Assignment 1 LLM Transcript – Andy Catlin

I started by asking both Claude and ChatGPT about getting more recent and augmented data on Pittsburgh bridges. Then I hand-coded a basic subset/filter using BaseR. At this point, I iterated back and forth with Claude, testing the code as I went. At only one point, did Claude give me broken code, but because I was iterating and testing, it was very easy to fix the broken code. I asked Claude, ChatGPT, and Gemini all to draw a picture of the three rivers and Pittsburgh. Claude produced the best map, which was an Interactive map using leaflet. I don't (yet) understand all of the leaflet code, but it is definitely something that I'll want to build up my skill with. In my conversations, my part is in bold, green, italic.

[1] Conversation with ChatGPT 5.2 on more recent and augmented data around Pittsburgh bridges:

Where can I find a publicly available dataset of Pittsburgh bridges, including year erected and building material?

Thought for 1m 8s

If you specifically want a **public dataset that is explicitly "Pittsburgh bridges" and already includes both "year erected" and "material,"** the most direct option is the classic **UCI "Pittsburgh Bridges" dataset**.

Option 1 (fastest): UCI "Pittsburgh Bridges" dataset (108 bridges)

- **What it contains:** 108 Pittsburgh bridges; includes variables **ERECTED** (year erected / construction era) and **MATERIAL** (bridge material).
- **Where to get it:** UCI Machine Learning Repository ("Pittsburgh Bridges").
- **Good for:** Teaching, quick analysis, lightweight case studies (not a comprehensive asset inventory).

Also commonly mirrored on Kaggle (same underlying dataset).

Option 2 (most "official" and comprehensive): FHWA National Bridge Inventory (NBI)

If you need an authoritative dataset you can filter to **Pittsburgh / Allegheny County**, use the **FHWA National Bridge Inventory**:

- **Where to get it:** FHWA provides annual downloadable NBI files.

- **Fields you care about:**
    - **Item 27 = Year Built** (construction year)
    - **Item 43A = Kind of Material** (material code for main structure type)
- **Good for:** Real inventory work, completeness, joining to GIS, longitudinal analysis.

A note on "City of Pittsburgh bridges" open GIS layers

There *are* City/Regional open data listings for "Pittsburgh Bridges," but at least one of the ArcGIS layers exposed via the catalog appears to be essentially just geometry (e.g., latitude/longitude) and does **not** obviously include year built/material.
(So for your two required attributes, **UCI** or **NBI** are the safer bets.)

Practical recommendation

- If you want a **clean dataset immediately** with exactly those fields: start with **UCI Pittsburgh Bridges**.
- If you want a **real-world, current, defensible dataset**: use **NBI**, filtered to Pittsburgh/Allegheny County, using **Item 27 (Year Built)** + **Item 43A (Material)**.

If you tell me whether your goal is **teaching/demo** vs **real inventory/official analysis**, I can point you to the most efficient "download → filter to Pittsburgh → extract year/material" workflow (Python or R).

[2] Conversation with Claude Opus 4.5 on more recent and augmented data around Pittsburgh bridges:

Where can I find a publicly available dataset of Pittsburgh bridges, including year erected and building material?

AC: Note. I included the same query to both Claude and ChatGPT, because they both produced useful although differing information.

There are several publicly available sources for a Pittsburgh bridges dataset with year erected and building material:

1. UCI Machine Learning Repository (Primary Source)

This is the classic dataset, containing 108 observations with 13 variables including "ERECTED" (year built, ranging from 1818-1986) and "MATERIAL" (WOOD, IRON, STEEL).

UCI Machine Learning Repository It was originally compiled by researchers at Carnegie Mellon University.

You can access it at:

- **https://archive.ics.uci.edu/dataset/18/pittsburgh+bridges**

You can import it directly in Python using:

python

```
from ucimlrepo import fetch_ucirepo

pittsburgh_bridges = fetch_ucirepo(id=18)

X = pittsburgh_bridges.data.features
```

UCI Machine Learning Repository

2. Kaggle

The same UCI dataset is also available on Kaggle Kaggle at:

- **https://www.kaggle.com/datasets/heitornunes/pittsburgh-bridges-data-set**

3. Western Pennsylvania Regional Data Center (WPRDC)

The WPRDC also hosts City of Pittsburgh bridges data Wprdc at:

- **https://data.wprdc.org/dataset/city-of-pittsburgh-bridges**

This is an official government data portal, so it may have more current information compared to the UCI dataset.

The UCI dataset is the most commonly used for machine learning and data analysis projects. It also includes other variables like river, location, purpose, length, lanes, span type, and bridge type.

[3] Conversation with Claude about code

Transcript also here: https://claude.ai/share/a55866c9-78d9-469c-a0ce-f0631a9932bf

Take the code in the last code-block and create a new code block, adding the same functionality for subsetting and filtering using modern tidyverse syntax

8:58 AM

1 step

View the uploaded QMD file to see the code blocks and understand what needs to be converted to tidyverse syntax

Show working file

Identified base R syntax and translated to tidyverse equivalents.

1.
2.

I can see the last code block uses base R syntax for subsetting and filtering. Here's a new code block that accomplishes the same thing using modern tidyverse syntax:

```
## Subset Data and filter out missing data (tidyverse version)

# Using dplyr's filter() and select() functions with pipe operator


df2 <- df |>

  filter(material != "?") |>

  select(identif, river, erected, material)


df2 |>

  gt()
```

Key differences from the base R version:

Base R	**Tidyverse**
df[df$material != "?", ...]	filter(material != "?")
c("identif", "river", "erected", "material")	select(identif, river, erected, material)

The tidyverse approach is more readable because:

- filter() clearly indicates row filtering based on a condition
- select() clearly indicates column selection
- Column names don't need to be quoted inside select()

- The pipe (|>) chains operations in a logical, left-to-right flow

No need to reference the dataframe name (df$) inside the filter condition

Going forward, always prefer tidyverse syntax; use ggplot2 when appropriate . Next, create exploratory data analysis on the river column, showing both a table of values with counts, and a bar graph, showing counts with percentage count value at top of each bar.

9:04 AM

Architected exploratory analysis with tidyverse and ggplot2 visualization.

1.
2.
3.
4.

Here's a new code block for exploratory data analysis on the river column:

```
## EDA: River Column Analysis


# Table of values with counts and percentages
river_summary <- df2 |>
 count(river) |>
 mutate(
  percentage = n / sum(n) * 100,
  percentage_label = paste0(round(percentage, 1), "%")
 )


river_summary |>
 gt() |>
 cols_label(
  river = "River",
    n = "Count",
    percentage = "Percentage",
    percentage_label = "Pct Label"
  ) |>
  fmt_number(columns = percentage, decimals = 1)


# Bar graph with percentage labels at top of each bar
ggplot(river_summary, aes(x = river, y = n, fill = river)) +
  geom_col() +
  geom_text(
    aes(label = percentage_label),
    vjust = -0.5,
    size = 4
  ) +
  labs(
    title = "Distribution of Pittsburgh Bridges by River",
    x = "River",
    y = "Count"
  ) +
  theme_minimal() +
  theme(legend.position = "none") +
  scale_y_continuous(expand = expansion(mult = c(0, 0.1)))
```

What this code does:

1. **Table creation:**
    - count(river) tallies occurrences of each river value
    - mutate() calculates percentages and creates formatted labels

    - gt() formats it as a nice table with proper column headers

2. **Bar graph:**

    - geom_col() creates the bars (uses the pre-computed counts)
    - geom_text() adds percentage labels above each bar
    - scale_y_continuous(expand = ...) adds space above bars so labels aren't cut off
    - theme_minimal() provides a clean look
    - Legend is hidden since the x-axis already shows river names

Now, perform some feature engineer. Create a new column called "era" with values "old" and "new" where bridged erected before 1900 have the value "old" and bridges erected in or after 1900 have the value new. Create a table showing the era in the rows and the material name in the columns.

9:12 AM

Devised feature engineering strategy and cross-tabulation approach.

1.
2.
3.
4.

Here's a new code block for feature engineering and creating a cross-tabulation table:

```
## Feature Engineering: Create "era" column


df3 <- df2 |>

 mutate(

  era = if_else(erected < 1900, "old", "new")

 )


# Preview the new column
```

```
df3 |>
  select(identif, erected, era, material) |>
  head(10) |>
  gt()


# Cross-tabulation: era (rows) x material (columns)
era_material_table <- df3 |>
  count(era, material) |>
  pivot_wider(
    names_from = material,
    values_from = n,
    values_fill = 0
  )


era_material_table |>
  gt() |>
  tab_header(
    title = "Bridge Material by Era",
    subtitle = "old = before 1900, new = 1900 and after"
  ) |>
  cols_label(era = "Era")
```

What this code does:

1. **Feature engineering:**
    - mutate() creates the new era column
    - if_else() assigns "old" for erected < 1900, otherwise "new"
2. **Cross-tabulation table:**

- count(era, material) counts occurrences for each combination
- pivot_wider() reshapes the data so materials become columns
- values_fill = 0 replaces any missing combinations with zero
- gt() formats it as a nice table with a header explaining the era definitions

This gives you a clear view of how bridge materials changed between the pre-1900 and post-1900 eras.

Next, provide code to convert all instances of river with value "Y" to value "M", and check my work by redisplaying the table of river values. In the table, remove the redundant second percentage column.

9:20 AM

Devised strategy to transform river values and streamline table display.

1.
2.
3.

Here's a new code block to convert river values and verify the change:

```
## Convert river value "Y" to "M"


df3 <- df3 |>
  mutate(
    river = if_else(river == "Y", "M", river)
  )


# Verify the change with updated river summary table
river_summary <- df3 |>
  count(river) |>
  mutate(
    percentage = n / sum(n) * 100
```

```
 )


river_summary |>
 gt() |>
 cols_label(
  river = "River",
  n = "Count",
  percentage = "Percentage"
 ) |>
 fmt_number(columns = percentage, decimals = 1) |>
 fmt_suffix(columns = percentage, suffix = "%")
```

What this code does:

1. **Convert "Y" to "M":**
    - mutate() with if_else() replaces "Y" values with "M"
    - All other river values remain unchanged
2. **Verification table:**
    - Recreates the count summary to confirm "Y" no longer exists
    - The count for "M" should now include the former "Y" bridges
    - Removed the percentage_label column, using fmt_suffix() instead to append "%" directly to the percentage values

You should now see only "A", "M", and "O" in the river column, with no "Y" entries.

I got an error with fmt_suffix. Does this exist at part of gt? Please either fix, or remove the line of code.

9:25 AM

You're right, fmt_suffix() doesn't exist in gt. Here's the corrected code:

```
## Convert river value "Y" to "M"
```

```
df3 <- df3 |>
  mutate(
    river = if_else(river == "Y", "M", river)
  )

# Verify the change with updated river summary table
river_summary <- df3 |>
  count(river) |>
  mutate(
    percentage = n / sum(n) * 100
  )

river_summary |>
  gt() |>
  cols_label(
    river = "River",
    n = "Count",
    percentage = "Percentage"
  ) |>
  fmt_percent(columns = percentage, scale_values = FALSE, decimals = 1)
```

What I changed:

- Replaced the non-existent fmt_suffix() with fmt_percent()
- scale_values = FALSE tells gt the values are already in percentage form (not decimals), so it just formats them with the "%" symbol

Create a map of the Pittsburgh area, that includes the three rivers, with the river names and the city of Pittsburgh itself clearly labeled. Show where the Youghiogheny River enters the Monongahela river.

9:36 AM

Check available skills for creating this map

I'll create an interactive map using the leaflet package, which is well-suited for displaying geographic data in R:

```
## Map of Pittsburgh Area with Three Rivers

# install.packages("leaflet")
library(leaflet)

# Key geographic coordinates
pittsburgh_point <- c(-79.9959, 40.4406)  # Downtown Pittsburgh (The Point)
yough_confluence <- c(-79.8517, 40.3475)  # McKeesport - where Yough meets Mon

# Create the map
leaflet() |>
  setView(lng = -79.92, lat = 40.40, zoom = 11) |>
  addTiles() |>

  # Mark Pittsburgh (The Point - confluence of three rivers)
  addMarkers(
    lng = pittsburgh_point[1],
    lat = pittsburgh_point[2],
    popup = "Pittsburgh - The Point<br>Confluence of Three Rivers"
  ) |>

  # Mark Youghiogheny confluence with Monongahela
```

```
addMarkers(
  lng = yough_confluence[1],
  lat = yough_confluence[2],
  popup = "McKeesport<br>Youghiogheny enters Monongahela"
) |>

# Label the rivers
addLabelOnlyMarkers(
  lng = c(-80.05, -79.93, -79.98, -79.82),
  lat = c(40.45, 40.38, 40.48, 40.33),
  label = c("Ohio River", "Monongahela River", "Allegheny River", "Youghiogheny River"),
  labelOptions = labelOptions(
    noHide = TRUE,
    direction = "center",
    textOnly = TRUE,
    style = list(
      "color" = "darkblue",
      "font-weight" = "bold",
      "font-size" = "12px"
    )
  )
) |>

# Add a title
addControl(
  html = "<h4 style='margin:0; padding:5px; background:white; border-radius:5px;'>Pittsburgh's Three Rivers</h4>",
  position = "topright"
 )
```

What this code does:

1. **Uses leaflet** - an interactive mapping library that works well in R/Quarto documents
2. **Key locations marked:**
    - Pittsburgh's "Point" - where the Allegheny and Monongahela meet to form the Ohio River
    - McKeesport - where the Youghiogheny River enters the Monongahela (southeast of Pittsburgh)
3. **River labels** - placed approximately along each river's path with bold blue text
4. **Interactive features:**
    - Users can zoom in/out and pan around
    - Clicking markers shows popup information
    - Base map tiles show the actual river geography

Note: If leaflet is not installed, uncomment the install.packages("leaflet") line first.

Describe three specific discrete next steps that you would recommend to make this analysis more complete and/or more interesting?

9:50 AM

Here are three specific discrete next steps I would recommend:

1. Analyze Bridge Material Trends Over Time

Create a visualization showing how bridge construction materials changed across decades. This would reveal the transition from wood to iron to steel as construction technology evolved.

```
# Group by decade and material, then visualize the shift

df3 |>

 mutate(decade = floor(erected / 10) * 10) |>
```

```
  count(decade, material) |>
  ggplot(aes(x = decade, y = n, fill = material)) +
  geom_col(position = "stack") +
  labs(
    title = "Bridge Construction Materials by Decade",
    x = "Decade",
    y = "Number of Bridges"
  ) +
  theme_minimal()
```

2. Compare Bridge Characteristics by River

Investigate whether different rivers have different bridge profiles. For example, are bridges on the Ohio River newer or built with different materials than those on the Monongahela? This could reveal historical development patterns across different parts of Pittsburgh.

```
# Cross-tabulation of river by material and era
df3 |>
  count(river, era, material) |>
  ggplot(aes(x = river, y = n, fill = material)) +
  geom_col(position = "dodge") +
  facet_wrap(~era) +
  labs(title = "Bridge Materials by River and Era") +
  theme_minimal()
```

3. Geocode Bridges and Plot on Interactive Map

Augment the dataset by obtaining latitude/longitude coordinates for each bridge (using the bridge names or locations), then plot them on the leaflet map with color-coded markers by material or era. This would create a compelling visual showing the spatial distribution of bridge types across the Pittsburgh region and could reveal geographic clustering patterns.

Claude is AI and can make mistake