UNIVERSITY OF AMSTERDAM

Assignment #1

The Weight of Efficiency

MPG–Weight trade-off with Horsepower, Cylinders, Origin, Year, and Model encoded

Friday 31st October, 2025   15:30

Student:
Maarten Stork - 15761770
Lecturer:
Dr. Rob Belleman

Course:
Scientific Visualization and Virtual Reality
Course code:
5284SVVR6Y

Abstract

This report presents a single visualization that explains fuel efficiency (MPG) across a `cars` dataset while encoding seven different attributes. Following principles from Bertin and Mackinlay, MPG vs. weight is mapped to position×position (highest accuracy), horsepower to point size, origin to shape, cylinders to value, and year to a lightness ramp on the marker edge; model names are kept out to avoid clutter, but are present upon hover. The plot shows a strong inverse MPG-weight relationship, modulated by horsepower, with European/Japanese cars clustering at lower weight and higher MPG, and a temporal drift (1970–1982) toward better efficiency. This design maximizes readability by reserving spatial position for the main message and using separable, type-appropriate channels for the rest, ordered by level of importance.

1 Introduction

Fuel efficiency is shaped by a web of interacting factors: amongst which a car's mass and engine power, the engineering choices behind cylinder count, the era in which it was built, and regional design philosophies. Understanding how such variables jointly affect miles per gallon (MPG) informs both history and today's efficiency trade-offs - a key issue given cars' role in energy waste and greenhouse pollution [1].

This report uses a single, information-rich visualization to study all cars in a dataset (with variables *model*, *MPG*, *cylinders*, *horsepower*, *weight*, *year*, and *origin*) as supplied by the University of Amsterdam[1]. The narrative is focused on fuel efficiency versus vehicle weight as mass is the main driver of energy use in conventional cars (heavier cars need more energy to move, directly lowering MPG), a relationship the U.S. EPA directly identifies as a primary determinant of fuel consumption and tailpipe CO2 [2]. Around that core, we also examine the remaining variables in the dataset to see how they interact with weight and fuel efficiency, and whether any secondary patterns emerge that would not

[1]Dataset is available here: `https://github.com/MaartenStork/SVVR/blob/main/Assignment1/cars.csv`

be obvious from the raw data alone. This method of visualization follows Bertin's semiology (visual variables and ordering) [3] and Mackinlay's expressiveness/effectiveness principles [5].

Our setup gives us the following (natural) research question:

What is the relationship between vehicle weight and fuel efficiency (MPG), and how is this relationship modulated by horsepower and cylinder count, varying across region, model year ('70-'82), and individual model?

The results will hopefully sharpen insight into where efficiency is actually won or lost, helping allocate attention (and improvements) where they matter most.

2 Methods

Data and Type Classification We analyze the provided `cars.csv` with seven attributes per car: *model*, *MPG*, *cylinders*, *horsepower*, *weight* (spelled `weigth` in the file), *year*, and *origin*. Column names are standardized (`weigth`→`weight`); numeric fields are coerced to numbers (`errors="coerce"`); and rows missing any of the variables are removed. When *year* appears as two digits (70–82), we define *year_full* = 1900 + *year* for ordered mappings. We refer readers to the Appendix code for the exact implementation details. Additionally, see Table 1 for an overview and explanation of the columns and their types in the `cars` dataset.

Table 1: *Data type classification and justification for the seven columns.*

Column	**Type**	**Justification**
model	Nominal	Identifier; no inherent order.
MPG	Quant. (ratio)	Positive, continuous; differences/ratios meaningful.
cylinders	Ordinal	Integer count; naturally ordered classes.
horsepower	Quant. (ratio)	Positive, continuous; differences/ratios meaningful.
weight	Quant. (ratio)	Mass proxy; differences/ratios meaningful.
year	Ordinal	Differences meaningful; used as ordered index.
origin	Nominal	Region label (US/Europe/Japan).

To align the visualization with perceptual accuracy and the assignment's requirement to encode all variables, we adopt the following importance order: (i) MPG (outcome to explain), (ii) weight (dominant physical driver), (iii) horsepower (performance factor that shifts MPG at fixed weight), (iv) cylinders (engine class proxy), (v) year (temporal drift, 1970–1982), (vi) origin (regional grouping), (vii) model (identifier). This approach lets readers first assess the core quantitative trade-off (MPG vs. weight) using the most accurate channel (position), then layer on separable channels to explain structure around that trend - ending with the least globally informative but lookup-useful detail, the model identifier for point-specific inspection. More details on this are given following the next paragraph.

Design principle and mappings Following Bertin's variables and Mackinlay's effectiveness ordering, we (i) reserve spatial position for the most important quantitative relation and (ii) assign type-appropriate, separable channels to the remaining variables so they add information without competing with the axes.

For the quantitative data, we use channels that support precise magnitude and comparison [5]. Thus, we map *MPG* to the y-position and *weight* to the x-position, so the primary quantitative trade-off is read with the most accurate channel (position×position). Axes use simple linear scales so positions read directly. *Horsepower* is then encoded by area (size): it's the strongest remaining quantitative cue, uses a monotone (linear) stretch between a small minimum (to keep low values visible) and a capped maximum (to limit overlap), and stays secondary to position while remaining separable from other channels. We intentionally avoided a 3D plot (even with three quantitative variables lending itself to position×position×position) because in a static medium we believed it adds some clutter and perspective distortion, while making it more difficult to read and compare.

Then, for the nominal types, we use channels that emphasize clear grouping and comparison across categories [5]. *Origin* is encoded with three easily recognizable shapes that align with common cultural cues: Europe as a square (a simple, neutral geometric block often used for regions/blocks in maps and charts), the USA as a star (referencing the stars on the U.S. flag), and Japan as a circle (reflecting the red sun disc of the Hinomaru). Using this highly discriminable symbol set makes groups preattentively separable without implying any numeric order, keeps the legend compact, and avoids interference with other channels. Secondly, *model* is kept as on-demand tooltip/metadata for point-level lookup; the popup shows only the model name so identification stays separate from the other visuals (that are seen at a glance) and the plot remains uncluttered.

Lastly, for the ordinal datatypes, we use channels that make rank/order obvious without overclaiming precise magnitude [5]. *Year* is normalized across its observed range and encoded as a value for edge lightness (in grayscale, where earlier is darker, later lighter) so temporal order is conveyed subtly. *Cylinders* are encoded with a sequential blue–green value palette, where lighter, desaturated tints represent fewer cylinders and darker, more saturated tones represent more cylinders (3 cyl = light green, . . . , 8 cyl = deep blue-green). A sequential scale conveys order without implying precise magnitude, making it well-suited to ordinal data. We sort the legend in the order of color-values to reinforce rank, and we keep the mapping perceptually monotonic so equal steps in cylinder count produce consistent visual steps. As evident, we chose to order value scales for both *year* and *cylinders*, and we explicitly avoid using color for nominal fields. This follows accessibility literature: sequential palettes with a clear lightness change are easier to read (including for many color-blind readers [7]) and are a good fit for ordered data, whereas mixing multiple unrelated color schemes reduces distinguishability [5, 6].

Referring back to the importance order in the paragraph on **Data and Type Classification**, we made sure each variable's priority aligns with an appropriate visual channel, not just in the context of what kind of data type, but also in the context of the importance laid out in that subsection. For example, for our research question, we judged *horsepower* to be more important than *cylinders*: we center the analysis on MPG versus weight and consider horsepower a more direct design lever than cylinder count. Following Mackinlay's principle that more important features should receive more effective channels [5], we then map *horsepower* to mark size (area), making high-power cars immediately noticable from afar, and map *cylinders* to a blue-green sequential value. The result is a clear size for horsepower, with cylinder count conveyed by a subtler, more gradual, ordered shift in value. This is one example; the same priority-driven mapping, alongside correct channel–data-type pairing, was the base for every channel-assignment in the figure.

3 Results

Figure 1 presents the complete cars dataset in a single scatter plot. Legends for value, shape, and size are shown on the right; a grayscale colorbar below the plot indicates the mapping from year to edge lightness. Axes titles and a light grid for better positional reading.

Figure 2 shows a magnified excerpt of the same plot illustrating the on-hover tooltip, which reveals the corresponding *model* name.

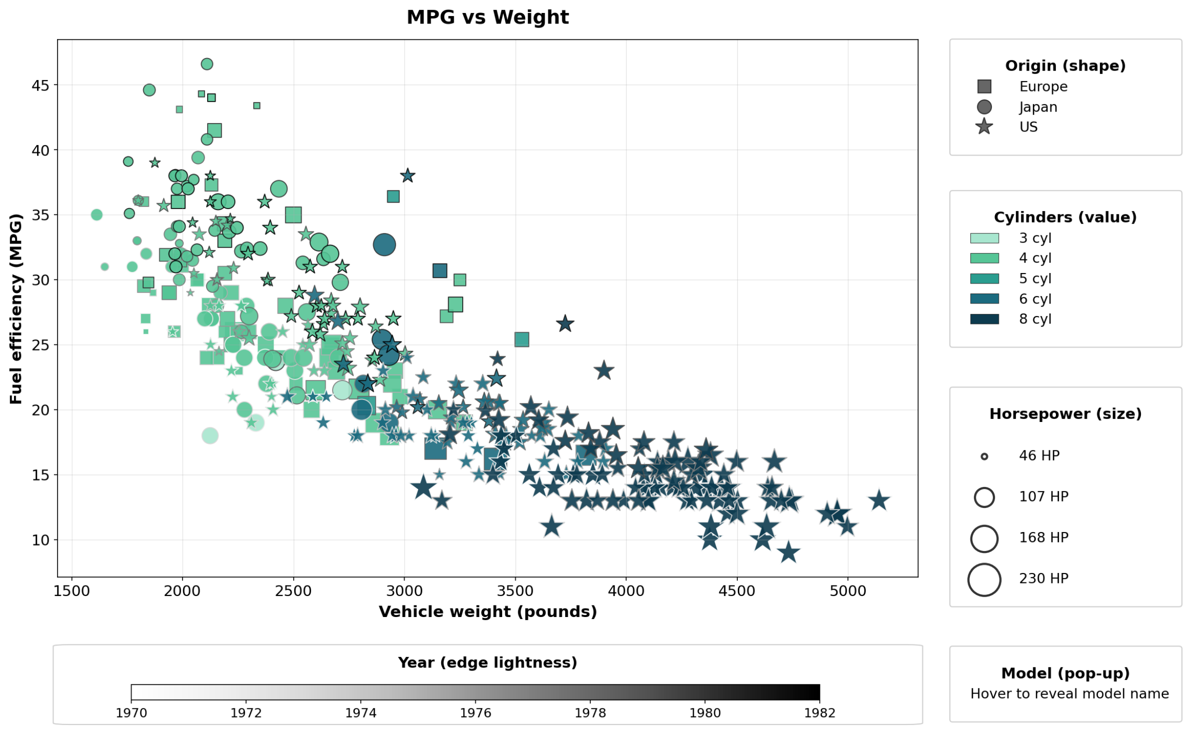

Figure 1: *Single-figure encoding of all seven attributes: weight (x), MPG (y), horsepower (size), origin (shape), cylinders (value), and year (edge lightness); model names are available on hover in the SVG export.*

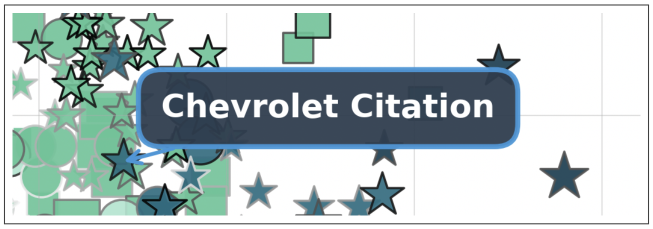

Figure 2: *Detail view showing the on-hover model label in the SVG version of the figure (example: "Chevrolet Citation").*

4 Discussion

The visualization answers the primary question clearly: there is a strong inverse relationship between vehicle weight and fuel efficiency (MPG). Cars cluster along a downward trend (i.e. heavier vehicles achieve fewer miles per gallon) making mass a very dominant determinant of efficiency in this dataset. This core pattern is legible at a glance because it is encoded with position×position, the most accurate channel.

Secondary patterns help explain other notabilities around that trend. Larger markers (more horsepower) tend to sit lower on the MPG axis with higher weights, consistent with what we expect of performance trading off against efficiency. Cylinder count also aligns with this: higher-cylinder engines appear mostly in the heavier, lower-MPG region, while low-cylinder engines are concentrated among lighter, more efficient cars (virtually none among the heaviest vehicles). Regional groupings show many U.S. models in the heavier/less efficient space, while European and Japanese models more often occupy the lighter/higher-MPG zone. The grayscale edge encodes time and indicates a temporal drift from 1970 to 1982 toward lighter, more efficient cars with fewer cylinders and lower horsepower. This is also consistent with known information on the period's regulatory and market pressures [4]. Although model names add little to answering the research question, the option to show them does support identifying specific outliers and inviting comparisons.

References

[1] Jillian Anable and Christian Brand. Energy, pollution and climate change. In *Transport Matters*, pages 55–82. Policy Press, 2019.

[2] Edward Becker. Automotive trends report. *Tribology & Lubrication Technology*, 80(6):96–98, 2024.

[3] Jacques Bertin. *Semiology of Graphics: Diagrams, Networks, Maps*. University of Wisconsin Press, Madison, 1983.

[4] Melvyn A Fuss and Leonard Waverman. Productivity growth in the automobile industry, 1970-1980: A comparisonof canada, japan and the united states. Technical report, National Bureau of Economic Research, 1985.

[5] Jock Mackinlay. Automating the design of graphical presentations of relational information. *Acm Transactions On Graphics (Tog)*, 5(2):110–141, 1986.

[6] Kenneth Moreland. Diverging color maps for scientific visualization. In *International symposium on visual computing*, pages 92–103. Springer, 2009.

[7] Christine Rigden. 'the eye of the beholder'-designing for colour-blind users. *British Telecommunications Engineering*, 17:291–295, 1999.

Appendix

For completeness, I also showcase the core script below. The complete repository (containing the code, `cars` dataset, and several alternative visualizations) is available on GitHub[2].

Listing 1: Data cleaning + visualization script

```python
# cars_viz_vibrant.py

import math
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.lines import Line2D
from matplotlib.patches import Patch
import matplotlib as mpl
import mplcursors

# ---------- Load & clean ----------
df = pd.read_csv("cars.csv")

# Normalize column names (handle misspelling "weigth")
```

[2]See the full codebase here: `https://github.com/MaartenStork/SVVR/main/Assignment1`

```python
cols = {c.lower().strip(): c for c in df.columns}
# rename to consistent lower-case names
rename_map = {}
if "weigth" in cols: # original misspelling
    rename_map[cols["weigth"]] = "weight"
if "mpg" in cols:
    rename_map[cols["mpg"]] = "mpg"
if "cylinders" in cols:
    rename_map[cols["cylinders"]] = "cylinders"
if "horsepower" in cols:
    rename_map[cols["horsepower"]] = "horsepower"
if "year" in cols:
    rename_map[cols["year"]] = "year"
if "model" in cols:
    rename_map[cols["model"]] = "model"
if "origin" in cols:
    rename_map[cols["origin"]] = "origin"

df = df.rename(columns=rename_map)

# Coerce numeric columns; some datasets store '?' for horsepower.
for col in ["mpg", "weight", "horsepower", "year", "cylinders"]:
    if col in df.columns:
        df[col] = pd.to_numeric(df[col], errors="coerce")

# If years are 70..82, convert to full years 1970..1982 for labeling; keep numeric continuity.
if df["year"].dropna().between(0, 150).all(): # heuristic for model-year encoding
    df["year_full"] = np.where(df["year"] < 100, 1900 + df["year"], df["year"])
else:
    df["year_full"] = df["year"]

# Drop rows missing essentials for the mark
df = df.dropna(subset=["mpg", "weight", "horsepower", "cylinders", "origin", "year_full"]).copy()

# ---------- Encodings ----------
# Position -> (weight, mpg)
x = df["weight"].values
y = df["mpg"].values

# Size (area) -> horsepower (use perceptual area scaling)
hp = df["horsepower"].values.astype(float)
hp_min, hp_max = float(np.nanmin(hp)), float(np.nanmax(hp))
# Map horsepower to area in [Amin, Amax]; choose radii so small values are still visible.
Amin, Amax = 25.0, 900.0 # scatter 's' argument is area in points^2
hp_norm = (hp - hp_min) / (hp_max - hp_min + 1e-9)
sizes = Amin + hp_norm * (Amax - Amin)

# Shape -> origin (nominal)
# Different shapes for each origin
origin_markers = {
    'US': '*', # star
    'Europe': 's', # square
    'Japan': 'o' # circle
}

```

```python
def marker_for_origin(o):
    return origin_markers.get(str(o), "o")

origins = df["origin"].astype(str)
origin_levels = list(pd.Categorical(origins).categories)

# Color (facecolor) -> cylinders (ordered - sequential color scale)
# Map cylinders to colors - darker as more cylinders
unique_cyls = sorted(df["cylinders"].dropna().unique())
# Create a blue-green sequential palette (lighter to darker)
from matplotlib.colors import LinearSegmentedColormap
cyl_colors = {
    3: '#a8e6cf', # Lightest green
    4: '#56c596', # Light green
    5: '#2a9d8f', # Medium teal
    6: '#1b6b7f', # Dark teal
    8: '#0d3b4f' # Darkest blue-green
}

def color_for_cyl(c):
    return cyl_colors.get(int(c), '#2a9d8f')

# Year (ordered) -> lightness on the marker EDGE using a grayscale ramp
yr = df["year_full"].values.astype(float)
yr_min, yr_max = float(yr.min()), float(yr.max())
yr_norm = (yr - yr_min) / (yr_max - yr_min + 1e-9)

# Build a grayscale mapper for edges (0 -> dark, 1 -> light)
year_cmap = mpl.colormaps.get_cmap("Greys")
edge_colors = [year_cmap(val) for val in yr_norm]

# ---------- Figure ----------
# Create figure with more space for legends and margins
fig = plt.figure(figsize=(16, 10), dpi=150)

# Set default font size larger
plt.rcParams.update({'font.size': 12})

# Create main plot area with specific position to leave room for legends
ax = plt.axes([0.06, 0.15, 0.70, 0.70]) # [left, bottom, width, height]

# Store all points and their model names for hover functionality
all_scatter_objects = []
model_names = []

# Plot by origin (for shape) and cylinders (for color)
for orig in origin_levels:
    subset = df[origins == orig]
    if subset.empty:
        continue

    # Get the marker for this origin
    marker = marker_for_origin(orig)

    # Now group by cylinders to get colors
    for cyl_val in sorted(subset["cylinders"].dropna().unique()):
        cyl_subset = subset[subset["cylinders"] == cyl_val]
        if cyl_subset.empty:
            continue

        idx = cyl_subset.index

        # Get color for this cylinder count
        fc = color_for_cyl(cyl_val)

        # Get sizes for this subset
        ss = sizes[idx]

        # Get edge colors for year
        ec = [year_cmap(v) for v in ((cyl_subset["year_full"].values.astype(float) - yr_min) / (yr_max -
            yr_min + 1e-9))]

        # Get data
        sub_x = cyl_subset["weight"].values
        sub_y = cyl_subset["mpg"].values
        sub_models = cyl_subset["model"].astype(str).values

        sc = ax.scatter(sub_x, sub_y, s=ss, marker=marker,
                       facecolor=fc, edgecolor=ec, linewidth=0.9, alpha=0.9)
        all_scatter_objects.append(sc)
        # Store model names for this scatter object
        model_names.append(sub_models.tolist())

# Add hover tooltips using mplcursors with beautiful styling
def make_annotation_func(scatter_objs, model_lists):
    """Create annotation function that shows model name on hover"""
    def annotate(sel):
        # Find which scatter object this point belongs to
        point_idx = sel.index
        for scatter_obj, models in zip(scatter_objs, model_lists):
            if sel.artist == scatter_obj:
                if point_idx < len(models):
                    model = models[point_idx]
                    # Normalize model name - capitalize properly
                    model_clean = model.strip().title()
                    # Show only the model name
                    sel.annotation.set_text(model_clean)
                    sel.annotation.set_fontsize(13)
                    sel.annotation.set_fontweight('600') # Semi-bold

                    # Style the box beautifully
                    bbox = sel.annotation.get_bbox_patch()
                    bbox.set_boxstyle("round,pad=0.7")
                    bbox.set_facecolor('#2c3e50') # Dark blue-gray
                    bbox.set_edgecolor('#3498db') # Bright blue border
                    bbox.set_linewidth(2)
                    bbox.set_alpha(0.95)

                    # Style the text
                    sel.annotation.set_color('white')
```

```

                 # Style the arrow
                 sel.annotation.arrow_patch.set_color('#3498db')
                 sel.annotation.arrow_patch.set_linewidth(1.5)
             break
    return annotate

# Create cursor with hover functionality
cursor = mplcursors.cursor(all_scatter_objects, hover=True)
cursor.connect("add", make_annotation_func(all_scatter_objects, model_names))

# Axes labels & title
ax.set_xlabel("Vehicle weight (pounds)", labelpad=6, fontsize=15, fontweight='bold')
ax.set_ylabel("Fuel efficiency (MPG)", labelpad=6, fontsize=15, fontweight='bold')

# Make tick labels bigger
ax.tick_params(axis='both', which='major', labelsize=14)
ax.set_title("MPG vs Weight",
           fontsize=18, pad=15, fontweight='bold')

# Make grid subtle to support reading by position
ax.grid(True, linestyle="-", linewidth=0.5, alpha=0.4)

# ---------- Legends (Unified Professional Layout) ----------

# Create unified legend area on the right
legend_x = 0.78 # X position for all legends
legend_width = 0.20 # Width for legends
legend_height = 0.16 # Height for each legend box
legend_gap = 0.09 # Consistent gap between all legends
legend_gap_large = 0.178 # Larger gap before Horsepower

# Style settings for consistency
legend_style = {
    'frameon': True,
    'framealpha': 1.0,
    'edgecolor': '#d0d0d0',
    'fancybox': True,
    'fontsize': 13,
    'title_fontsize': 14,
    'borderpad': 1.5,
    'columnspacing': 1.0,
    'handletextpad': 1.5
}

# Calculate positions with proper spacing
legend_top = 0.69 # Top position for first legend

# 1. Origin legend - Top (now shapes)
# Adjust marker sizes so they appear visually similar
marker_sizes = {
    'Europe': 12, # square
    'Japan': 13, # circle (slightly bigger)
    'US': 17 # star (needs to be bigger to appear same size)
}
```

```python
origin_handles = [Line2D([0], [0], marker=origin_markers[o], color="none",
                         markerfacecolor="#666666", markeredgecolor="#333333",
                         markersize=marker_sizes.get(o, 12), linewidth=0)
                  for o in origin_levels]
origin_y = legend_top
origin_ax = fig.add_axes([legend_x, origin_y, legend_width, legend_height])
origin_ax.axis('off')
leg1 = origin_ax.legend(origin_handles, origin_levels, title="Origin (shape)",
                         loc='lower left', mode='expand', ncol=1,
                         bbox_to_anchor=(0, 0, 1, 1),
                         **legend_style)
leg1.get_title().set_fontweight('bold')

# 2. Cylinders legend - Middle (now colors)
unique_cyl = sorted(df["cylinders"].dropna().astype(int).unique())
cyl_handles = [Patch(facecolor=cyl_colors[c], edgecolor="#333333", linewidth=0.5)
               for c in unique_cyl]
cylinders_y = origin_y - legend_height - legend_gap
cylinders_ax = fig.add_axes([legend_x, cylinders_y, legend_width, legend_height])
cylinders_ax.axis('off')
leg2 = cylinders_ax.legend(cyl_handles, [f"{c} cyl" for c in unique_cyl],
                           title="Cylinders (value)", loc='lower left', mode='expand', ncol=1,
                           bbox_to_anchor=(0, 0, 1, 1),
                           **legend_style)
leg2.get_title().set_fontweight('bold')

# 3. Horsepower legend - Bottom
hp_ticks = np.linspace(hp_min, hp_max, 4)
hp_sizes = Amin + (hp_ticks - hp_min) / (hp_max - hp_min + 1e-9) * (Amax - Amin)
size_handles = [plt.scatter([], [], s=s, facecolor="white", edgecolor="#333333",
                           linewidth=2.0)
                for s in hp_sizes]
hp_y = cylinders_y - legend_height - legend_gap_large
hp_ax = fig.add_axes([legend_x, hp_y, legend_width, legend_height])
hp_ax.axis('off')
leg3 = hp_ax.legend(size_handles, [f"{int(v)} HP" for v in hp_ticks],
                    title="Horsepower (size)", loc='lower left', mode='expand', ncol=1,
                    bbox_to_anchor=(0, 0, 1, 1),
                    scatterpoints=1, labelspacing=2.0,
                    **legend_style)
leg3.get_title().set_fontweight('bold')

# 4. Model legend (hover info) - Below Horsepower
from matplotlib.patches import Rectangle
model_gap = -0.01 # Smaller gap for Model legend
model_y = hp_y - legend_height - model_gap
model_ax = fig.add_axes([legend_x, model_y, legend_width, legend_height])
model_ax.axis('off')

# Create invisible handles for the text
invisible_handle = Rectangle((0, 0), 0, 0, alpha=0)
legend_style_no_handle = {k: v for k, v in legend_style.items()
                          if k not in ['handletextpad', 'handlelength']}
leg4 = model_ax.legend([invisible_handle],
                       ['Hover to reveal model name'],
                    title="Model (pop-up)",
                    loc='lower left', mode='expand', ncol=1,
                    bbox_to_anchor=(0, 0, 1, 1),
                    handlelength=0,
                    handletextpad=0,
                    **legend_style_no_handle)
leg4.get_title().set_fontweight('bold')

# Year colorbar - with proper box styling
norm = mpl.colors.Normalize(vmin=yr_min, vmax=yr_max)
sm = mpl.cm.ScalarMappable(cmap=year_cmap, norm=norm)
sm.set_array([])

# First create the box background - slightly taller and moved down more
box_ax = fig.add_axes([0.07, -0.04, 0.68, 0.10])
box_ax.set_xlim(0, 1)
box_ax.set_ylim(0, 1)
box_ax.axis('off')

# Draw a rectangle with rounded corners matching the legend style
from matplotlib.patches import FancyBboxPatch
rect = FancyBboxPatch(
    (0, 0), 1, 1,
    boxstyle="round,pad=0.02",
    facecolor='white',
    edgecolor='#d0d0d0',
    linewidth=1,
    transform=box_ax.transAxes,
    clip_on=False,
    zorder=1 # Behind colorbar
)
box_ax.add_patch(rect)

# Add the title
box_ax.text(0.5, 0.87, 'Year (edge lightness)', fontsize=14, fontweight='bold',
          ha='center', va='top', transform=box_ax.transAxes, zorder=3)

# Now create the colorbar ON TOP of the box - adjusted for new box position
cbar_ax = fig.add_axes([0.12, -0.01, 0.56, 0.02])
cbar = plt.colorbar(sm, cax=cbar_ax, orientation='horizontal')
cbar.ax.tick_params(labelsize=12)
cbar.ax.set_zorder(2) # On top of box

# ---------- Nice ticks ----------
# If year_full was 1970..1982, ensure the colorbar shows integers
if yr_max - yr_min <= 20:
    cbar.set_ticks(np.arange(math.floor(yr_min), math.ceil(yr_max)+1, 2))

# No need for tight_layout since we're manually positioning everything

# Save both PNG and SVG with margins to ensure nothing is cut off
plt.savefig("cars_viz_vibrant_final.png", dpi=150, bbox_inches='tight', pad_inches=0.5)
plt.savefig("cars_viz_vibrant_final.svg", bbox_inches='tight', pad_inches=0.5)

print(" Saved: cars_viz_vibrant_final.png")
```

```python
print(" Saved: cars_viz_vibrant_final.svg")

plt.show()
```