Q5 — Accuracy and Rates

```
import pandas as pd
from sklearn.metrics import confusion_matrix

# Reuse from Q4 if available; otherwise recompute:
try:
    tn, fp, fn, tp
except NameError:
```

```
    y_pred = clf.predict(X)
    tn, fp, fn, tp = confusion_matrix(y, y_pred).ravel()

N = tp + tn + fp + fn

accuracy   = (tp + tn) / N if N else float("nan")
error_rate = 1 - accuracy if N else float("nan")
TPR = tp / (tp + fn) if (tp + fn) else float("nan")   # recall / sensitivity
 ↪  (class 1: big)
TNR = tn / (tn + fp) if (tn + fp) else float("nan")   # specificity (class 0:
 ↪  small)
FPR = fp / (fp + tn) if (fp + tn) else float("nan")
FNR = fn / (fn + tp) if (fn + tp) else float("nan")

pd.DataFrame([{
    "Accuracy": accuracy,
    "Error": error_rate,
    "TPR": TPR,
    "TNR": TNR,
    "FPR": FPR,
    "FNR": FNR
}]).round(4)
```

	Accuracy	Error	TPR	TNR	FPR	FNR
0	1.0	0.0	1.0	1.0	0.0	0.0

Dataset B

Q1 — Normalize and visualize Dataset B

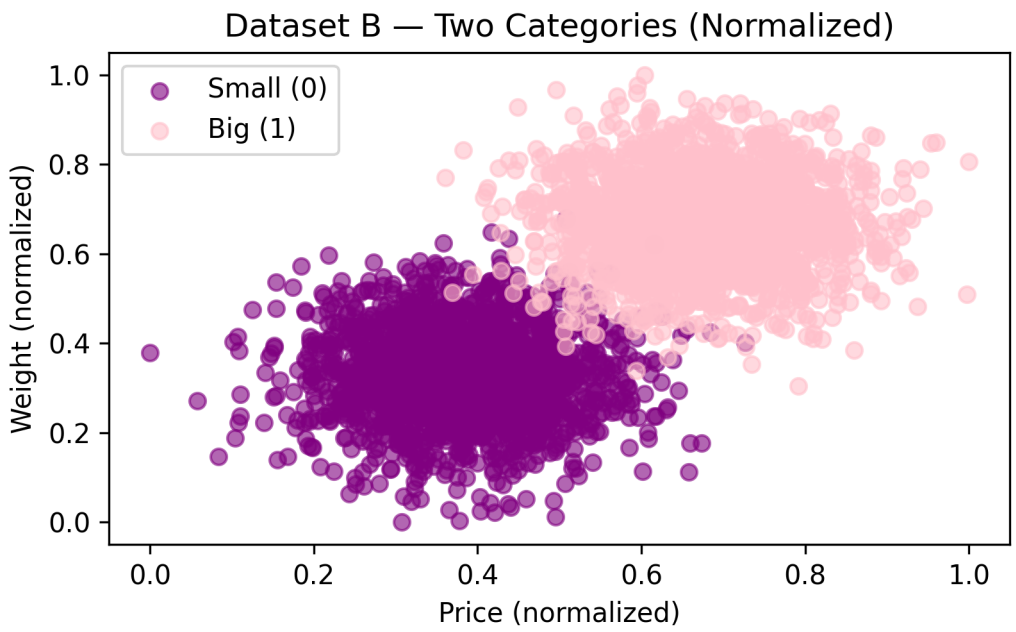
```
import pandas as pd
import matplotlib.pyplot as plt

# 1. Load the dataset into a DataFrame
dfB = pd.read_csv("data/groupB.txt", header=None,
 ↪  names=["price","weight","type"])

# 2. Normalize price and weight (min-max)
dfB["price"] = (dfB["price"] - dfB["price"].min()) / (dfB["price"].max() -
 ↪  dfB["price"].min())
```

```
dfB["weight"] = (dfB["weight"] - dfB["weight"].min()) / (dfB["weight"].max()
 ↪  - dfB["weight"].min())

# 3. Show first 5 rows as an HTML table
dfB.head()
```

	price	weight	type
0	0.579170	0.819156	1
1	0.674541	0.562297	1
2	0.735913	0.646515	1
3	0.895834	0.611533	1
4	0.773647	0.763013	1

```
# 4. Scatter plot of normalized data

# Split the dataset into two groups based on the "type" column
# type = 0 → small cars, type = 1 → big cars
small = dfB[dfB["type"]==0]
big   = dfB[dfB["type"]==1]

# Start a new figure
plt.figure()

# Plot the small cars (purple dots)
# x-axis = normalized price, y-axis = normalized weight
# alpha=0.6 makes the dots slightly transparent so overlaps are visible
plt.scatter(small["price"], small["weight"], alpha=0.6, label="Small (0)",
 ↪  color="purple")

# Plot the big cars (pink dots)
plt.scatter(big["price"],   big["weight"],   alpha=0.6, label="Big (1)",
 ↪  color="pink")

# Add axis labels
plt.xlabel("Price (normalized)")
plt.ylabel("Weight (normalized)")

# Add a plot title
plt.title("Dataset B - Two Categories (Normalized)")
```

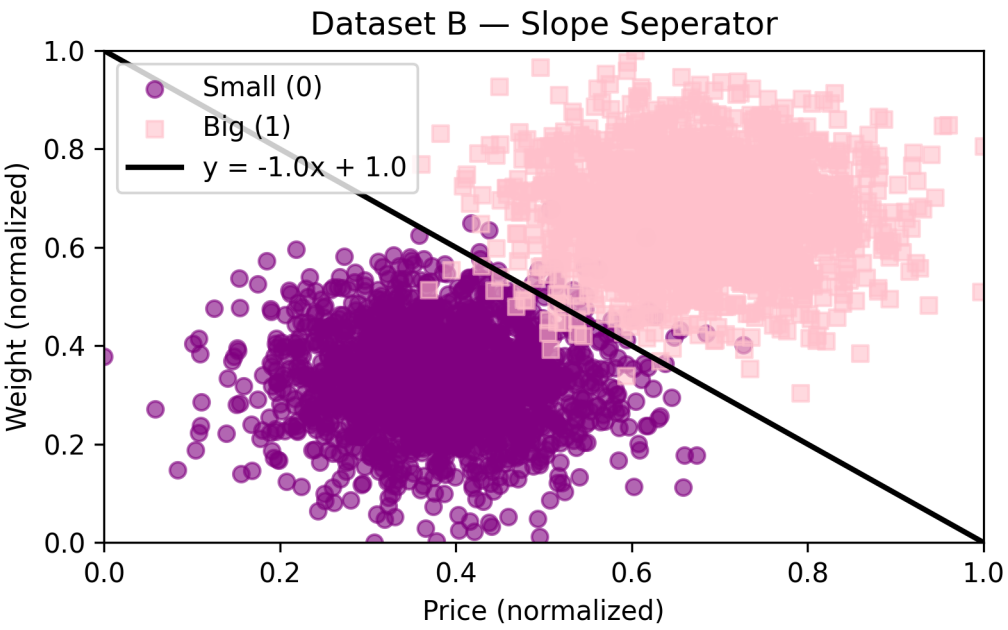
```
# Show the legend so we know which color = which type
plt.legend()

# Adjust the layout so labels/titles don't overlap
plt.tight_layout()

# Save the figure to a PNG file (in the figs/ folder)
plt.savefig("figs/B_scatter.png", dpi=150)

# Display the figure in the Quarto-rendered HTML report
plt.show()
```



Q2 — Drawing my line on top of the scatter plot

```
import numpy as np
import matplotlib.pyplot as plt

m, b = -1.0, 1.0   # slope and intercept for the line

# split the data into small and big again
small = dfB[dfB["type"] == 0]
```

```
big   = dfB[dfB["type"] == 1]

# x values from 0 to 1 since everything is normalized
x_vals = np.linspace(0, 1, 500)
y_vals = m * x_vals + b

plt.figure()

# scatter plot of both groups (purple circles vs pink squares)
plt.scatter(small["price"], small["weight"], alpha=0.6, label="Small (0)",
 ↪  color="purple", marker="o")
plt.scatter(big["price"],   big["weight"],   alpha=0.6, label="Big (1)",
 ↪  color="pink",   marker="s")

# plot the line across the graph
plt.plot(x_vals, y_vals, color="black", linewidth=2, label=f"y = {m}x + {b}")

# keep the axes locked to the normalized range
plt.xlim(0, 1)
plt.ylim(0, 1)

plt.xlabel("Price (normalized)")
plt.ylabel("Weight (normalized)")
plt.title("Dataset B - Slope Seperator")
plt.legend()
plt.tight_layout()
plt.savefig("figs/B_separator.png", dpi=150)
plt.show()
```



Q3 — Turning line into the neuron inequality

We start with the estimated linear function:

$$\text{weight} = -1.0 \cdot \text{price} + 1.0$$

Rearranging into linear form:

$$\text{price} + \text{weight} - 1.0 = 0$$

which is equivalent to:

$$f(x) = 1.0 \cdot \text{price} + 1.0 \cdot \text{weight} - 1.0$$

The activation is the weighted sum:

$$f(x) \geq 0 \quad \Longrightarrow \quad \text{Predict Type} = 1 \text{ (big car)}$$

```
# weights from the line: y = -1*x + 1
w1, w2, theta = 1.0, 1.0, 1.0

net = w1*dfB["price"] + w2*dfB["weight"]
pred = (net >= theta).astype(int)

dfB.assign(pred=pred)[["price","weight","type","pred"]].head()
```

	price	weight	type	pred
0	0.579170	0.819156	1	1
1	0.674541	0.562297	1	1
2	0.735913	0.646515	1	1
3	0.895834	0.611533	1	1
4	0.773647	0.763013	1	1

Q4 - Confusion matrix (TP, TN, FP, FN)

```
# true labels and my predictions from Q3
y_true = dfB["type"].astype(int)
y_pred = pred.astype(int)

TP = int(((y_true == 1) & (y_pred == 1)).sum())
TN = int(((y_true == 0) & (y_pred == 0)).sum())
FP = int(((y_true == 0) & (y_pred == 1)).sum())
FN = int(((y_true == 1) & (y_pred == 0)).sum())

print("confusion matrix counts:")
print("TP:", TP, "  FN:", FN)
print("FP:", FP, "  TN:", TN)

# table
import pandas as pd
pd.DataFrame(
    [[TP, FN],
     [FP, TN]],
    index=["Actual 1 (Big)", "Actual 0 (Small)"],
    columns=["Pred 1 (Big)", "Pred 0 (Small)"]
)
```

```
confusion matrix counts:
TP: 1982   FN: 18
FP: 32   TN: 1968
```

	Pred 1 (Big)	Pred 0 (Small)
Actual 1 (Big)	1982	18
Actual 0 (Small)	32	1968

Q5 Accuracy and Rates

```
# I already have TP, TN, FP, FN from Q4
N = TP + TN + FP + FN

accuracy   = (TP + TN) / N if N else float("nan")
error_rate = 1 - accuracy if N else float("nan")

TPR = TP / (TP + FN) if (TP + FN) else float("nan")   # recall for big (class
 ↪  1)
TNR = TN / (TN + FP) if (TN + FP) else float("nan")   # specificity for small
 ↪  (class 0)
FPR = FP / (FP + TN) if (FP + TN) else float("nan")   # false alarms: small →
 ↪  big
FNR = FN / (FN + TP) if (FN + TP) else float("nan")   # misses: big → small

print("metrics:")
print("accuracy:", round(accuracy, 4))
print("error_rate:", round(error_rate, 4))
print("TPR:", round(TPR, 4))
print("TNR:", round(TNR, 4))
print("FPR:", round(FPR, 4))
print("FNR:", round(FNR, 4))

# table for the report
import pandas as pd
pd.DataFrame([{
    "Accuracy": accuracy,
    "Error": error_rate,
    "TPR": TPR,
    "TNR": TNR,
    "FPR": FPR,
```

```
    "FNR": FNR
}]).round(4)
```

```
metrics:
accuracy: 0.9875
error_rate: 0.0125
TPR: 0.991
TNR: 0.984
FPR: 0.016
FNR: 0.009
```

	Accuracy	Error	TPR	TNR	FPR	FNR
0	0.9875	0.0125	0.991	0.984	0.016	0.009

Dataset C

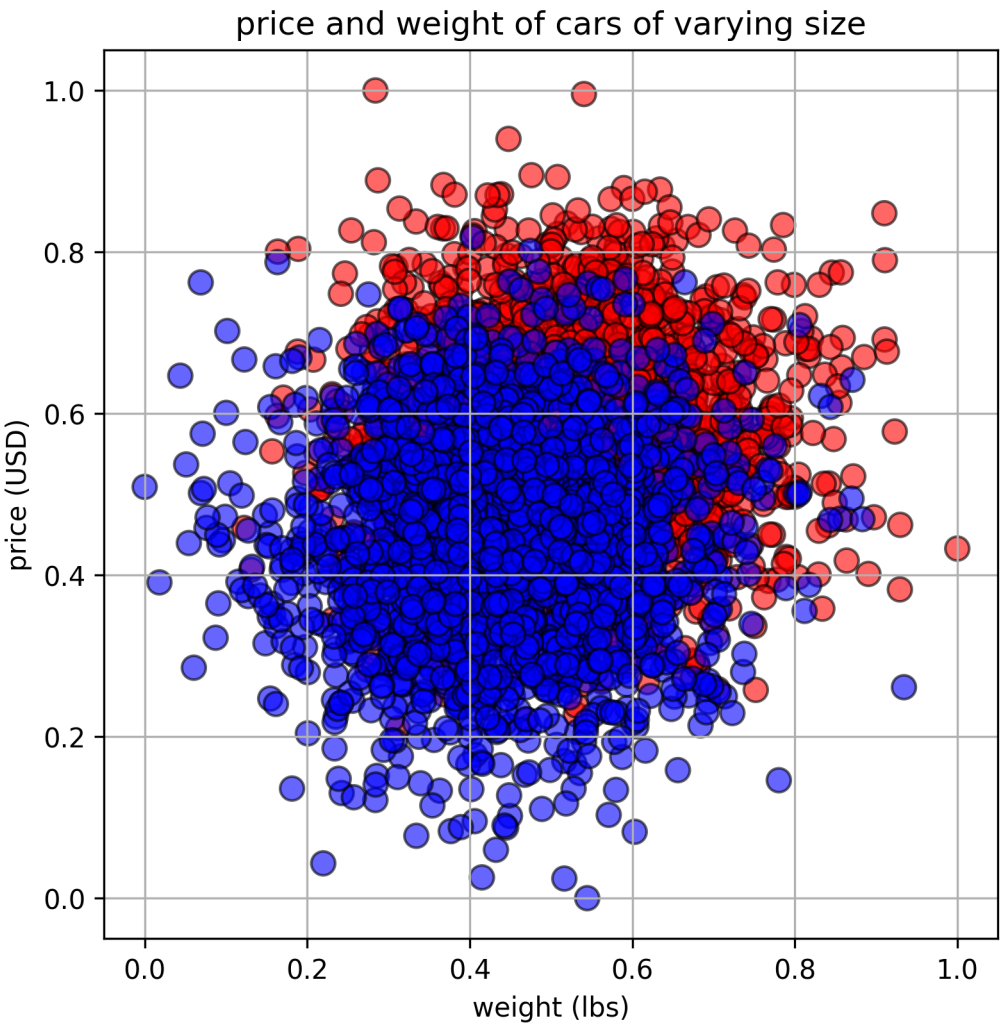
Q1 — Normalize and visualize Dataset C

```
import numpy as np
import matplotlib.patches as mpatches

# input data: price (USD), weight (lbs), and size (small (0) or big (1))
group_c_df = pd.read_csv("data/groupC.txt", header=None,
 ↪  names=["price","weight","size"])

# normalize columns in dataframe
def normalize(df: pd.DataFrame, columns: list):

    df_norm = df.copy()

    for column in columns:
        min_value = df_norm[column].min()
        max_value = df_norm[column].max()
        df_norm[column] = (df_norm[column] - min_value) / (max_value -
 ↪  min_value)

    return df_norm
```

```
processed_df = normalize(group_c_df, columns=['price', 'weight'])

processed_df.head(5)
```

	price	weight	size
0	0.746636	0.736333	1
1	0.775273	0.566274	1
2	0.376447	0.449942	1
3	0.527154	0.529040	1
4	0.759191	0.502520	1

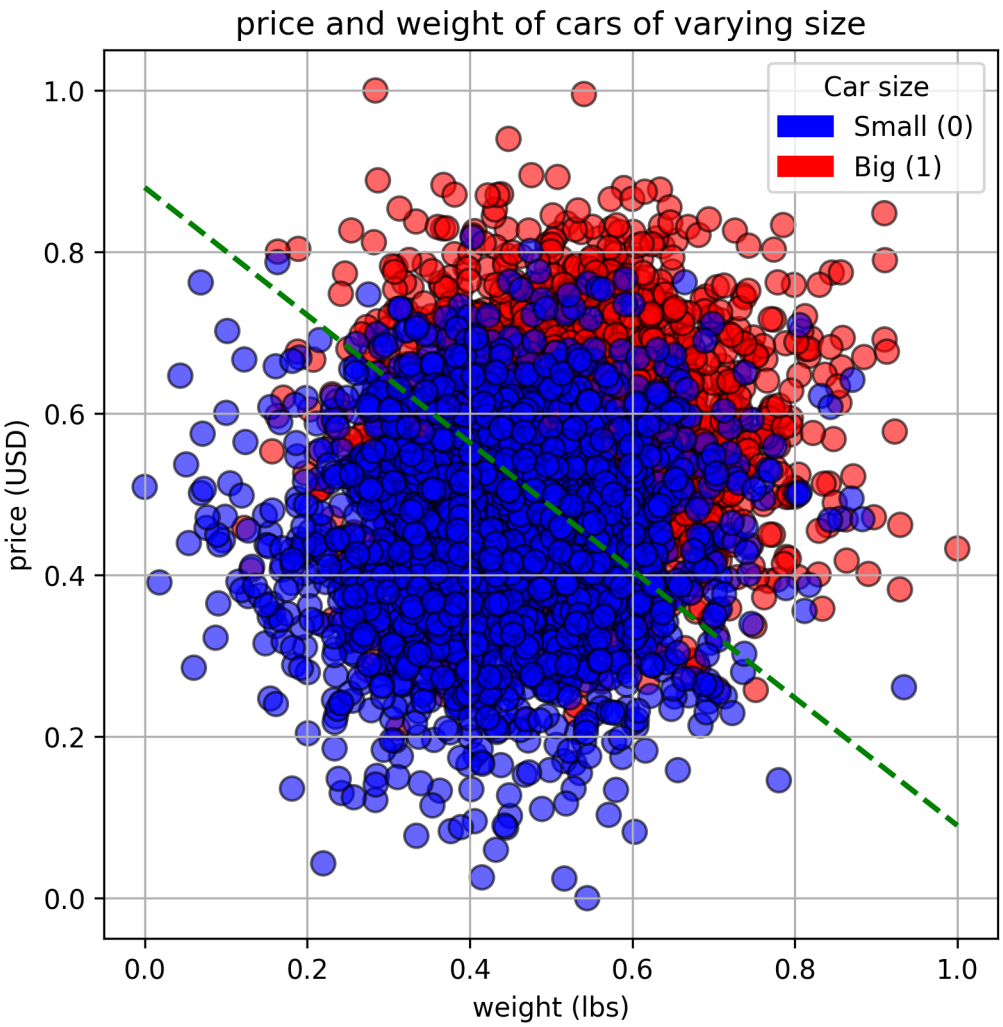
```
# prepare axes and viz for points
x_axis = processed_df['weight'].values
y_axis = processed_df['price'].values
colors = processed_df['size'].values

# plot everything
plt.figure(figsize=(6, 6))
plt.scatter(x_axis, y_axis, c=colors, cmap="bwr", alpha=0.6, s=80,
 ↪  edgecolors="k")
plt.xlabel("weight (lbs)")
plt.ylabel("price (USD)")
plt.title("price and weight of cars of varying size")

plt.grid(True)
plt.show()
```



Q2 — Drawing my line on top of the scatter

```
# estimated function
def f(x):
    return -.79*x + .88

# get values for function

x = np.linspace(0, 1, 30)
y = f(x)
```

```
# prepare axes and viz for points
x_axis = processed_df['weight'].values
y_axis = processed_df['price'].values
colors = processed_df['size'].values

# plot everything again
plt.figure(figsize=(6, 6))
plt.scatter(x_axis, y_axis, c=colors, cmap="bwr", alpha=0.6, s=80,
 ↪  edgecolors="k")
plt.xlabel("weight (lbs)")
plt.ylabel("price (USD)")
plt.title("price and weight of cars of varying size")

plt.plot(x, y, color='green', linestyle='--', linewidth=2, label="y =  -0.79x
 ↪  + 0.88")
small_patch = mpatches.Patch(color='blue', label='Small (0)')
big_patch = mpatches.Patch(color='red', label='Big (1)')

plt.legend(handles=[small_patch, big_patch],
           title="Car size",
           loc="upper right")

plt.grid(True)
plt.show()
```



Q3 — Turning line into the neuron inequality

We start with the estimated linear function:

$$\text{price} = -0.79 \cdot \text{weight} + 0.88$$

Rearranging into linear form:

$$y + 0.79x - 0.88 = 0$$

which is equivalent to:

$$1.0 \cdot \text{price} + 0.79 \cdot \text{weight} - 0.88 = 0$$

The activation is the weighted sum:

$$a(x, y) \leq 0 \quad \Longrightarrow \quad 0.79 \cdot \text{weight} + \text{price} \leq 0.88$$

Q4 - Confusion matrix (TP, TN, FP, FN)

```
from IPython.display import display

pred_labels = (0.79 * processed_df['weight'] + processed_df['price'] >=
 ↪  0.88).astype(int)

# true labels
true_labels = processed_df['size'].astype(int)

# confusion matrix components
TP = ((pred_labels == 1) & (true_labels == 1)).sum()  # true positive -
 ↪  predicted big & actually big
TN = ((pred_labels == 0) & (true_labels == 0)).sum()  # true negative -
 ↪  predicted small & actually small
FP = ((pred_labels == 1) & (true_labels == 0)).sum()  # false positive -
 ↪  predicted big & actually small
FN = ((pred_labels == 0) & (true_labels == 1)).sum()  # false negative -
 ↪  predicted small & actually big

# display confusion matrix
confusion_matrix = pd.DataFrame(
    [[TP, FP],
     [FN, TN]],
    index=['Actual Big (1)', 'Actual Small (0)'],
    columns=['Predicted Big (1)', 'Predicted Small (0)']
)

print("Confusion Matrix:")
display(confusion_matrix)
```

```
Confusion Matrix:
```

	Predicted Big (1)	Predicted Small (0)
Actual Big (1)	1478	579
Actual Small (0)	522	1421

Q5 Accuracy and Rates

```
# metrics / rates
accuracy = (TP + TN) / (TP + TN + FP + FN)
error_rate = 1 - accuracy
true_positive_rate = TP / (TP + FN)
true_negative_rate = TN / (TN + FP)
false_positive_rate = FP / (FP + TN)
false_negative_rate = FN / (FN + TP)

print(f"Accuracy: {accuracy:.3f}")
print(f"Error rate: {error_rate:.3f}")
print(f"True Positive Rate (TPR / Recall for big): {true_positive_rate:.3f}")
print(f"True Negative Rate (TNR / Specificity): {true_negative_rate:.3f}")
print(f"False Positive Rate (FPR): {false_positive_rate:.3f}")
print(f"False Negative Rate (FNR): {false_negative_rate:.3f}")
```

```
Accuracy: 0.725
Error rate: 0.275
True Positive Rate (TPR / Recall for big): 0.739
True Negative Rate (TNR / Specificity): 0.711
False Positive Rate (FPR): 0.289
False Negative Rate (FNR): 0.261
```

Q6 - Similarity of All Data Sets

Comparison of Datasets and Metrics

1. Confusion and Accuracy

Dataset	Separation	Accuracy	TPR (Big)	TNR (Small)	FPR	FNR
A	Most	1	1	1	0	0
B	Moderate	0.987	0.991	0.984	0.016	0.009

Dataset	Separation	Accuracy	TPR (Big)	TNR (Small)	FPR	FNR
C	Least	0.725	0.739	0.711	0.289	0.261

- **Dataset A**: Big and small cars are clearly clustered → All predictions correct.
- **Dataset B**: Classes start to overlap → some misclassifications.
- **Dataset C**: Strong overlap → many false positives and false negatives, lower accuracy.

---

2. Why the datasets differ

- Datasets represent progressively more challenging classification problems:
  - A: clearly separated clusters
  - B: partially overlapping
  - C: heavily overlapping

---

3. Role of Normalization

- Features (price, weight) have different scales.
- Normalizing to [0,1] ensures **both features contribute proportionally** to the linear separator.
- Without normalization, the larger-scale feature would dominate the decision, reducing classifier performance.

Part 2 — McCulloch–Pitts Neuron

Neuron Output Rule

$$\text{Output} = \begin{cases} 1, & \text{if net} \geq 0 \\ 0, & \text{if net} < 0 \end{cases}$$

Truth Table

X	Y	Z	Net = (-3X + 3Y + Z)	Inequality	Output
0	0	0	(-3(0) + 3(0) + 0 = 0)	(0 -1)	1
0	0	1	(-3(0) + 3(0) + 1 = 1)	(1 -1)	1
0	1	0	(-3(0) + 3(1) + 0 = 3)	(3 -1)	1
0	1	1	(-3(0) + 3(1) + 1 = 4)	(4 -1)	1
1	0	0	(-3(1) + 3(0) + 0 = -3)	(-3 < -1)	0
1	0	1	(-3(1) + 3(0) + 1 = -2)	(-2 < -1)	0
1	1	0	(-3(1) + 3(1) + 0 = 0)	(0 -1)	1
1	1	1	(-3(1) + 3(1) + 1 = 1)	(1 -1)	1

Logic Function

$$y = X + Y \quad \text{(equivalently } y = \neg X \vee Y\text{)}$$

Justification: The neuron fires ((y=1)) whenever (X=0) or (Y=1), regardless of the value of (Z).

Activation Range

Range of net values: $(-2,\ 0]$