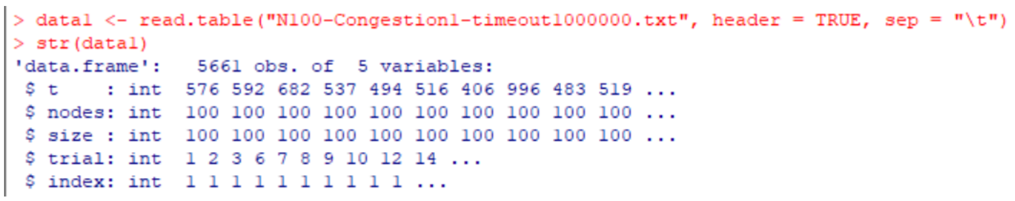
# Data Analytics Midterm Project

Erika Dommer

March 27, 2025

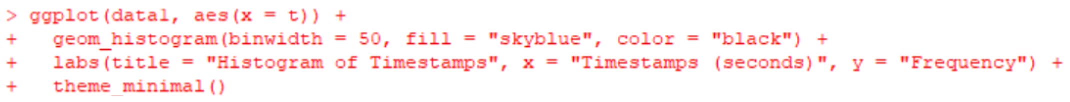
Purppose:
The purpose of this analysis is to explore the application of coupled equations within Traffic This analysis aims to explore how Traffic Management Systems use coupled equations to manage real-world traffic flow and efficiency. It will look into how these systems can reduce congestion, improve operations, and their overall impact on socioeconomic development.

Data1 – N100 Congestion 1

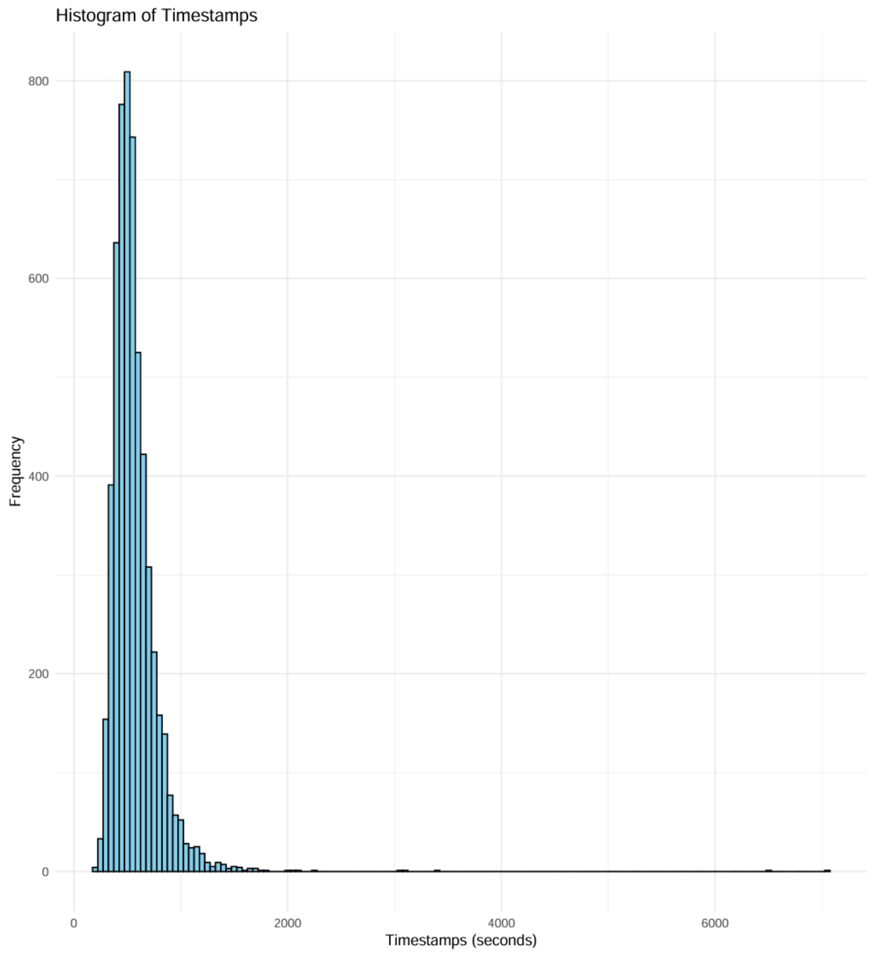


```
> data1 <- read.table("N100-Congestion1-timeout1000000.txt", header = TRUE, sep = "\t")
> str(data1)
'data.frame':   5661 obs. of  5 variables:
 $ t    : int  576 592 682 537 494 516 406 996 483 519 ...
 $ nodes: int  100 100 100 100 100 100 100 100 100 100 ...
 $ size : int  100 100 100 100 100 100 100 100 100 100 ...
 $ trial: int  1 2 3 6 7 8 9 10 12 14 ...
 $ index: int  1 1 1 1 1 1 1 1 1 1 ...
```

Histogram (Note: binwidth = 50 means each bin spans an interval of 50 seconds, kept at 50 show more precise distributions.
Ex: Bin 2 (550–600): Count = 2 (values: 576, 592)

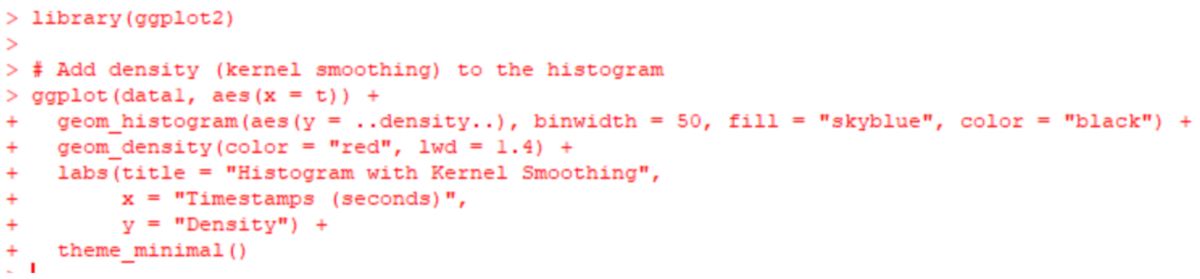


```
> ggplot(data1, aes(x = t)) +
+   geom_histogram(binwidth = 50, fill = "skyblue", color = "black") +
+   labs(title = "Histogram of Timestamps", x = "Timestamps (seconds)", y = "Frequency") +
+   theme_minimal()
```

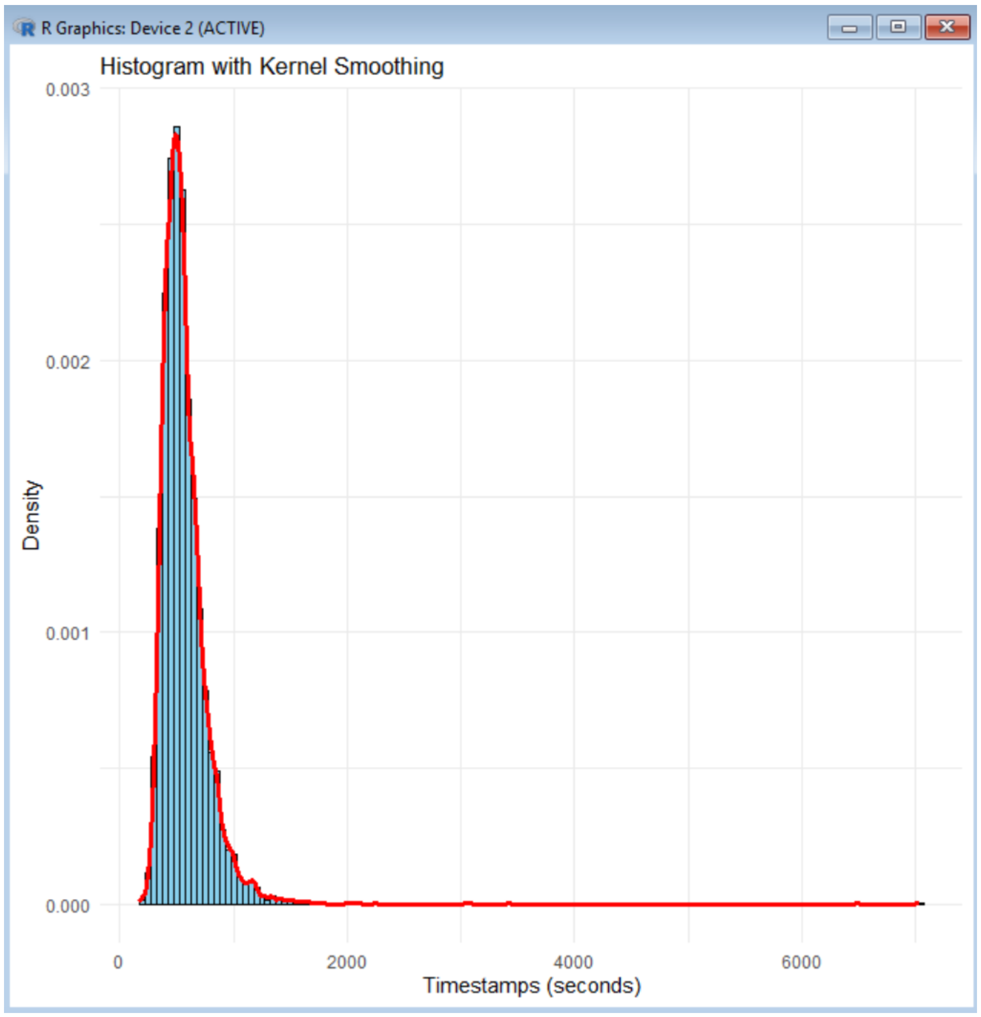


To smooth out the distribution to reveal trends, I have opted to use Kernel Smoothing fit.

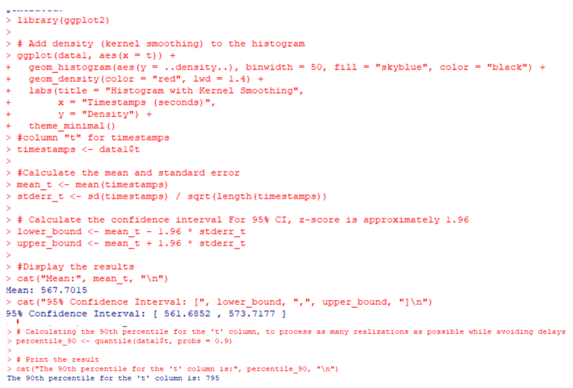
Applied Kernel Density Estimation (KDE) to smooth histogram and fit a curve.



```
> library(ggplot2)
>
> # Add density (kernel smoothing) to the histogram
> ggplot(data1, aes(x = t)) +
+   geom_histogram(aes(y = ..density..), binwidth = 50, fill = "skyblue", color = "black") +
+   geom_density(color = "red", lwd = 1.4) +
+   labs(title = "Histogram with Kernel Smoothing",
+        x = "Timestamps (seconds)",
+        y = "Density") +
+   theme_minimal()
```



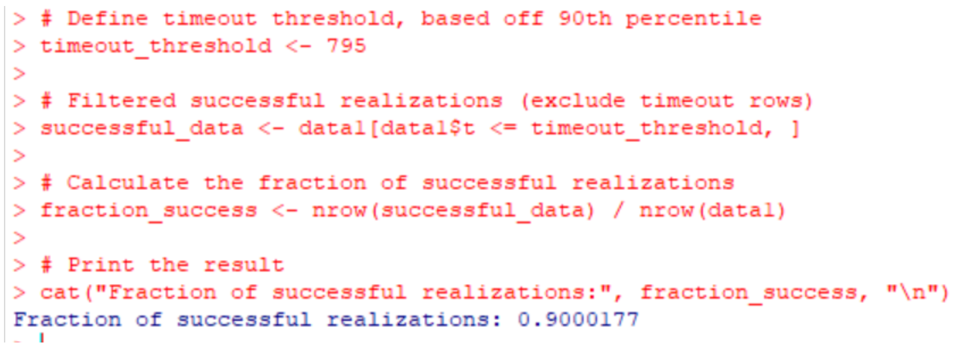
Mean: 567.7015
95% Confidence Interval: [ 561.6852 , 573.7177 ]



```
> library(ggplot2)
>
> # Add density (kernel smoothing) to the histogram
> ggplot(datal, aes(x = t)) +
+   geom_histogram(aes(y = ..density..), binwidth = 50, fill = "skyblue", color = "black") +
+   geom_density(color = "red", lwd = 1.4) +
+   labs(title = "Histogram with Kernel Smoothing",
+        x = "Timestamps (seconds)",
+        y = "Density") +
+   theme_minimal()
> #column "t" for timestamps
> timestamps <- datal$t
>
> #Calculate the mean and standard error
> mean_t <- mean(timestamps)
> stderr_t <- sd(timestamps) / sqrt(length(timestamps))
>
> # Calculate the confidence interval For 95% CI, z-score is approximately 1.96
> lower_bound <- mean_t - 1.96 * stderr_t
> upper_bound <- mean_t + 1.96 * stderr_t
>
> #Display the results
> cat("Mean:", mean_t, "\n")
Mean: 567.7015
> cat("95% Confidence Interval: [", lower_bound, ",", upper_bound, "]\n")
95% Confidence Interval: [ 561.6852 , 573.7177 ]
> # Calculating the 90th percentile for the 't' column, to process as many realizations as possible while avoiding delays
> percentile_90 <- quantile(datal$t, probs = 0.9)
>
> # Print the result
> cat("The 90th percentile for the 't' column is:", percentile_90, "\n")
The 90th percentile for the 't' column is: 795
```

To consider the following:
Timeout - is the total number of seconds the system waits for a solution. If no solution is found until this instant (= 1000000), a line is skipped in the dataset, and the system moves to the next trial or index.

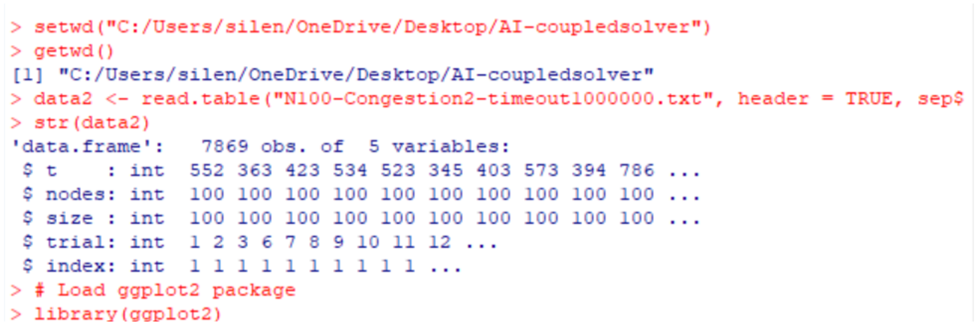
1. Calculated the 90th percentile for the 't' column, to process as many realizations as possible while avoiding delays.
2. Identified Timeout Rows: Rows where the t column value exceeds the timeout threshold should be considered as unsuccessful realizations.
3. Excluded Timeout Rows: Filter out those rows when calculating the fraction of realizations that achieve a solution.
4. Calculated the Fraction**:** From the filtered dataset, calculate the fraction of successful realizations.



```
> # Define timeout threshold, based off 90th percentile
> timeout_threshold <- 795
>
> # Filtered successful realizations (exclude timeout rows)
> successful_data <- data1[data1$t <= timeout_threshold, ]
>
> # Calculate the fraction of successful realizations
> fraction_success <- nrow(successful_data) / nrow(data1)
>
> # Print the result
> cat("Fraction of successful realizations:", fraction_success, "\n")
Fraction of successful realizations: 0.9000177
```

This means that approx.. 90% if the realizations from dataset data1 (Data1 – N100 Congestion 1) were successful, they met the defined criteria for achieving the solution. The remaining approx.. 10% either timed out or failed to produce a solution. The majority of the processes captured in data 1 performed effectively within the defined threshold of the 90$^{th}$ percentile. The high fraction reflects strong reliability in my model.

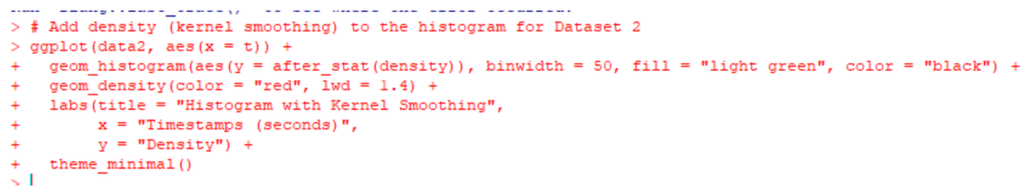
Data2 – N100 Congestion 2
Loaded data2 in R



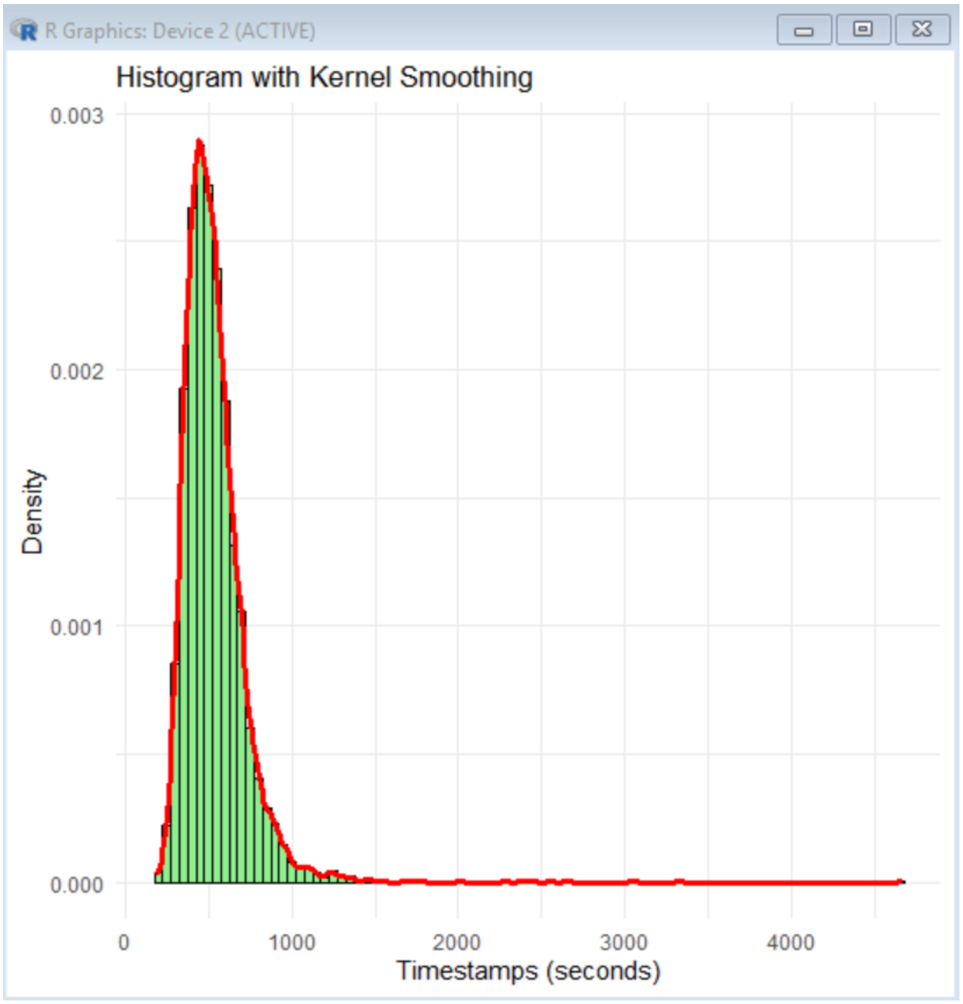
```
> setwd("C:/Users/silen/OneDrive/Desktop/AI-coupledsolver")
> getwd()
[1] "C:/Users/silen/OneDrive/Desktop/AI-coupledsolver"
> data2 <- read.table("N100-Congestion2-timeout1000000.txt", header = TRUE, sep$
> str(data2)
'data.frame':   7869 obs. of  5 variables:
 $ t    : int  552 363 423 534 523 345 403 573 394 786 ...
 $ nodes: int  100 100 100 100 100 100 100 100 100 100 ...
 $ size : int  100 100 100 100 100 100 100 100 100 100 ...
 $ trial: int  1 2 3 6 7 8 9 10 11 12 ...
 $ index: int  1 1 1 1 1 1 1 1 1 1 ...
> # Load ggplot2 package
> library(ggplot2)
```

## 1 Plotted Histogram

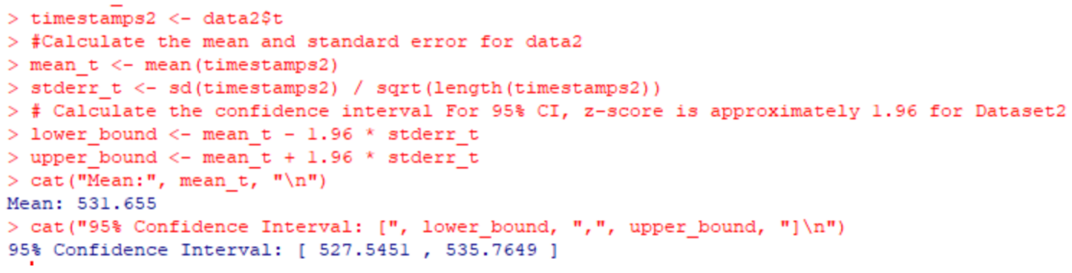
For fit applied Kernal Smoothing as done for previous data set



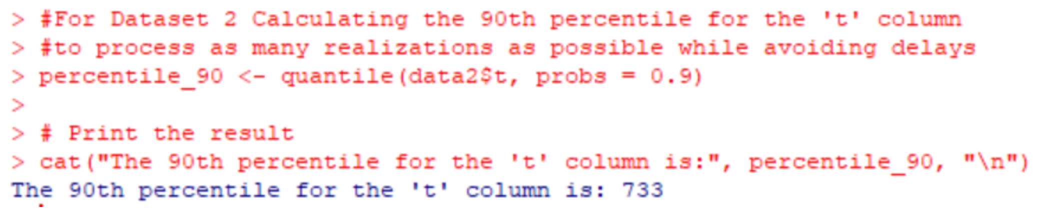
```
> # Add density (kernel smoothing) to the histogram for Dataset 2
> ggplot(data2, aes(x = t)) +
+   geom_histogram(aes(y = after_stat(density)), binwidth = 50, fill = "light green", color = "black") +
+   geom_density(color = "red", lwd = 1.4) +
+   labs(title = "Histogram with Kernel Smoothing",
+        x = "Timestamps (seconds)",
+        y = "Density") +
+   theme_minimal()
>
```



Performed calculations for Mean and 95% confidence interval for Dataset 2
Mean: 531.655
Confidence Interval: [ 527.5451 , 535.7649 ]



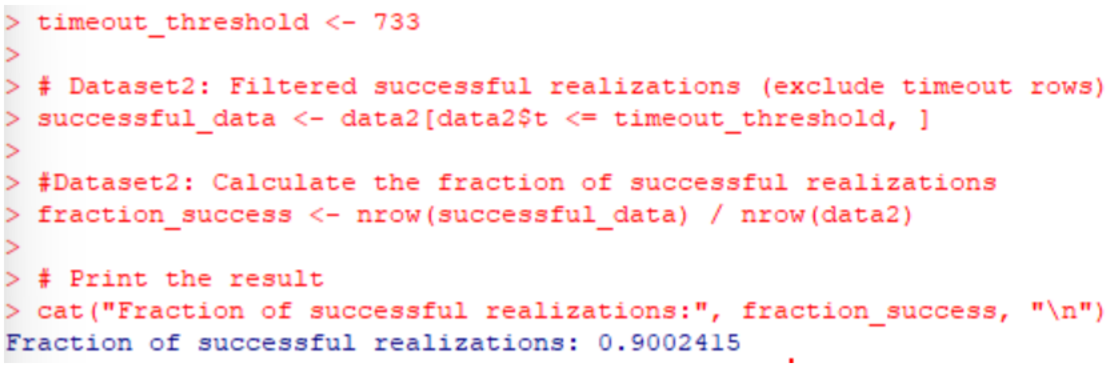
```
> timestamps2 <- data2$t
> #Calculate the mean and standard error for data2
> mean_t <- mean(timestamps2)
> stderr_t <- sd(timestamps2) / sqrt(length(timestamps2))
> # Calculate the confidence interval For 95% CI, z-score is approximately 1.96 for Dataset2
> lower_bound <- mean_t - 1.96 * stderr_t
> upper_bound <- mean_t + 1.96 * stderr_t
> cat("Mean:", mean_t, "\n")
Mean: 531.655
> cat("95% Confidence Interval: [", lower_bound, ",", upper_bound, "]\n")
95% Confidence Interval: [ 527.5451 , 535.7649 ]
```



```
> #For Dataset 2 Calculating the 90th percentile for the 't' column
> #to process as many realizations as possible while avoiding delays
> percentile_90 <- quantile(data2$t, probs = 0.9)
>
> # Print the result
> cat("The 90th percentile for the 't' column is:", percentile_90, "\n")
The 90th percentile for the 't' column is: 733
```

To consider the following for Dataset 2:
Timeout - is the total number of seconds the system waits for a solution. If no solution is found until this instant (= 1000000), a line is skipped in the dataset, and the system moves to the next trial or index.

1. Calculated the 90th percentile for the 't' column, to process as many realizations as possible while avoiding delays.
2. Identified Timeout Rows: Rows where the t column value exceeds the timeout threshold should be considered as unsuccessful realizations.
3. Excluded Timeout Rows: Filter out those rows when calculating the fraction of realizations that achieve a solution.
4. Calculated the Fraction**:** From the filtered dataset, calculate the fraction of successful realizations.

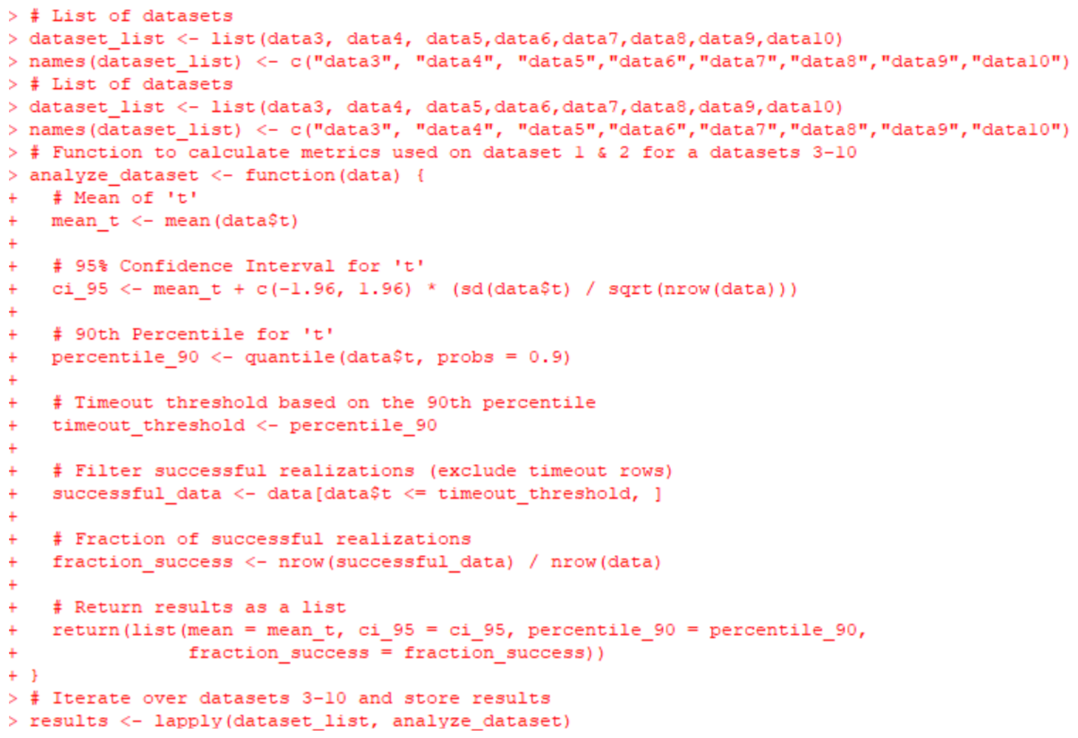


```
> timeout_threshold <- 733
>
> # Dataset2: Filtered successful realizations (exclude timeout rows)
> successful_data <- data2[data2$t <= timeout_threshold, ]
>
> #Dataset2: Calculate the fraction of successful realizations
> fraction_success <- nrow(successful_data) / nrow(data2)
>
> # Print the result
> cat("Fraction of successful realizations:", fraction_success, "\n")
Fraction of successful realizations: 0.9002415
```

This means that approx.. 90% if the realizations from dataset data2 (Data2 – N100 Congestion 2) were successful, they met the defined criteria for achieving the solution. The remaining approx.. 10% either timed out or failed to produce a solution. The majority of the processes captured in data 1 performed effectively within the defined threshold of the 90$^{th}$ percentile. The high fraction reflects strong reliability in my model.

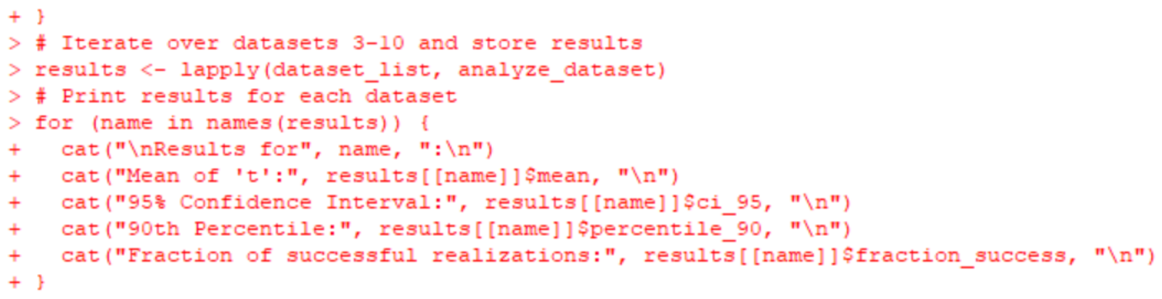
## 2 Loaded Datasets 3 to 10

Created a Loop to calculate metrics used on dataset 1 & 2 for a datasets 3-10

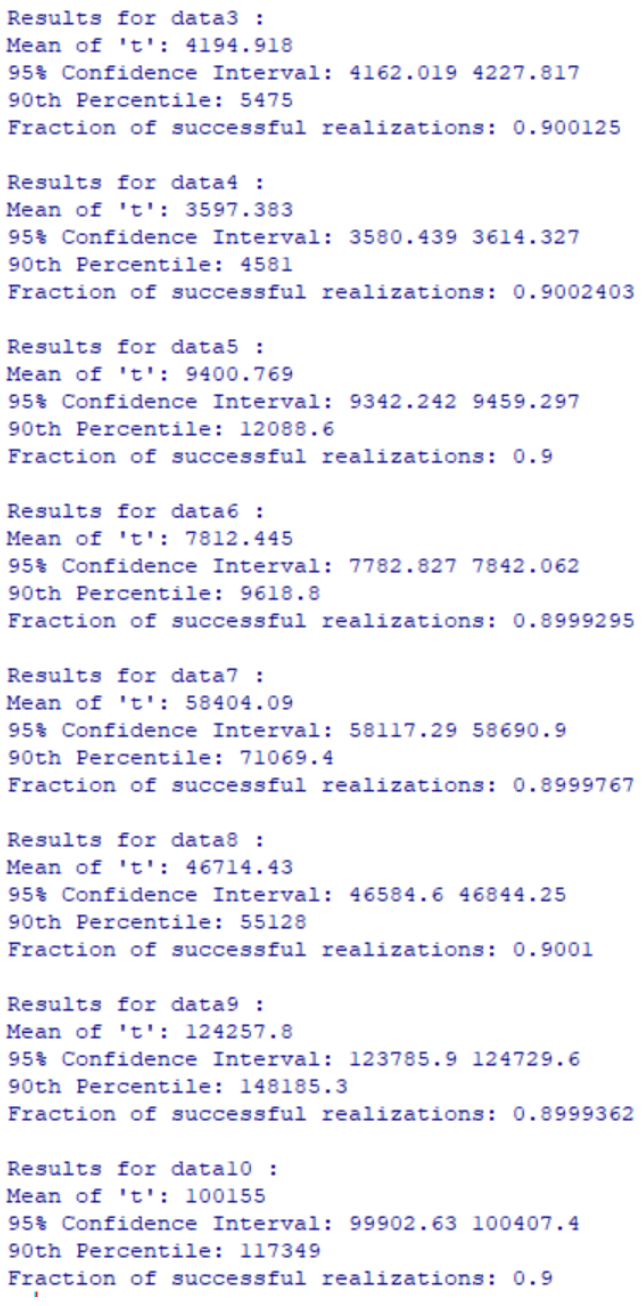


```
> # List of datasets
> dataset_list <- list(data3, data4, data5,data6,data7,data8,data9,data10)
> names(dataset_list) <- c("data3", "data4", "data5","data6","data7","data8","data9","data10")
> # List of datasets
> dataset_list <- list(data3, data4, data5,data6,data7,data8,data9,data10)
> names(dataset_list) <- c("data3", "data4", "data5","data6","data7","data8","data9","data10")
> # Function to calculate metrics used on dataset 1 & 2 for a datasets 3-10
> analyze_dataset <- function(data) {
+   # Mean of 't'
+   mean_t <- mean(data$t)
+
+   # 95% Confidence Interval for 't'
+   ci_95 <- mean_t + c(-1.96, 1.96) * (sd(data$t) / sqrt(nrow(data)))
+
+   # 90th Percentile for 't'
+   percentile_90 <- quantile(data$t, probs = 0.9)
+
+   # Timeout threshold based on the 90th percentile
+   timeout_threshold <- percentile_90
+
+   # Filter successful realizations (exclude timeout rows)
+   successful_data <- data[data$t <= timeout_threshold, ]
+
+   # Fraction of successful realizations
+   fraction_success <- nrow(successful_data) / nrow(data)
+
+   # Return results as a list
+   return(list(mean = mean_t, ci_95 = ci_95, percentile_90 = percentile_90,
+               fraction_success = fraction_success))
+ }
> # Iterate over datasets 3-10 and store results
> results <- lapply(dataset_list, analyze_dataset)
```

Printing Results for Datasets 3 -10

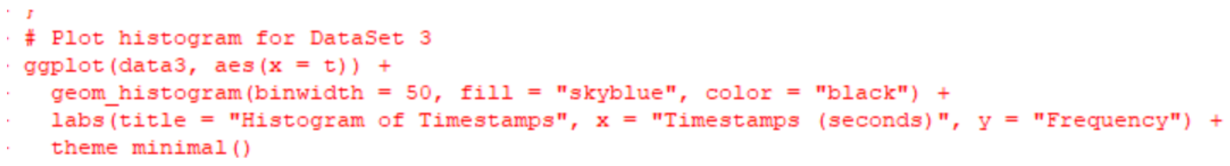


```
+ }
> # Iterate over datasets 3-10 and store results
> results <- lapply(dataset_list, analyze_dataset)
> # Print results for each dataset
> for (name in names(results)) {
+   cat("\nResults for", name, ":\n")
+   cat("Mean of 't':", results[[name]]$mean, "\n")
+   cat("95% Confidence Interval:", results[[name]]$ci_95, "\n")
+   cat("90th Percentile:", results[[name]]$percentile_90, "\n")
+   cat("Fraction of successful realizations:", results[[name]]$fraction_success, "\n")
+ }
```



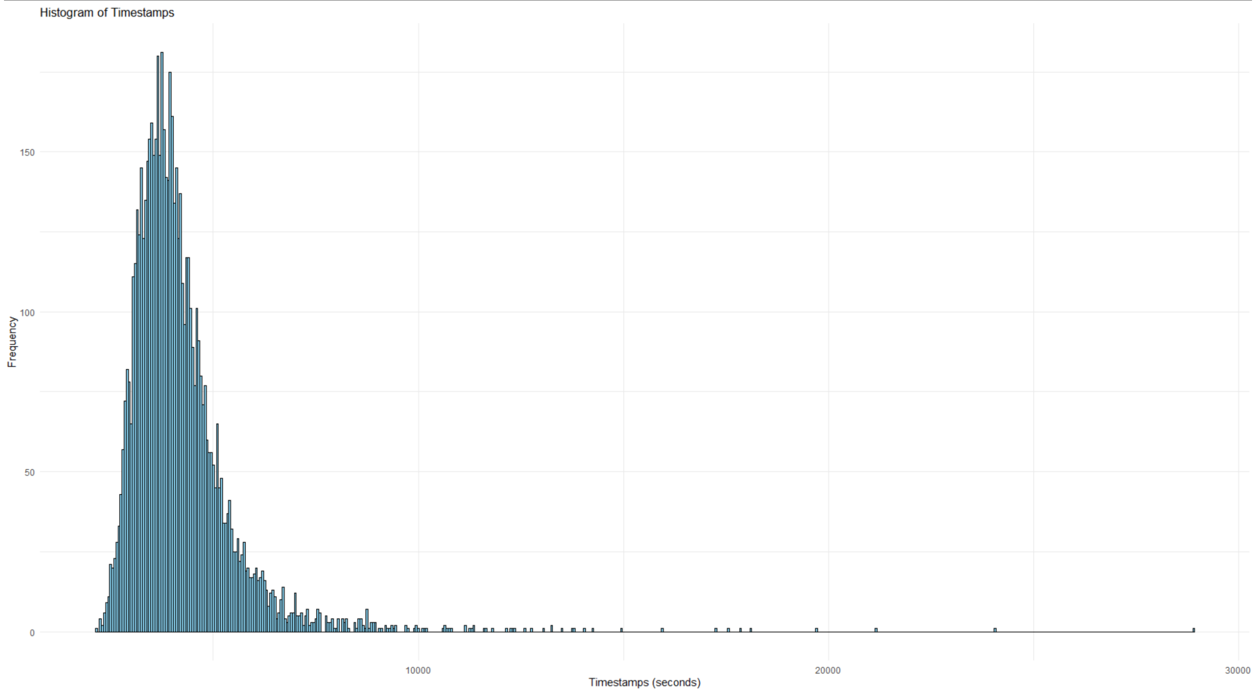
```
Results for data3 :
Mean of 't': 4194.918
95% Confidence Interval: 4162.019 4227.817
90th Percentile: 5475
Fraction of successful realizations: 0.900125

Results for data4 :
Mean of 't': 3597.383
95% Confidence Interval: 3580.439 3614.327
90th Percentile: 4581
Fraction of successful realizations: 0.9002403

Results for data5 :
Mean of 't': 9400.769
95% Confidence Interval: 9342.242 9459.297
90th Percentile: 12088.6
Fraction of successful realizations: 0.9

Results for data6 :
Mean of 't': 7812.445
95% Confidence Interval: 7782.827 7842.062
90th Percentile: 9618.8
Fraction of successful realizations: 0.8999295

Results for data7 :
Mean of 't': 58404.09
95% Confidence Interval: 58117.29 58690.9
90th Percentile: 71069.4
Fraction of successful realizations: 0.8999767

Results for data8 :
Mean of 't': 46714.43
95% Confidence Interval: 46584.6 46844.25
90th Percentile: 55128
Fraction of successful realizations: 0.9001

Results for data9 :
Mean of 't': 124257.8
95% Confidence Interval: 123785.9 124729.6
90th Percentile: 148185.3
Fraction of successful realizations: 0.8999362

Results for data10 :
Mean of 't': 100155
95% Confidence Interval: 99902.63 100407.4
90th Percentile: 117349
Fraction of successful realizations: 0.9
```

The below code will be used for all Datasets from 3-10 (each screenshot will not be pasted to save space in the document), could not sure iteration to display all histograms and applied kernel smoothing.

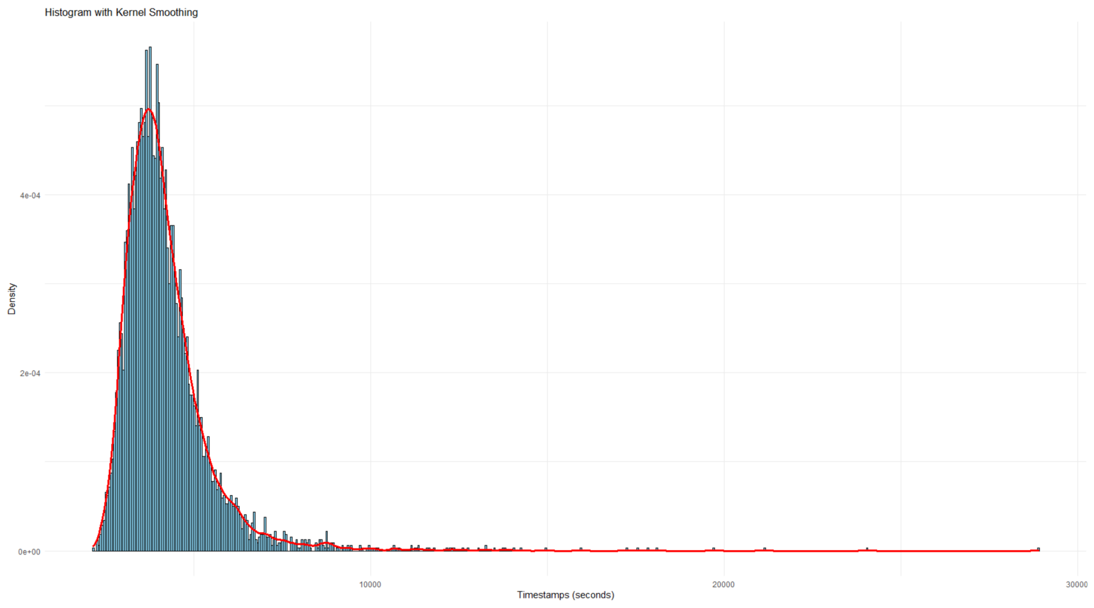


```
;
# Plot histogram for DataSet 3
ggplot(data3, aes(x = t)) +
  geom_histogram(binwidth = 50, fill = "skyblue", color = "black") +
  labs(title = "Histogram of Timestamps", x = "Timestamps (seconds)", y = "Frequency") +
  theme minimal()
  #Add density (kernel smoothing) to the histogram for Dataset 3
ggplot(data3, aes(x = t)) +
  geom_histogram(aes(y = after_stat(density)), binwidth = 50, fill = "skyblue", color = "black") +
  geom_density(color = "red", lwd = 1.4) +
  labs(title = "Histogram with Kernel Smoothing",
       x = "Timestamps (seconds)",
       y = "Density") +
  theme_minimal()
```

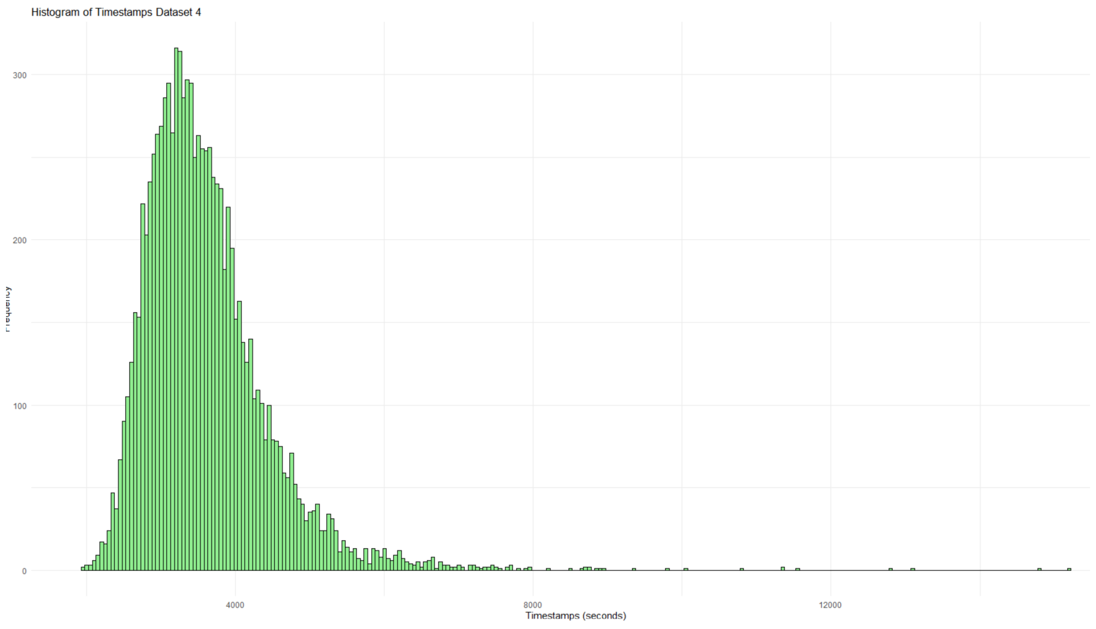
## Dataset 3 Histogram

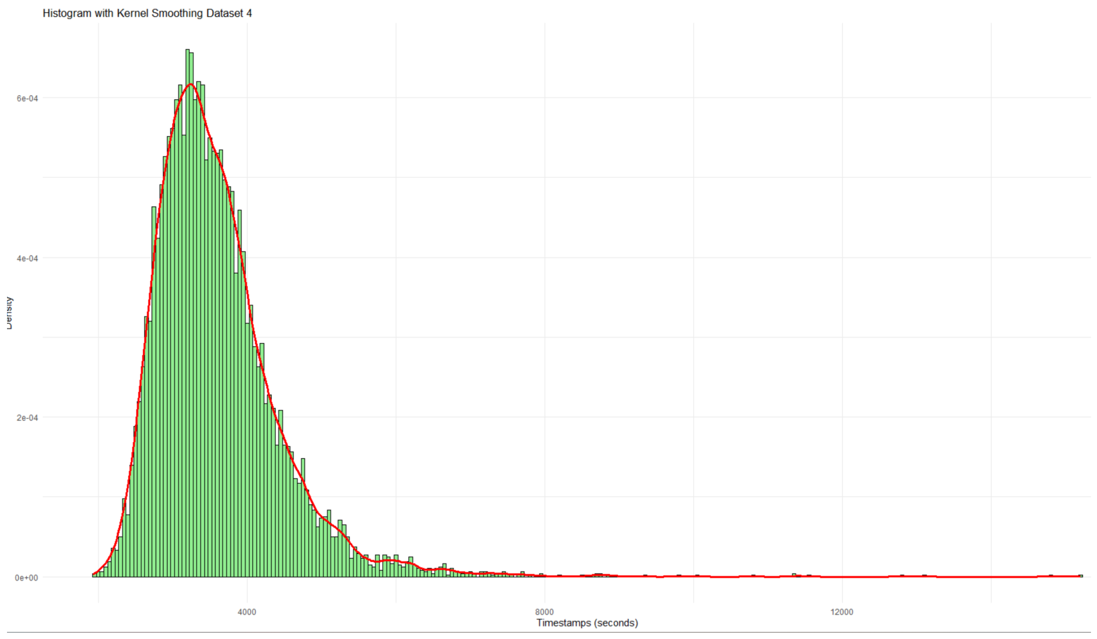


## 3 Dataset 3 with Kernal Smoothing

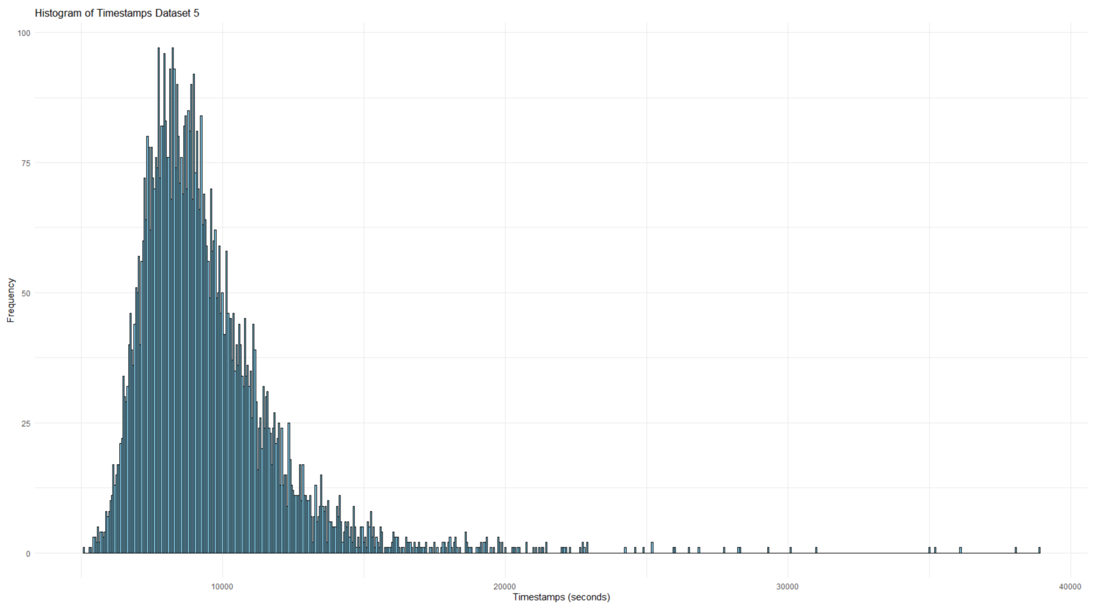


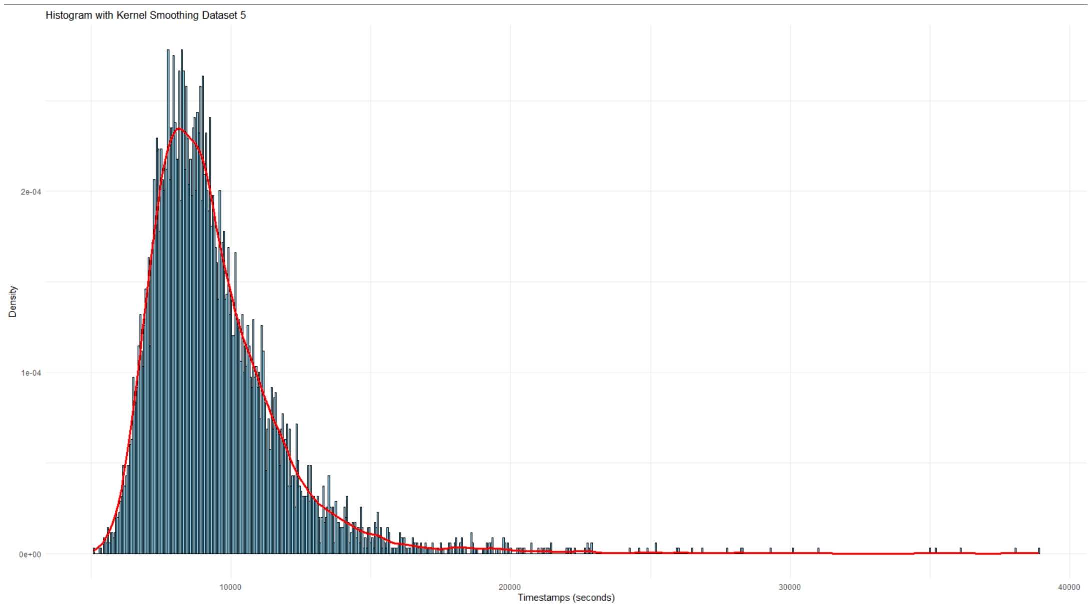
Dataset 3 Findings:
-With a fraction of success realization of 90.01%, we observe majority of realizations complete within the defined timeout threshold of 5475 second (90th percentile)
-The mean 4194.918 seconds, shows that on average, solutions are achieved quickly relative to the timeout threshold (1000000 seconds)
-Kernel Smoothing fit shows:
-A narrow peak suggesting less variability (were most timestamps are similar)
- The kernel density has a long tail toward higher values, this mean that although most realizations completed quickly, there are a few that took significantly longer.





Dataset 4 Findings:
-With a fraction of success realization of 90.02%, we observe majority of realizations complete within the defined timeout threshold of 4581 second (90$^{\text{th}}$ percentile)
-The mean 3597.383 seconds, shows that on average, solutions are achieved quickly relative to the timeout threshold (1000000 seconds)
-Kernel Smoothing fit shows:
-A slightly wider peak suggesting a little more variability than observed in Dataset 3
- The kernel density has a long tail toward higher values, this mean that although most realizations completed quickly, there are a few that took significantly longer

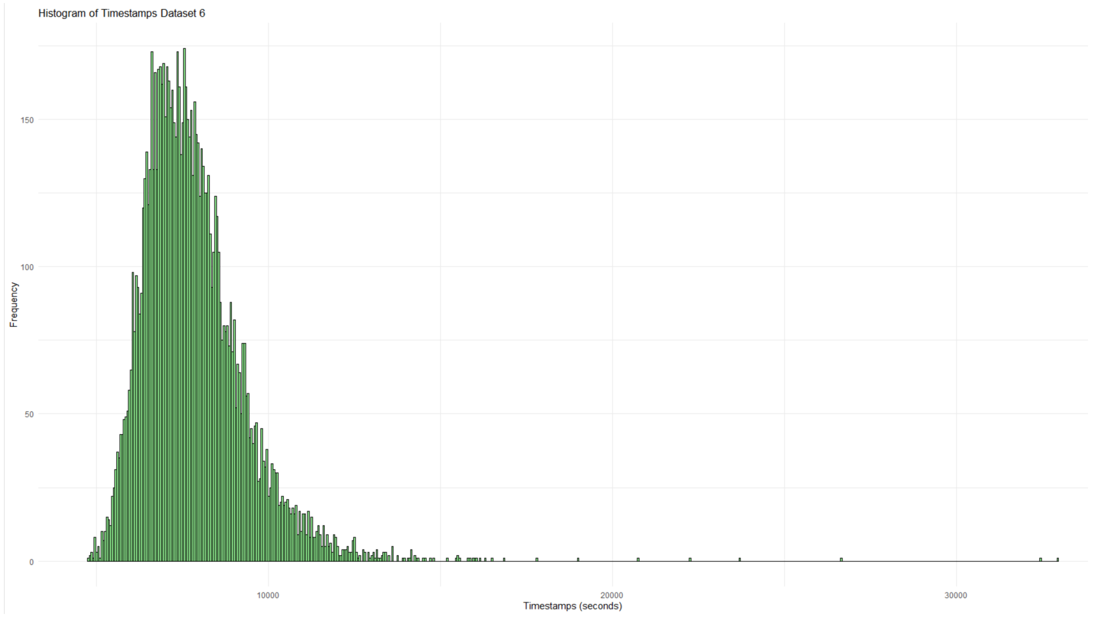


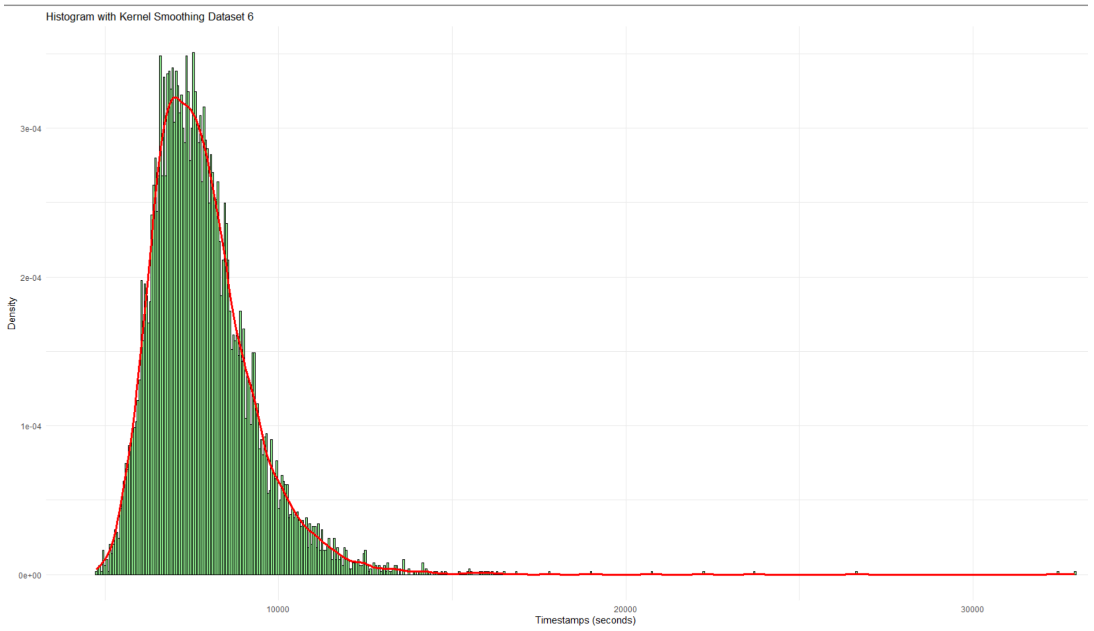


Dataset 5 Findings: -
-With a fraction of success realization of 90%, we observe majority of realizations complete within the defined timeout threshold of 12088.6 second (90th percentile)
-The mean is 9400.769 seconds, and shows that on average, solutions are achieved quickly relative to the timeout threshold (1000000 seconds)
-Kernel Smoothing fit shows:

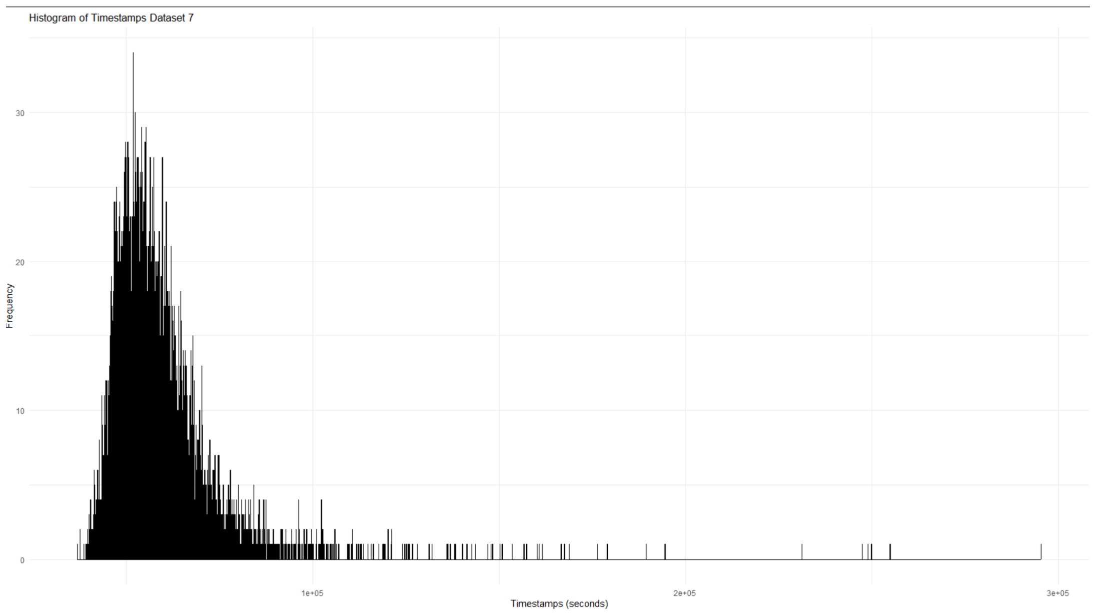
-The curve highlights how the data has been dispersed. Observed a wide curve for high frequency time stamps.

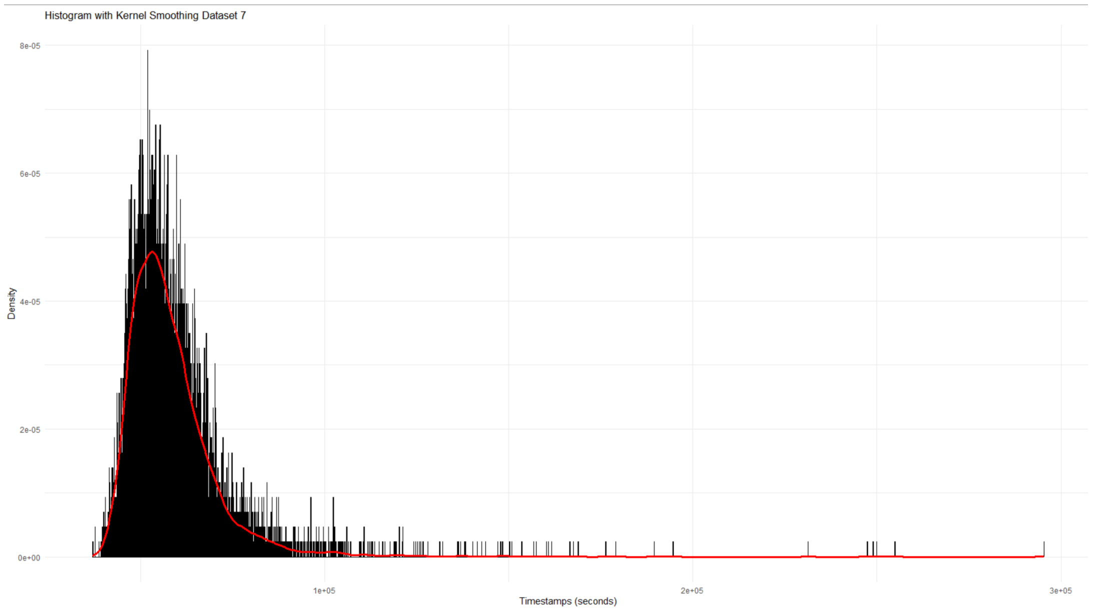
- The kernel density has a long tail toward higher values, this means that although most realizations completed quickly, there are a few that took significantly longer. Notably there are more outliers in this dataset than the previous.





Dataset 6 Findings:
-With a fraction of success realization of 89.9992%, we observe majority of realizations complete within the defined timeout threshold of 9618.8 second (90th percentile)
-The mean 7812.445 seconds, shows that on average, solutions are achieved quickly relative to the timeout threshold (1000000 seconds)
- Observed 95% Confidence Interval: 7782.827 7842.062 where timestamps fall within this range.
-Kernel Smoothing fit shows:
    -A narrow peak suggesting less variability, most timestamps are similar

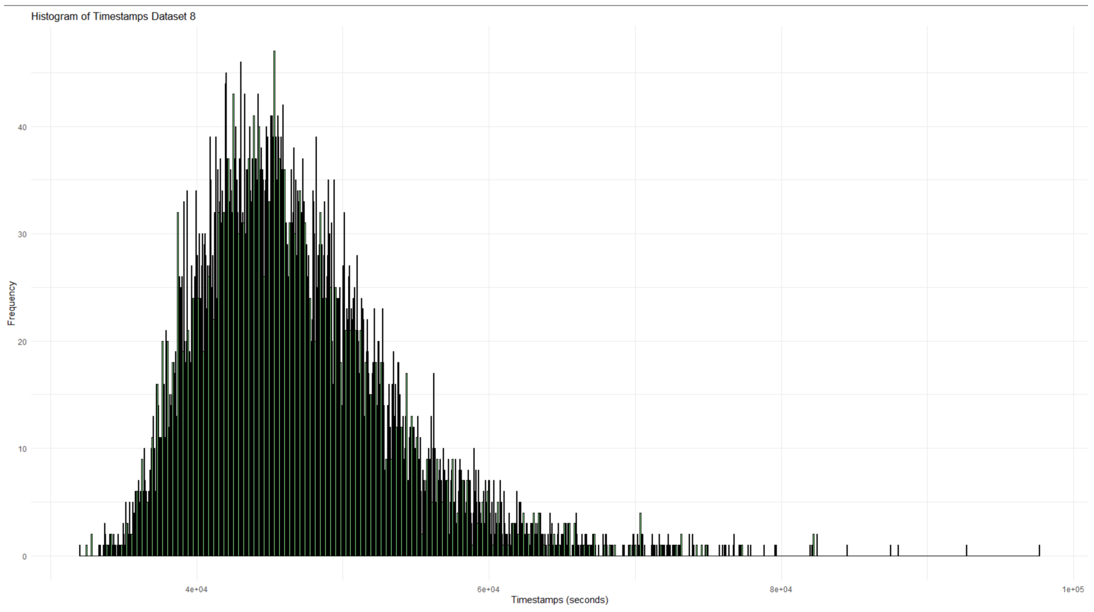


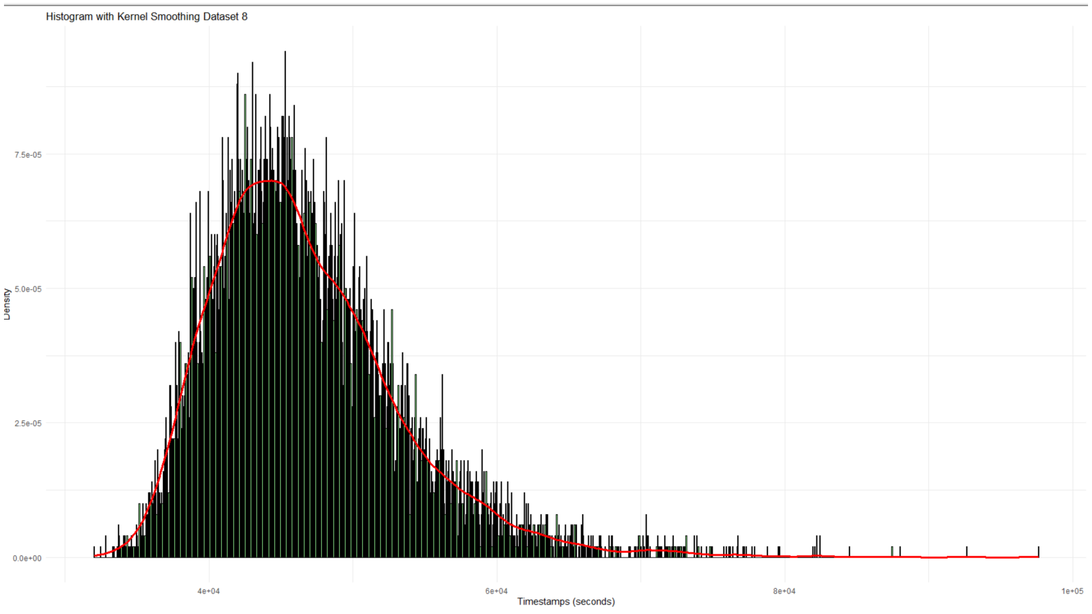


Dataset 7 Findings:
-With a fraction of success realization of 89. 997%, we observe majority of realizations complete within the defined timeout threshold of 71069.4 seconds (90$^{\text{th}}$ percentile)
-The mean is 58404.09 seconds, shows that on average, solutions are achieved quickly relative to the timeout threshold (1000000 seconds)
-Kernel Smoothing fit shows:

-The higher pikes in the histogram bins show values with some ranges of (t) have a greater frequency than the kernel density model assumes

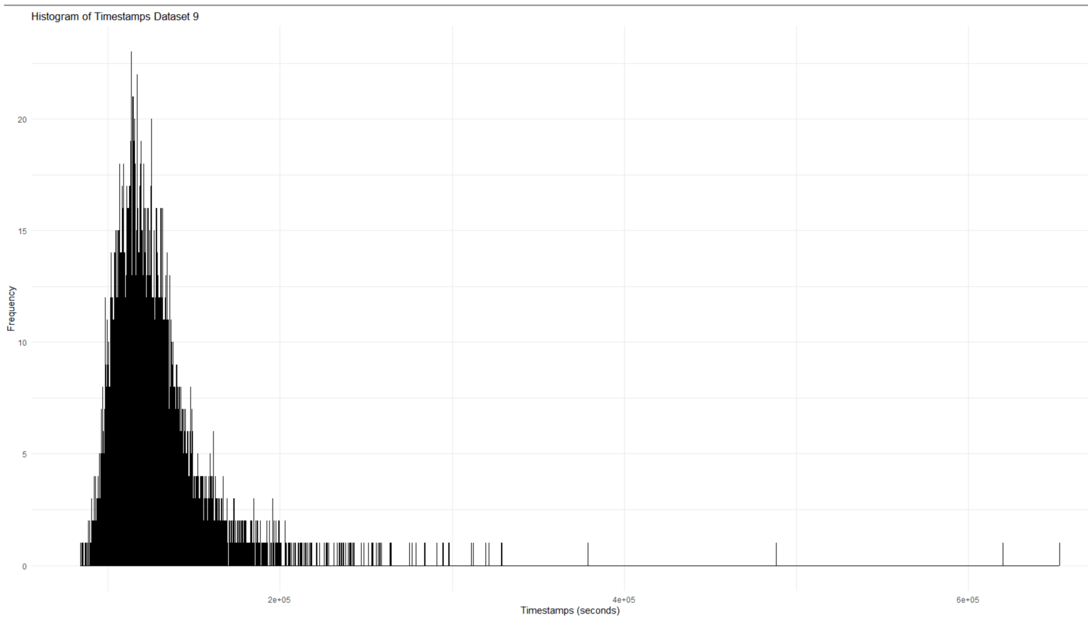
- The high-frequency clusters are shifted towards higher timestamps; it might indicate certain points where the system starts facing delays or struggles in comparison to dataset 1 and 2 for instance.

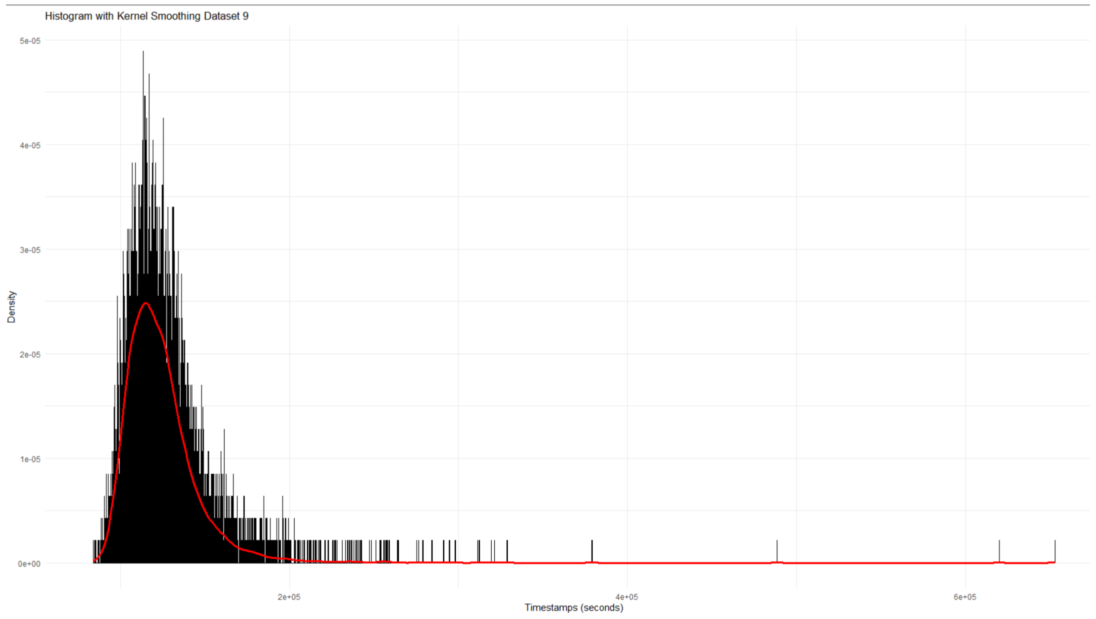




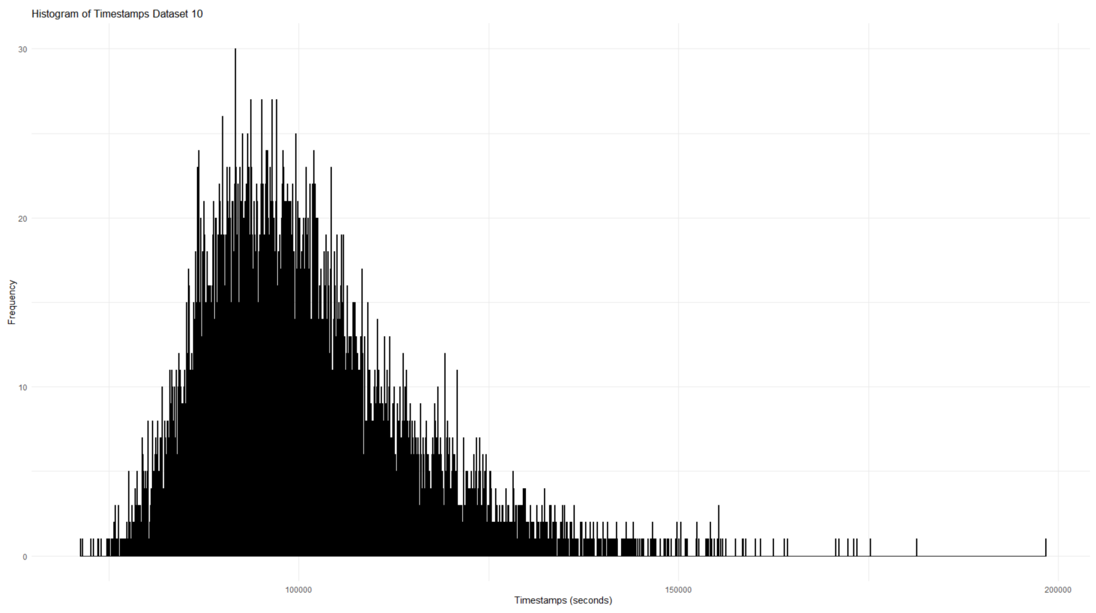
Dataset 8 Findings:
-With a fraction of success realization of 90.01%, we observe majority of realizations complete within the defined timeout threshold of 55128 second (90th percentile)
-We observe a 95% Confidence Interval: 46584.6 - 46844.25
-The mean is 46714.43 seconds, shows that on average, solutions are achieved quickly relative to the timeout threshold (1000000 seconds)

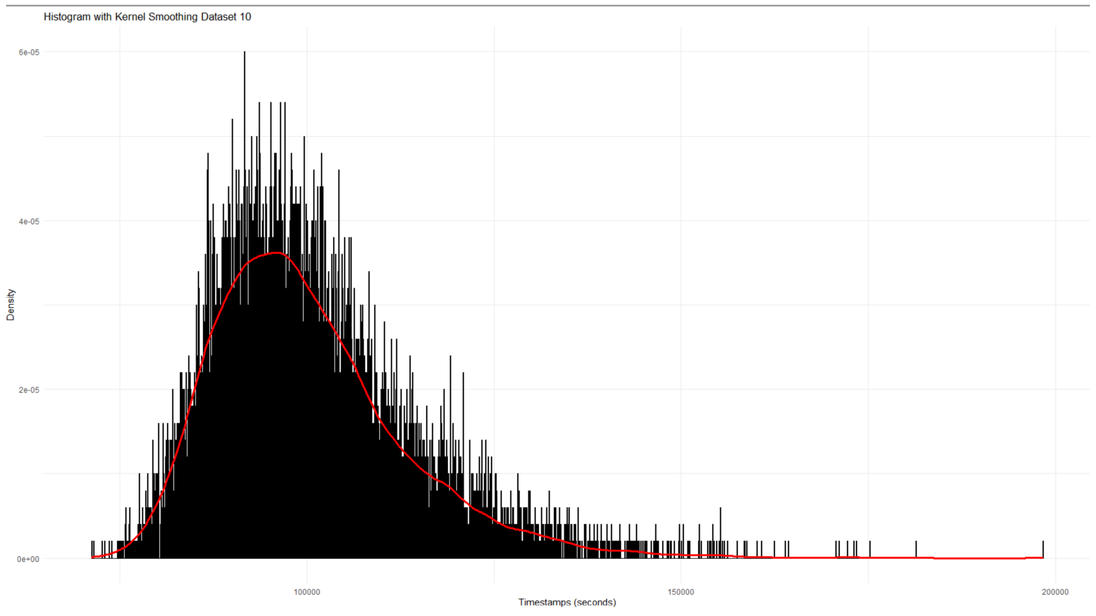
-Kernel Smoothing fit shows:
-Many realizations appear to have been successfully resolved when we observe the peak curve. A center concentration where the peak of the curve shows more timestamps concentrated around approx.. 40,0000 seconds





Dataset 9 Findings:
-With a fraction of success realization of 89. 993%, we observe majority of realizations complete within the defined timeout threshold of 148185.3 seconds (90th percentile)
-The mean 124257.8 seconds, shows that on average, solutions are achieved quickly relative to the timeout threshold (1000000 seconds)
-Kernel Smoothing fit shows:
-For items within the Kernel Density Curve, we observe high frequency areas. These areas are likely to have been timestamps where the system processed solutions consistently
- For items outside the Kenel Density Curve, we can assume optimal completion time. Optimal completion times meaning, where the system achieved solutions at an accelerated pace within 95% confidence interval.

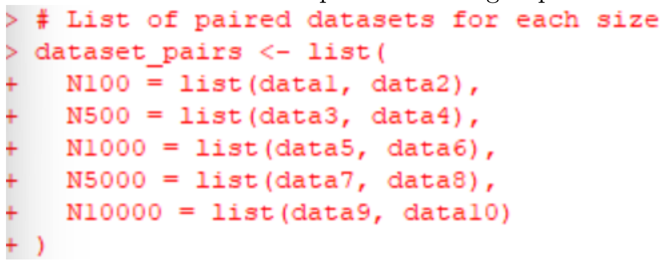




Dataset 10 Findings:
-With a fraction of success realization of 90%, we observe majority of realizations complete within the defined timeout threshold of 117349 second (90$^{\text{th}}$ percentile)
-The mean 100155 seconds, shows that on average, solutions are achieved quickly relative to the timeout threshold (1000000 seconds)
-Observed 95% Confidence Interval: 99902.63 -100407.4
-Kernel Smoothing fit shows:
-The curve peak around 100,000 seconds, suggesting that most timestamps are clustered near this value. This shows relatively consistent system behavior within this range, as observed in 95% Confidence Interval.
- The long right tail of the curve could imply that some realizations took significantly longer to complete. This may hint at outliers or challenging trials.

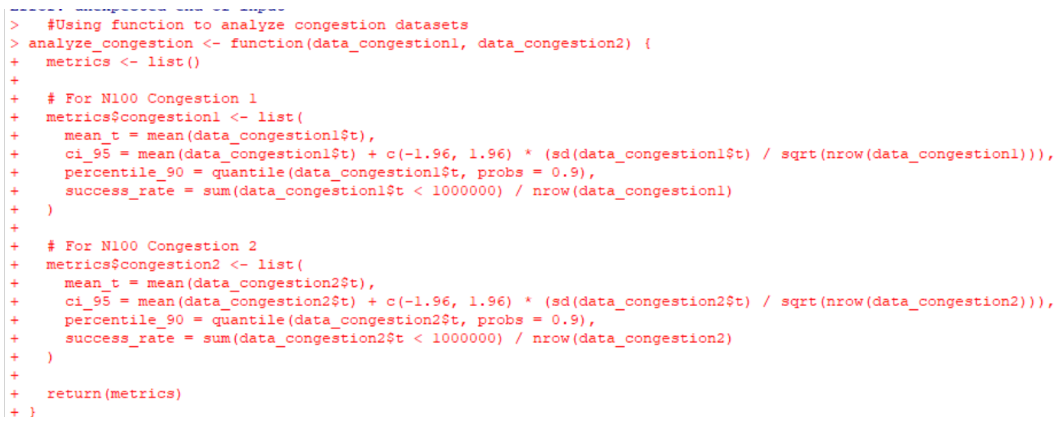
Comparing Congestion 1 or 2 (system status):

Created a list for each sample size set to group.



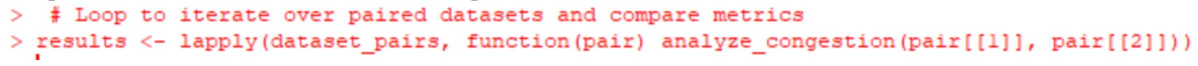
```
> # List of paired datasets for each size
> dataset_pairs <- list(
+   N100 = list(data1, data2),
+   N500 = list(data3, data4),
+   N1000 = list(data5, data6),
+   N5000 = list(data7, data8),
+   N10000 = list(data9, data10)
+ )
```

To attain metrics, created list with below calculations
mean_t: Average time taken.
ci_95: Confidence interval for the mean.
percentile_90: The 90th percentile of t.
success_rate: Fraction of successful realizations (those completing within timeout).

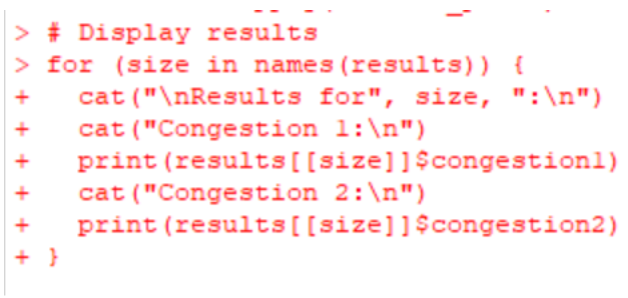


```
>   #Using function to analyze congestion datasets
> analyze_congestion <- function(data_congestion1, data_congestion2) {
+   metrics <- list()
+
+   # For N100 Congestion 1
+   metrics$congestion1 <- list(
+     mean_t = mean(data_congestion1$t),
+     ci_95 = mean(data_congestion1$t) + c(-1.96, 1.96) * (sd(data_congestion1$t) / sqrt(nrow(data_congestion1))),
+     percentile_90 = quantile(data_congestion1$t, probs = 0.9),
+     success_rate = sum(data_congestion1$t < 1000000) / nrow(data_congestion1)
+   )
+
+   # For N100 Congestion 2
+   metrics$congestion2 <- list(
+     mean_t = mean(data_congestion2$t),
+     ci_95 = mean(data_congestion2$t) + c(-1.96, 1.96) * (sd(data_congestion2$t) / sqrt(nrow(data_congestion2))),
+     percentile_90 = quantile(data_congestion2$t, probs = 0.9),
+     success_rate = sum(data_congestion2$t < 1000000) / nrow(data_congestion2)
+   )
+
+   return(metrics)
+ }
```

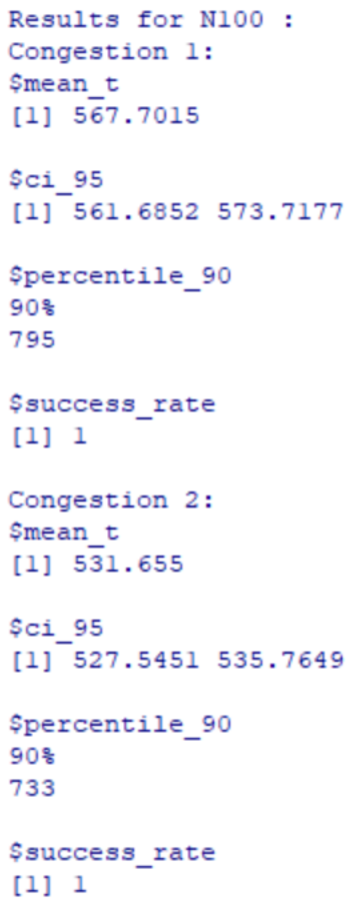
Loop to iteration over datasets and compare metrics



```
>  # Loop to iterate over paired datasets and compare metrics
> results <- lapply(dataset_pairs, function(pair) analyze_congestion(pair[[1]], pair[[2]]))
```

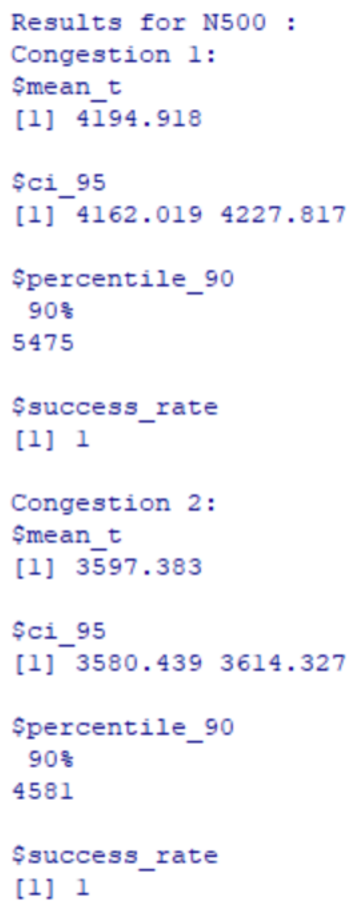


```
> # Display results
> for (size in names(results)) {
+   cat("\nResults for", size, ":\n")
+   cat("Congestion 1:\n")
+   print(results[[size]]$congestion1)
+   cat("Congestion 2:\n")
+   print(results[[size]]$congestion2)
+ }
```



```
Results for N100 :
Congestion 1:
$mean_t
[1] 567.7015

$ci_95
[1] 561.6852 573.7177

$percentile_90
90%
795

$success_rate
[1] 1

Congestion 2:
$mean_t
[1] 531.655

$ci_95
[1] 527.5451 535.7649

$percentile_90
90%
733

$success_rate
[1] 1
```

N100 (Sample Size):
- Congestion 1:
  - Mean time ('t') is 567.7015 seconds, with a 90th percentile of 795 seconds.
  - The system shows slightly higher completion times compared to Congestion 2.
- Congestion 2:
  - Mean time ('t') is 531.655 seconds, with a 90th percentile of 733 seconds.
  - Shows a slightly faster performance compared to Congestion 1.
- Congestion 2, seems to outperform Congestion 1 slightly for small sizes.


```
Results for N500 :
Congestion 1:
$mean_t
[1] 4194.918

$ci_95
[1] 4162.019 4227.817

$percentile_90
 90%
5475

$success_rate
[1] 1

Congestion 2:
$mean_t
[1] 3597.383

$ci_95
[1] 3580.439 3614.327

$percentile_90
 90%
4581

$success_rate
[1] 1
```

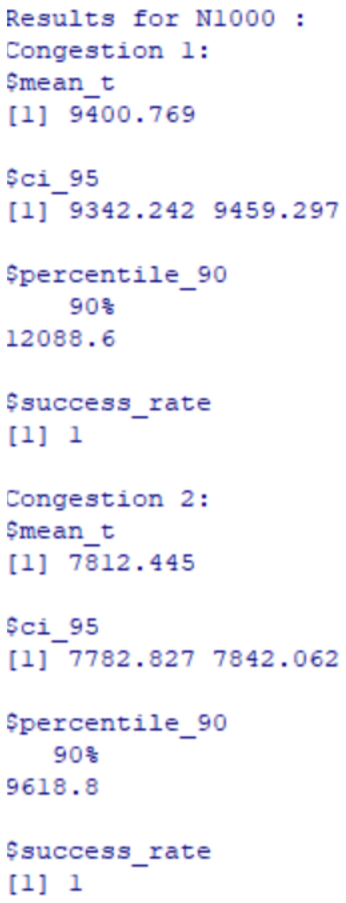
## 4 (Medium Sample Size):

Congestion 1:
- Mean time is 4194.918 seconds, and the 90th percentile is 5475 seconds.

## 5 Congestion 2:

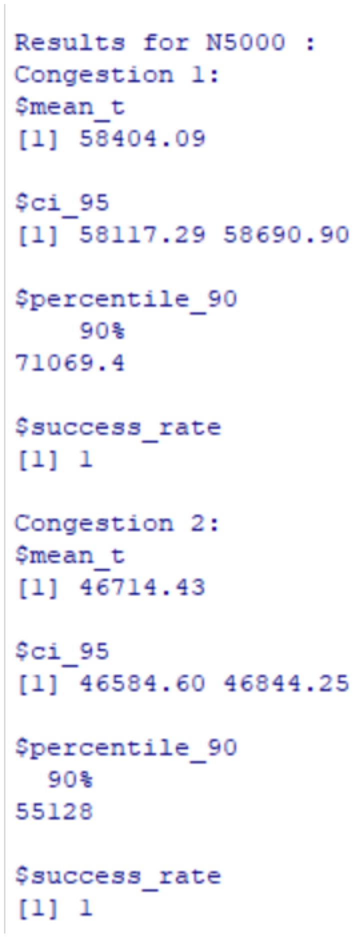
- Mean time is 3597.383 seconds, and the 90th percentile is 4581 seconds.
- Shows fewer delays and faster performance, reflected by the lower mean and 90th percentile.

- Congestion 2 handles medium-sized samples more efficiently than Congestion 1, with lower completion times overall.


```
Results for N1000 :
Congestion 1:
$mean_t
[1] 9400.769

$ci_95
[1] 9342.242 9459.297

$percentile_90
    90%
12088.6

$success_rate
[1] 1

Congestion 2:
$mean_t
[1] 7812.445

$ci_95
[1] 7782.827 7842.062

$percentile_90
   90%
9618.8

$success_rate
[1] 1
```

## 6 (Larger Sample Size):

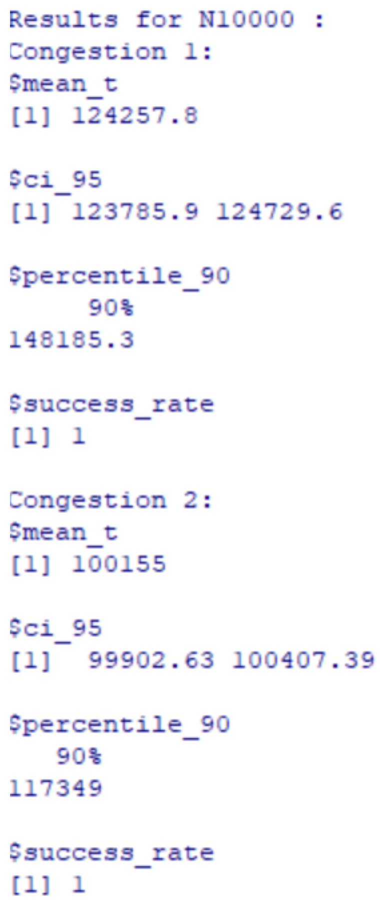
Congestion 1:
  - Mean time is 9400.769 seconds, with a 90th percentile of 12088.6 seconds.
Congestion 2:
  - Mean time is 7812.445 seconds, with a 90th percentile of 9618.8 seconds
  - Consistently better performance compared to Congestion 1.
- Observed Congestion 2 faster processing times, even as sample size increases.



```
Results for N5000 :
Congestion 1:
$mean_t
[1] 58404.09

$ci_95
[1] 58117.29 58690.90

$percentile_90
    90%
71069.4

$success_rate
[1] 1

Congestion 2:
$mean_t
[1] 46714.43

$ci_95
[1] 46584.60 46844.25

$percentile_90
  90%
55128

$success_rate
[1] 1
```

## 7 (Extra Large Sample Size):

Congestion 1:
  - Mean time is 58404.09 seconds, and the 90th percentile is 71069.4 seconds.
- Congestion 2:
  - Mean time is 46714.43 seconds, and the 90th percentile is 55128 seconds.
  - Visible significant improvement in performance compared to Congestion 1.
- As sample size scales up, Congestion 2 maintains a clear advantage.



```
Results for N10000 :
Congestion 1:
$mean_t
[1] 124257.8

$ci_95
[1] 123785.9 124729.6

$percentile_90
     90%
148185.3

$success_rate
[1] 1

Congestion 2:
$mean_t
[1] 100155

$ci_95
[1]  99902.63 100407.39

$percentile_90
   90%
117349

$success_rate
[1] 1
```

## 8 (Extremely Large Sample Size):

-Congestion 1:
  - Mean time is 124257.8 seconds, with a 90th percentile of 148185.3 seconds.
- Congestion 2:
  - Mean time is 100155 seconds, with a 90th percentile of 117349 seconds.
  - Substantial reduction in mean and high-percentile times.
- Although this is the largest sample size, Congestion 2 demonstrates better scalability and lower delays.

Overall Insights (comparing to real-life traffic management systems):

1. Sample Size – Scaling Complexity
Congestion 1:
  - As sample size increases (e.g., N100 → N5000 → N10000), timestamps ('t') show a significant increase in mean and 90th percentile values, thus longer delays and inefficiency under greater network demand. This is analogous to traffic congestion worsening during rush hour in densely populated cities, where bottlenecks form due to limited resources (e.g., intersections or road capacity).
Congestion 2:
  - Although timestamps also increase with sample size, the growth is much slower compared to Congestion 1. This implies that Congestion 2 performs more efficiently under higher complexity. Like an optimized

traffic network, where strategies like dynamic traffic lights or dedicated lanes reduce delays.
2. Overall Performance Across Sample Sizes (N100 to N10000):
- N100: Both Congestion 1 and 2 perform well, with similar mean timestamps and 90th percentiles. This reflects low traffic volumes, where either congestion strategy effectively handles the load.
- For Larger Sizes (N500…N5000)
 - Congestion 2 consistently outperforms Congestion 1 with shorter average timestamps and a lower 90th percentile, even as system size grows. Suggesting that Congestion 2 scales better, much like a well-planned traffic system that accommodates growing urban demands.
- N10000 (Largest System):
 - The disparity becomes more apparent. Congestion 1 shows substantially higher delays (mean is 124,257 seconds, 90th percentile 148,185 seconds), while Congestion 2 remains more efficient (mean time 100,155 seconds, 90th percentile 117,349 seconds). Like a traffic gridlock, Congestion 1 struggles with scalability, picture scenario in overcrowded urban centers. On the other hand, Congestion 2 sustains smoother traffic flow.

## 8.1 Insights on Scalability:

- In Congestion 1, the struggles with bottlenecks as sample size increases, leading to disproportionate increases in delays. Suggests inefficiency in resource allocation or handling of high demand.
- In Congestion 2 adapts more effectively, maintaining consistent performance patterns. The real-life strategies like traffic signal optimization ,variable tolling, or rerouting under congestion are comparable solutions that might suggest Congestion 2's better performance.

Final note:
Congestion 2 has a clear advantage in managing increasing system complexity, showcasing its ability to efficiently allocate resources and minimizing delays across all scales. This behavior mirrors modern intelligent traffic management systems, which adapt dynamically to congestion, ensuring smoother flow even under everyday high demand.
On the other hand, Congestion 1 mirrors outdated or rigid strategies, which struggle to cope as system size grows, leading to inefficiencies such as urban traffic gridlock. These findings highlight the importance of scalable, adaptive solutions to maintain performance in complex and demanding environments.