

A Real Example of How I Used ChatGPT

Software Bugs Case Study

Rivers “Riv” Jenkins

Email: rtjenkins@tecolote.com

Phone: 540-299-8199



04.15.25

Problem Context

- A DoD program office wants to create funding request for software sustainment
- Main piece of software sustainment is fixing software bugs
 - I have number of hours spent on each bug fix for past 8 years
 - After some back and forth, the program office produces number of outstanding bugs in the “backlog” for past year and a half – there are over 500 software bugs requiring a fix
- Need to estimate labor hours per year
 - If I can project labor hours fixing software bugs per year, I can multiply by burdened labor rates to estimate cost per year in future
 - But labor hours per year depends on desired speed of reducing backlog
- How to take this data and visualize “get well” scenarios?
 - Ultimately, I will need the program office to decide how quickly they want to reduce the backlog of outstanding software bugs

ChatGPT (Data Analysis & Report AI)

Prompt: Pretend you are an expert data visualization specialist. Suppose we have a system that is backed up with 500 items in the queue to process. Each month 5 new items are added to the queue. What are some visualizations showing possible "get well" scenarios at various levels of funding (i.e. speed of processing)?

Answer: To visualize possible "get well" scenarios for a system backed up with 500 items in the queue and 5 new items being added each month, we can consider the following scenarios based on various levels of funding, which translate to different speeds of processing. Here are some visualizations that can help illustrate the impact of these scenarios:

1. Line Chart Showing Queue Reduction Over Time:

- **Purpose:** To show how the size of the queue changes over time with different processing rates.
- **Variables:** Months on the x-axis, Number of items in the queue on the y-axis.
- **Scenarios:** Different lines for different processing rates (e.g., 10 items/month, 20 items/month, 50 items/month).

2. Bar Chart Showing Monthly Processing vs. Incoming Items:

- **Purpose:** To compare the number of items processed each month against the number of new items added.
- **Variables:** Months on the x-axis, Number of items on the y-axis.
- **Scenarios:** Bars for processed items (different heights for different rates) and a constant bar for 5 new items.

3. Stacked Area Chart Showing Cumulative Processed Items vs. New Items:

- **Purpose:** To illustrate the cumulative effect of processing over time and how it balances with the incoming items.
- **Variables:** Months on the x-axis, Cumulative number of items on the y-axis.
- **Scenarios:** Stacked areas representing processed items (different rates) and new items.

4. Heatmap Showing Processing Pressure Over Time:

- **Purpose:** To show the intensity of the queue backlog over time with different processing speeds.
- **Variables:** Months on the x-axis, Processing rates on the y-axis, Color intensity representing queue size.
- **Scenarios:** Different colors indicating the level of queue backlog.

Answers tend to be verbose and start with paraphrase of the question

Do you want me to proceed with creating these visualizations?

Data Visualization Option #1: Line Chart

Prompt: Show example charts

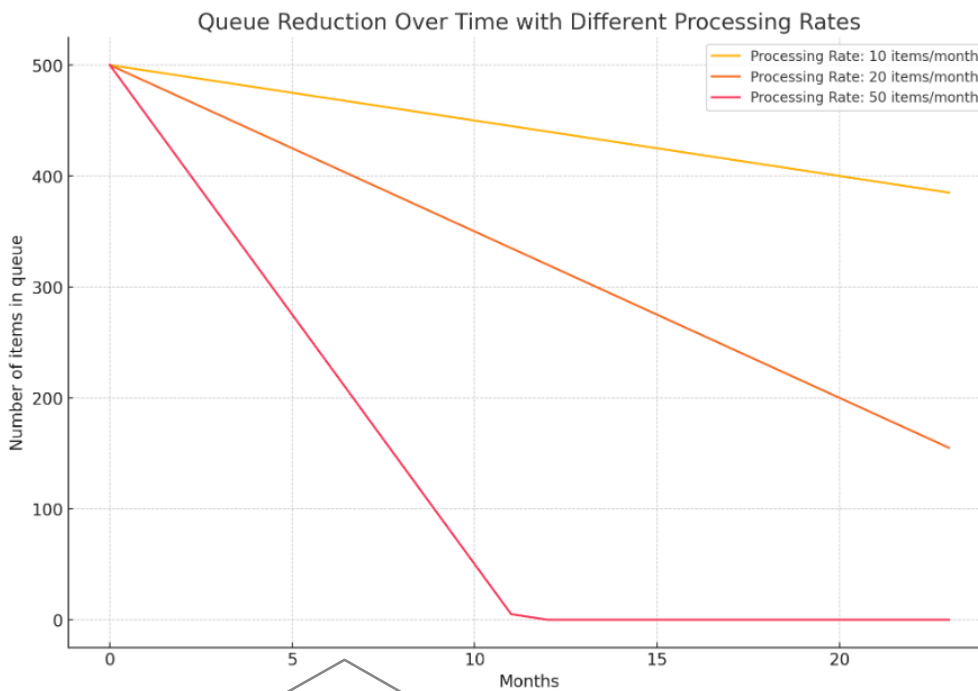
Answer:

1. Line Chart Showing Queue Reduction Over Time

Let's create the forecasting line chart.

```
python Copy code  
  
import matplotlib.pyplot as plt  
import numpy as np  
  
# Parameters  
months = np.arange(0, 24) # 2 years  
  
# Forecasting Data  
forecast_data = {}  
for rate in processing_rates:  
    queue_size = initial_queue_size  
    forecast_data[rate] = []  
    for month in months:  
        forecast_data[rate].append(queue_size)  
        queue_size = max(queue_size - rate + new_items_per_month, 0)  
        if queue_size == 0:  
            break  
  
# Plotting  
plt.figure(figsize=(12, 8))  
for rate in processing_rates:  
    plt.plot(months[:len(forecast_data[rate])], forecast_data[rate], label=f'Processing Ra  
  
plt.xlabel('Months')  
plt.ylabel('Number of Items in Queue')  
plt.title('Forecasting Queue Clearance with Different Processing Rates')  
plt.legend()  
plt.grid(True)  
plt.show()
```

Python code



Probably the visualization I would have eventually come up with on my own

Data Visualization Option #2: Bar Chart

Prompt: Show example charts

Answer:

2. Bar Chart Showing Monthly Processing vs. Incoming Items

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

# Parameters
months = np.arange(1, 13) # 1 year

# Bar Chart Data
processed_items = {
    10: [10] * len(months),
    20: [20] * len(months),
    50: [50] * len(months)
}
incoming_items = [new_items_per_month] * len(months)

# Plotting
bar_width = 0.2
index = np.arange(len(months))

fig, ax = plt.subplots(figsize=(12, 8))
for i, rate in enumerate(processing_rates):
    ax.bar(index + i * bar_width, processed_items[rate], bar_width, label=f'Processed: {rate}')

ax.bar(index + len(processing_rates) * bar_width, incoming_items, bar_width, label='New Items Added')

ax.set_xlabel('Months')
ax.set_ylabel('Number of Items')
ax.set_title('Monthly Processing vs. Incoming Items')
ax.set_xticks(index + bar_width)
ax.set_xticklabels(months)
ax.legend()
ax.grid(True)

plt.show()
```



Would be more interesting if new items added or processing rates varied over time

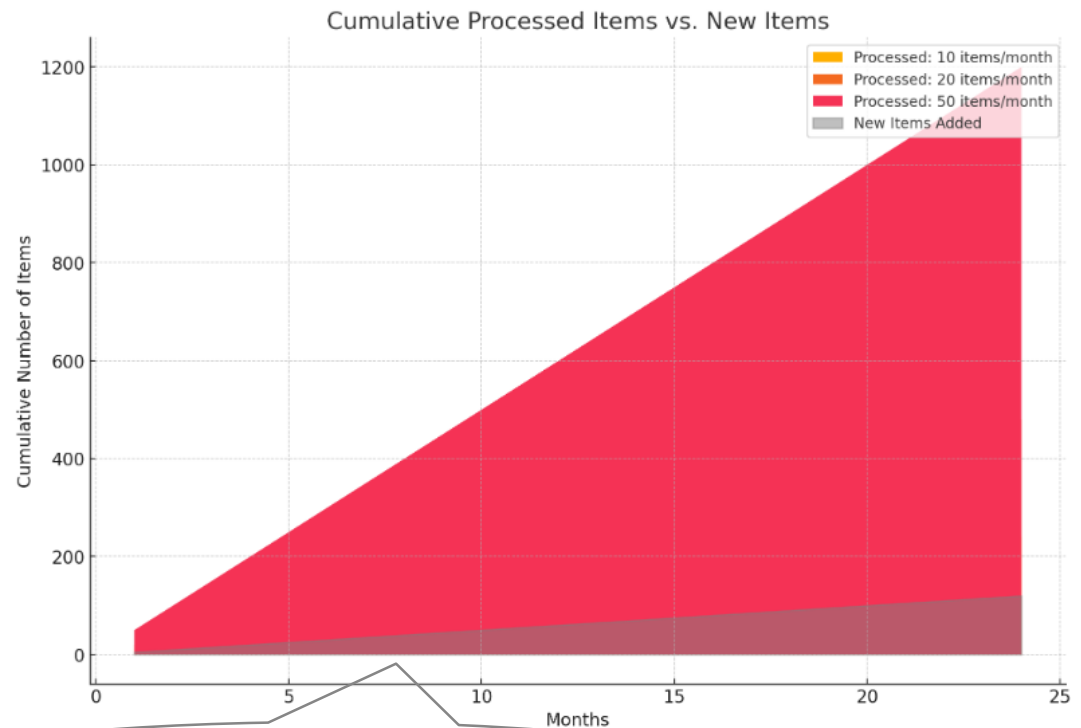
Data Visualization Option #3: Sand Chart

Prompt: Show example charts

Answer:

3. Stacked Area Chart Showing Cumulative Processed Items vs. New Items

```
python Copy code  
  
import matplotlib.pyplot as plt  
import numpy as np  
  
# Parameters  
months = np.arange(1, 25) # 2 years  
  
# Stacked Area Chart Data  
cumulative_processed = {  
    10: np.cumsum([10] * len(months)),  
    20: np.cumsum([20] * len(months)),  
    50: np.cumsum([50] * len(months))  
}  
cumulative_new = np.cumsum([new_items_per_month] * len(months))  
  
# Plotting  
fig, ax = plt.subplots(figsize=(12, 8))  
for rate in processing_rates:  
    ax.fill_between(months, cumulative_processed[rate], label=f'Processed: {rate} items/month', color=rate, alpha=0.5)  
  
ax.fill_between(months, cumulative_new, label='New Items Added', color='grey', alpha=0.5)  
  
ax.set_xlabel('Months')  
ax.set_ylabel('Cumulative Number of Items')  
ax.set_title('Cumulative Processed Items vs. New Items')  
ax.legend()  
ax.grid(True)  
  
plt.show()
```



Visualization error? Some colors in legend do not appear in chart

Data Visualization Option #4: Heatmap

Prompt: Show example charts

Answer:

4. Heatmap Showing Processing Pressure Over Time

```
python Copy code
import matplotlib.pyplot as plt
import numpy as np

# Parameters
months = np.arange(0, 24) # 2 years
processing_rates = np.array([10, 20, 50])

# Heatmap Data
heatmap_data = np.zeros((len(processing_rates), len(months)))

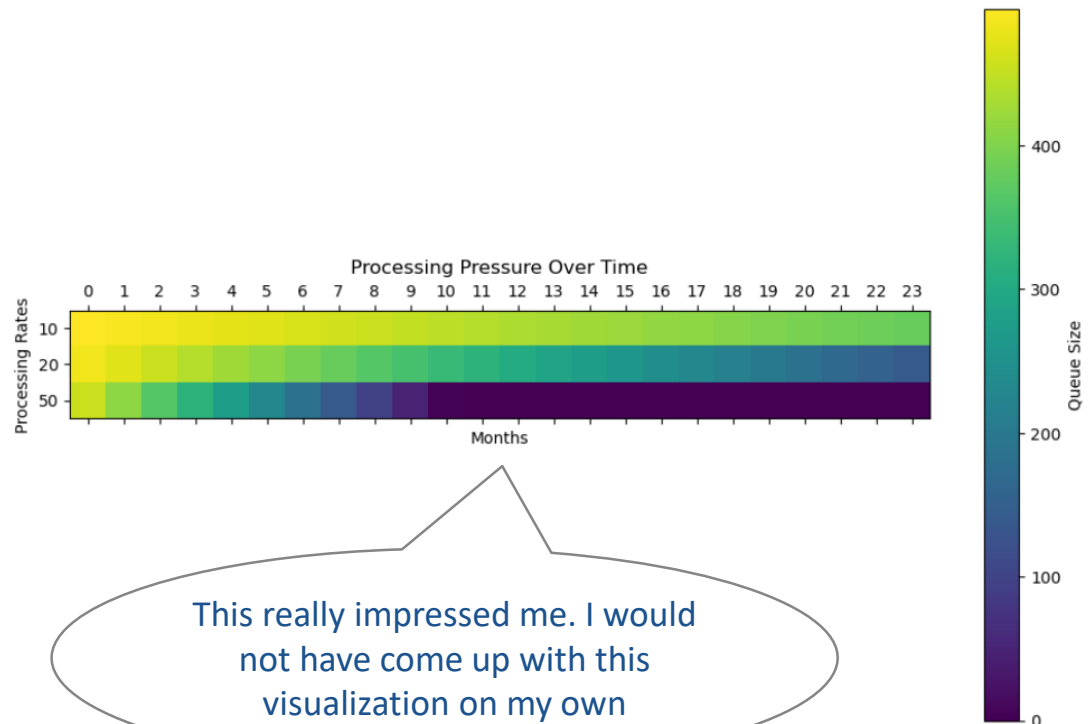
for i, rate in enumerate(processing_rates):
    queue_size = initial_queue_size
    for j, month in enumerate(months):
        queue_size = max(queue_size - rate + new_items_per_month, 0)
        heatmap_data[i, j] = queue_size

# Plotting
fig, ax = plt.subplots(figsize=(12, 8))
cax = ax.matshow(heatmap_data, cmap='viridis')

ax.set_xticks(np.arange(len(months)))
ax.set_yticks(np.arange(len(processing_rates)))
ax.set_xticklabels(months)
ax.set_yticklabels(processing_rates)

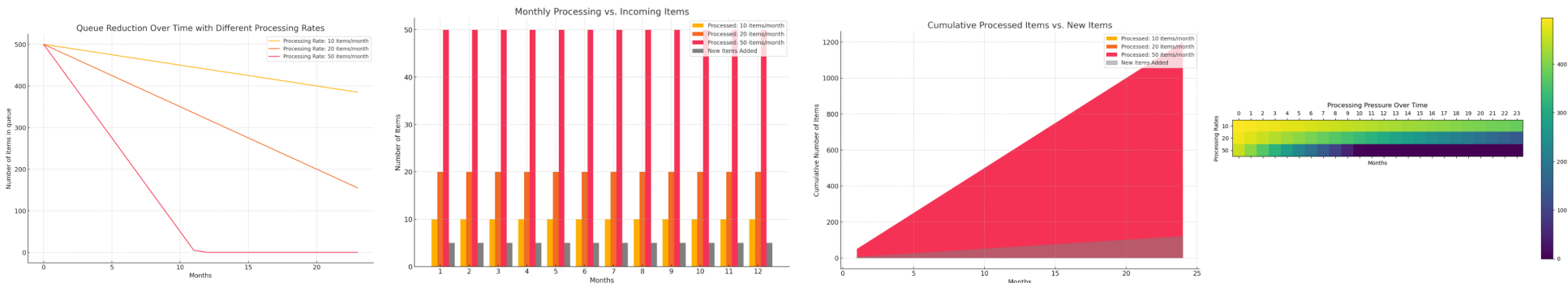
ax.set_xlabel('Months')
ax.set_ylabel('Processing Rates')
ax.set_title('Processing Pressure Over Time')

plt.colorbar(cax, label='Queue Size')
plt.show()
```



Data Visualization Selection

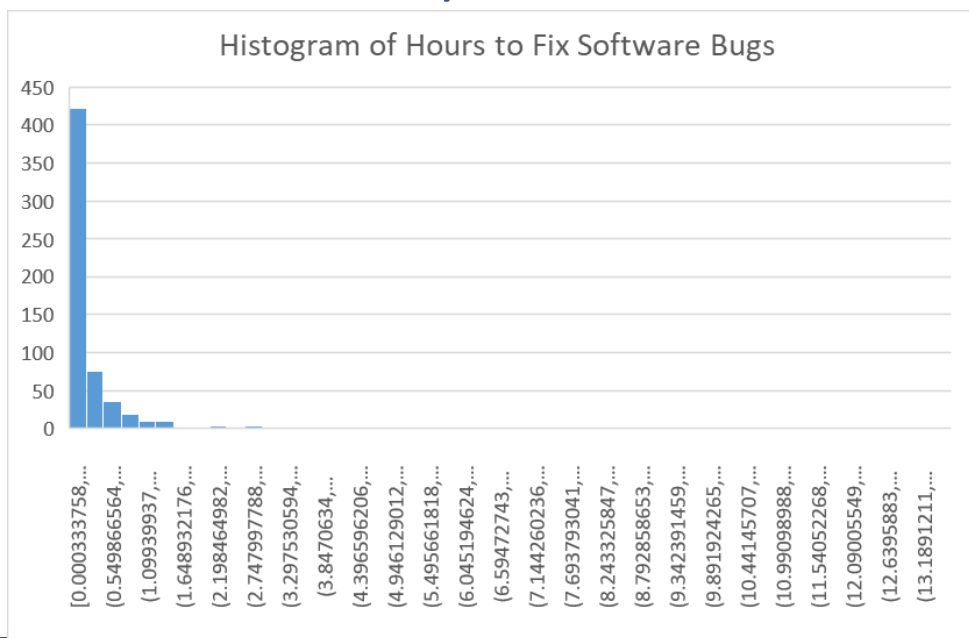
■ Answer:



- Prompt: I like visualizations 1 and 4. Use those going forward.
- Answer: Great choice! Visualizations 1 (Line Chart Showing Queue Reduction Over Time) and 4 (Heatmap Showing Processing Pressure Over Time) will provide valuable insights into how different processing rates impact the queue over time.

Adding Uncertainty to the Visualization

- Have data showing number of hours to fix almost 600 software bugs
 - Data in this presentation is transformed to obfuscate original proprietary data
- Looked like an exponential distribution
 - Spoiler Alert: I was wrong
- Assuming I can fit an exponential distribution to this data and then use that distribution to run a simulation, how could I add the uncertainty results from the simulation to my selected data visualizations?



ChatGPT

- **Prompt:** How would you modify these visualizations if the time to process an item is stochastic following an exponential distribution?
- **Answer:** If the time to process an item follows an exponential distribution, the processing rate becomes stochastic rather than deterministic. This introduces variability in the number of items processed each month. We can modify the visualizations to account for this stochastic nature by simulating the processing times. Here's how the Python code would look for these modifications:

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

# Parameters
initial_queue_size = 500
new_items_per_month = 5
mean_processing_rates = [10, 20, 50] # Mean processing rates
months = np.arange(1, 24) # 2 years
num_simulations = 1000

# Function to simulate queue reduction
def simulate_queue_reduction(mean_rate, months, initial_queue_size, new_items_per_month):
    queue_sizes = np.zeros((num_simulations, len(months)))
    for i in range(num_simulations):
        queue_size = initial_queue_size
        for j, month in enumerate(months):
            processed_items = np.random.exponential(mean_rate)
            queue_size = max(queue_size - processed_items + new_items_per_month, 0)
            queue_sizes[i, j] = queue_size
        return queue_sizes

# Simulate for each mean processing rate
simulated_queues = {rate: simulate_queue_reduction(rate, months, initial_queue_size, new_items_per_month)
                    for rate in mean_processing_rates}

# Calculate mean and confidence intervals for the line chart
mean_queue_sizes = {rate: np.mean(simulated_queues[rate], axis=0) for rate in mean_processing_rates}
conf_intervals = {rate: 1.96 * np.std(simulated_queues[rate], axis=0) for rate in mean_processing_rates}

# Plotting Line Chart Showing Queue Reduction Over Time
plt.figure(figsize=(12, 8))
for rate in mean_processing_rates:
    plt.plot(months, mean_queue_sizes[rate], label=f'Mean Processing Rate: {rate} items/month')
    plt.fill_between(months, mean_queue_sizes[rate] - conf_intervals[rate],
                    mean_queue_sizes[rate] + conf_intervals[rate], alpha=0.2)

plt.xlabel('Months')
plt.ylabel('Number of Items in Queue')
plt.title('Queue Reduction Over Time with Stochastic Processing Rates')
plt.legend()
plt.grid(True)
plt.show()

# Calculate average queue size and standard deviation for the heatmap
avg_queue_sizes = np.zeros((len(mean_processing_rates), len(months)))
std_queue_sizes = np.zeros((len(mean_processing_rates), len(months)))

for i, rate in enumerate(mean_processing_rates):
    avg_queue_sizes[i, :] = np.mean(simulated_queues[rate], axis=0)
    std_queue_sizes[i, :] = np.std(simulated_queues[rate], axis=0)

# Plotting Heatmap Showing Processing Pressure Over Time
fig, ax = plt.subplots(figsize=(12, 8))
cax = ax.imshow(avg_queue_sizes, cmap='viridis')

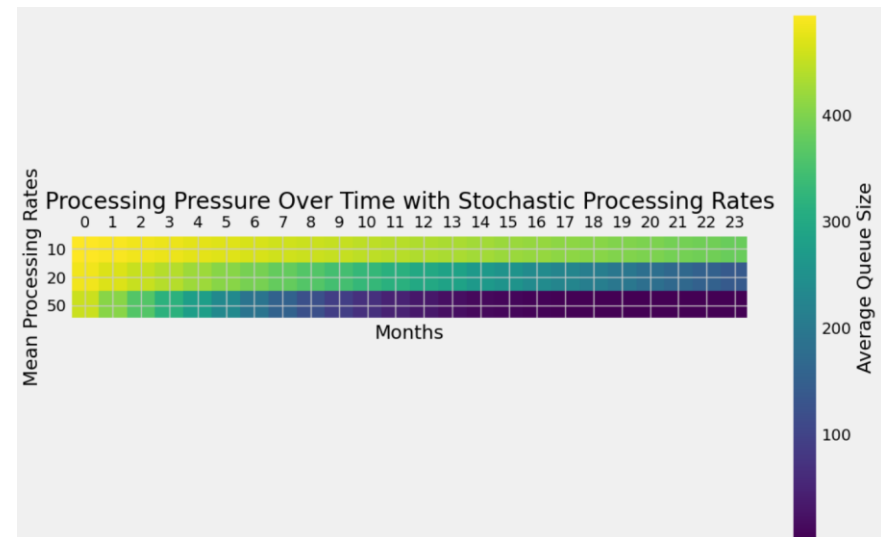
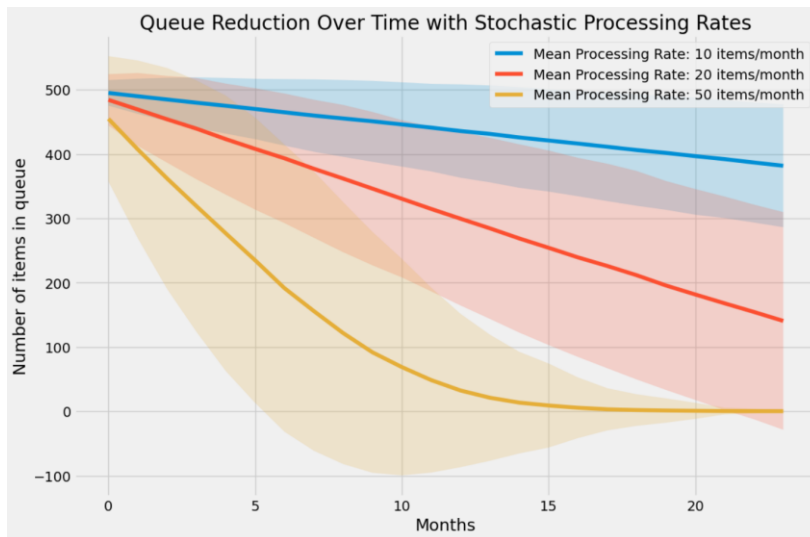
ax.set_xticks(np.arange(len(months)))
ax.set_yticks(np.arange(len(mean_processing_rates)))
ax.set_xticklabels(months)
ax.set_yticklabels(mean_processing_rates)

ax.set_xlabel('Months')
ax.set_ylabel('Mean Processing Rates')
ax.set_title('Processing Pressure Over Time with Stochastic Processing Rates')
fig.colorbar(cax, ax=ax, label='Average Queue Size')

plt.show()
```

Visualizations with Stochasticity

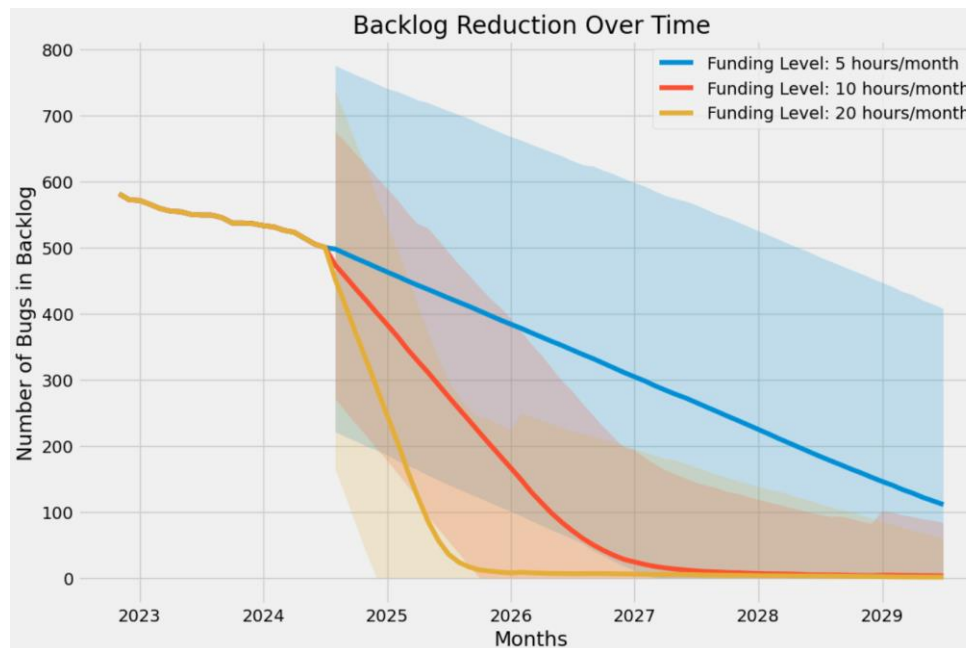
- The free version would no longer generate graphs, so I copied and pasted the Python code into my local interpreter to generate the visuals from ChatGPT
 - The code ran beautifully!
 - Queue size does dip below zero, but that can be fixed



- The heatmap did not really capture the uncertainty (just shows average queue size), so the line chart was the clear winning visualization

Final Visualization Result

- Plugged real data into visualization
- Allowed program office to select a level of effort they were comfortable with
 - They picked 10 hours/month



- Assuming a labor rate of \$250 per hour, that equates to a funding requirement of \$30k each year for two years
 - Actual funding requirement (using real data) was much higher than this
 - Actual cost estimate properly accounted for inflation

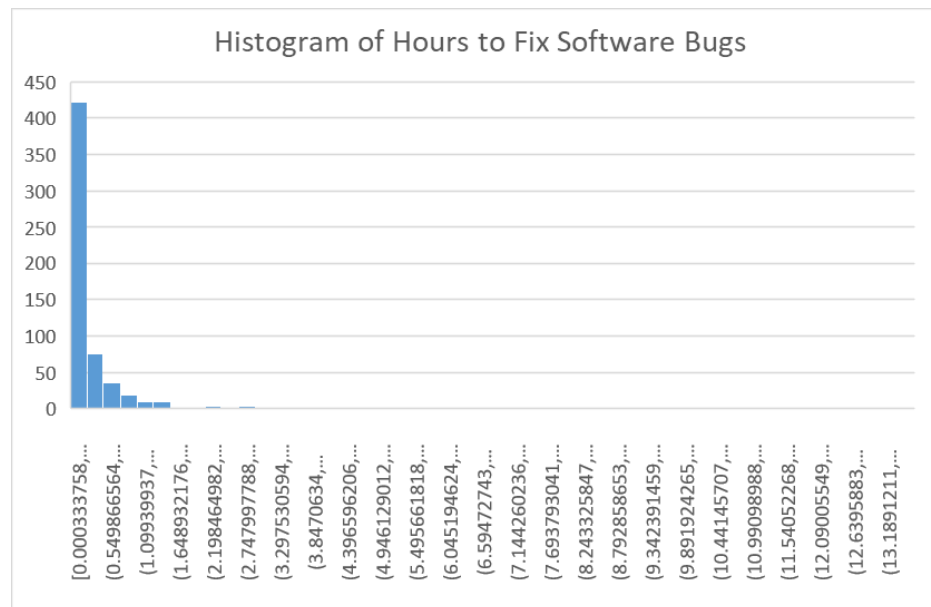
ChatGPT – Takeaways Based on Case Study

- Gives verbose answers
- Can jump start basic programming tasks
- Seems to be good at brainstorming
 - Generating data visualization options
 - How to add uncertainty
- Can give a starting point which user can further refine
 - Refining could be with ChatGPT's help (like asking it to add stochasticity)
 - Or without (like modifying final graphic to show historical data and not dip below 0)
- Don't expect perfection
 - Sand chart did not seem to be layered properly
 - Queue size dipped below zero when uncertainty applied

Bonus: Not Exponential?

Mean	$\frac{1}{\lambda}$
Median	$\frac{\ln 2}{\lambda}$
Variance	$\frac{1}{\lambda^2}$

- Data sample mean = 0.389, variance = 1, median = 0.172
- If I try to calculate lambda from each of these, end up with 2.57, 1, and 4.03 – all very different distributions, so maybe exponential does not fit so well after all
- But if not exponential, then what?



ChatGPT? Or Google + StackOverflow?

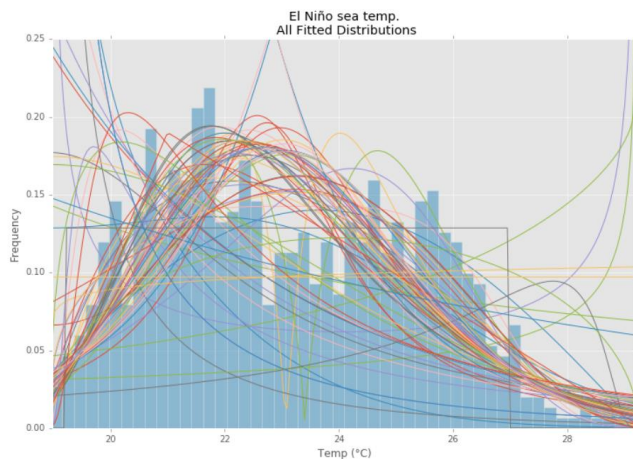
■ ChatGPT

- **Prompt:** How can I fit a distribution to data in python?
- **Answer:** Here's a step-by-step guide using libraries like scipy and matplotlib (with example code at each step)
 1. Import Required Libraries
 2. Generate or Load Data
 3. Visualize the Data
 4. Choose a Distribution and Fit It
 5. Compare the Fitted Distribution with Data

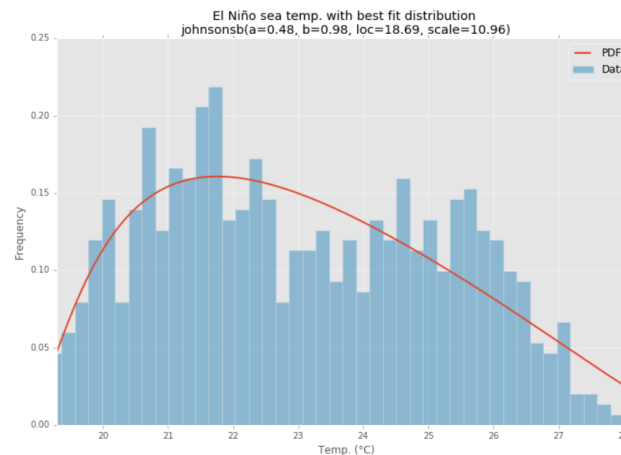
■ Google

- **Prompt:** fit distribution to data python
- **Answer:** First StackOverflow result: Distribution Fitting with Sum of Square Error (SSE)

All Distributions



Best Fit Distribution



Example Code

```
%matplotlib inline
import warnings
import numpy as np
import pandas as pd
import scipy.stats as st
import statsmodels.api as sm
from scipy.stats._continuous_distns import _distn_names
import matplotlib
import matplotlib.pyplot as plt

matplotlib.rcParams['figure.figsize'] = (16.0, 12.0)
matplotlib.style.use('ggplot')

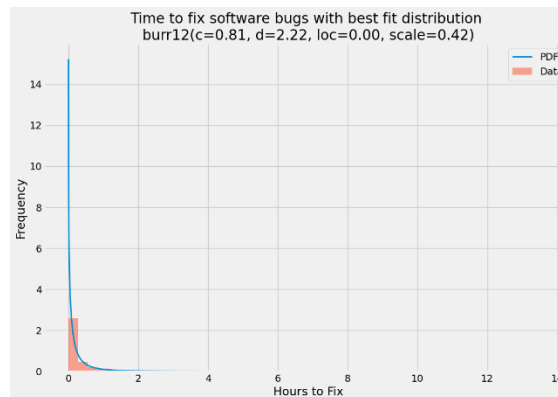
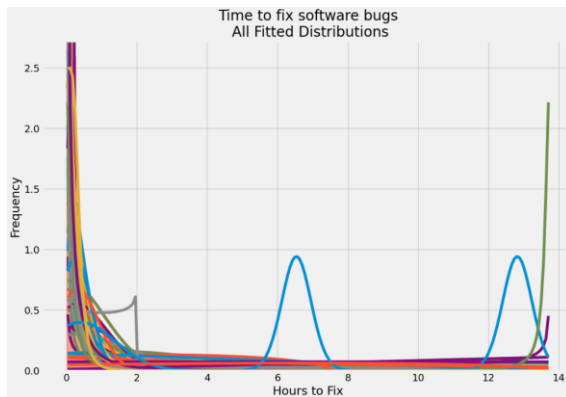
# Create models from data
def best_fit_distribution(data, bins=200, ax=None):
    """Model data by finding best fit distribution to data"""
    # Get histogram of original data
    y, x = np.histogram(data, bins=bins, density=True)
    x = (x + np.roll(x, -1))[:-1] / 2.0

    # Best holders
    best_distributions = []

    # Estimate distribution parameters from data
    for ii, distribution in enumerate([d for d in _distn_names if not d in ['levy_stable']]):
        print("{} / {}".format(ii+1, len(_distn_names), distribution))
```

Google wins this round (or do I just need better prompts?)

- Plug in own data and immediately get results (Burr distribution is best fitting)



- Primary reason may be that I didn't ask ChatGPT the right question
 - **Prompt:** What if I don't know which distributions to try?
 - **Answer:** If you're unsure which distributions to try, you can use a library like `fitter`, `scipy.stats`, or `distfit` to automate the process of testing multiple distributions and identifying the best fit for your data. These libraries compare a range of common distributions and rank them based on how well they match your data.
- When run ChatGPT Fitter package code with actual data and same setup parameters, get same results as above

Option 1: Using fitter

The `fitter` library is excellent for trying a wide range of distributions.

Installation

```
bash  
pip install fitter
```

Copy code

Example Code

```
python  
  
from fitter import Fitter  
  
# Your data  
data = np.random.normal(5, 2, 1000) # Replace with your actual data  
  
# Fit multiple distributions  
f = Fitter(data, distributions=None) # Distributions=None tries all available ones  
f.fit()  
  
# Summary of the best fits  
print(f.summary()) # Shows the top-ranked distributions  
  
# Plot the best-fitting distributions  
f.plot()
```

Copy code

Output

The `f.summary()` method shows the ranked distributions, with the best fit at the top based on criteria like sum of squared errors (SSE). The plot visualizes how well the distributions match the data.