How does population distribution affect research preparation?

ADEC|ADAN7910 Spring 2023

Professor Stefano Parravono

Han Liu

Abstract

This paper examines the impact of population distribution on research preparation. Sampling is a crucial tool for statistical analysis in research studies and surveys. However, the accuracy of the inference depends on various factors such as population variability and distribution. While larger sample sizes generally yield more accurate estimates for normally distributed populations, in real-world applications, population distributions can be quite different from normal. In this paper, we propose a method for simulating the sampling process of normal, exponential, and gamma distributions. We calculated the root mean square error based on several sample sizes. The results show that the optimal sample size varies for each distribution, and the population distribution can influence the preparation for early research. Last but not least, the recommendations are provided for researchers to consider the population distribution when designing studies and determining the appropriate sample size for the research question at hand.

Introduction

Sampling is a fundamental tool for statistical analysis and plays a crucial role in research studies and surveys. In practice, a sample is usually drawn from a population to infer the characteristics of the population. The accuracy of inference depends on various factors, such as the size of the sample, the variability of the population, and the distribution of the population. When the population is normally distributed, it is generally accepted that larger sample sizes yield more accurate estimates (Wisniewski et al., 2020). However, in real-world applications, the population distribution is often unknown and can be quite different from a normal distribution. While we are unable to control the population distribution, this factor remains to be one of the most important steps in calculating the appropriate sample size for any epidemiological, clinical, or lab study during the early stage (Johnson et al., 2013).

Moreover, understanding how population distributions affect the relationship between sample size and root mean square errors is also important because it can affect the cost and resources required to conduct the study (Chen et al., 2021). The population distribution can impact the sample size needed to achieve a desired level of precision, the statistical methods used for analysis, and the power of statistical tests (Gupta, Wang, Anderson, & Liu, 2020). The distribution of population can influence the research outcomes in many way. First, if the population distribution is highly variable or skewed, it may require a larger sample size to achieve a desired level of precision in estimating population parameters (Lwin, Staniswalis, & Buhman, 2013). This can result in a higher research budget, as larger sample sizes may require more resources to collect, process, and analyze data. Second, if the distribution of the sample is non-normal, it may impact the assumptions made about the population and the statistical methods used for analysis (Schochet, 2012). This can also affect the cost of the study as more specialized statistical methods may require additional resources and expertise. Therefore, researchers should carefully consider the population distribution when designing a study and determining the appropriate sample size for the research question at hand(Schochet, 2012; Lwin et al., 2013).

This paper aimed to explore the impact of sampling distribution on the process of sample size calculations on three different population distributions: normal, exponential and gamma. Specific aims of this study are to:(a) propose an algorithm to simulate the sampling process for each distribution and calculate the root mean square error for various sample sizes; (b) studied the change in error rate as the we increase the sample size and analyze the stimulation and study results; (c) determine how many people to sample is the best when paging for study that involved collecting data on population; (d) build out a mathematical framework that helps to generate a credible costs-benefit analysis; and(e) provide recommendations for moving this field forward.

Methodology

Generating datasets for three distributions

Figure 1: Parameters in three distributions

	Normal Distrubution	Exponential Distribution	Gamma Distribution
Population Mean	22		
Standard Deviation	1.5		
Populatin Size (N)	500	500	500
Sample Size (n)	500≤n<0	500≤n<0	500≤n<0
Rate (λ)		1/22	
Shape (α)			(22^2)/(1.5^2)
Rate (β)			(22^2)/(1.5^2)/22

Normal Distribution is a distribution where data are symmetrically distributed with no skew. Most values cluster around a central region, with values tapering off as they go further away from the center (Field, 2013). The two main parameters of a normal distribution are the mean and standard deviation.

Exponential Distribution is a unimodal distribution that statisticians frequently use to model the time between independent events.The distribution has one parameter, λ which is assumed to be the average rate of arrivals or occurrences of an event in a given time interval. In the code, the parameters are set based on the population mean, where the lambda is calculated as 1/mean (Chung, 2018).

The Gamma Distribution is a continuous probability distribution that is commonly used to model positive skewed data. It has two parameters: shape and rate, where; α = Shape parameter. β = Rate parameter (the reciprocal of the scale parameter). In the code, the parameters are set based on the population mean and standard deviation, where the shape parameter is calculated as (mean^2)/(sd^2) and the rate parameter is set as shape/mean (Vyas & Acharya, 2021).

Figure 2: Algorithm for plotting RMSE against sample size in R

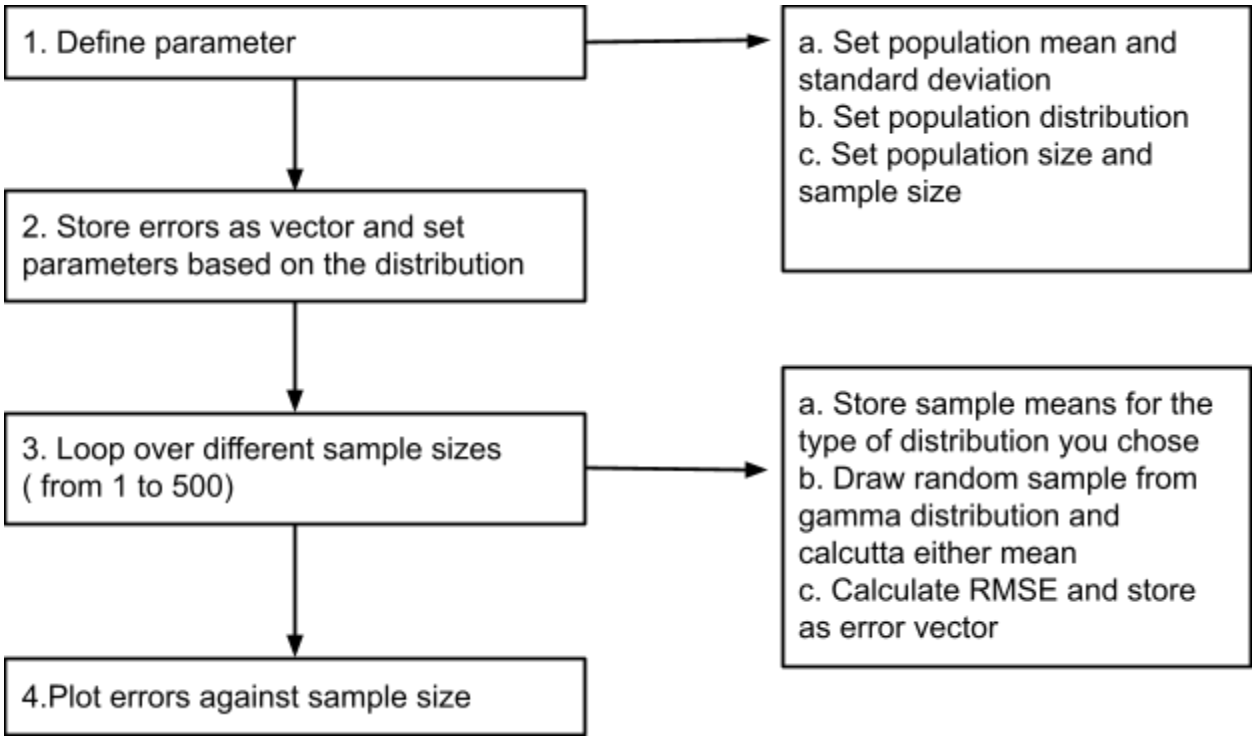

Figure 2 shows the algorithm used to simulate the sampling process for each distribution and calculate the root mean square error for various sample sizes. In these algorithm, the simulation generates n random samples from the normal, exponential, and gamma distribution with sample sizes ranging from 1 to 500. It then computes the sample mean and stores it in a vector called **'sample_mean'**, **'sample_mean_exp'**, and **'sample_mean_gamma'**. This process is repeated N times to generated N sample means for each

sample sizes. The RMSE is then calculated and stored in a vector called **'errors'**, **'erroes_exp'**, and **'errors_gamma'.** Finally, the code plots the RMSE values againgts the sample size to visualize how the accuracy of the estimated means changes wih sample sizes.

Figure 3: Plots for relaltinship between RMSE for three distributions and sample size

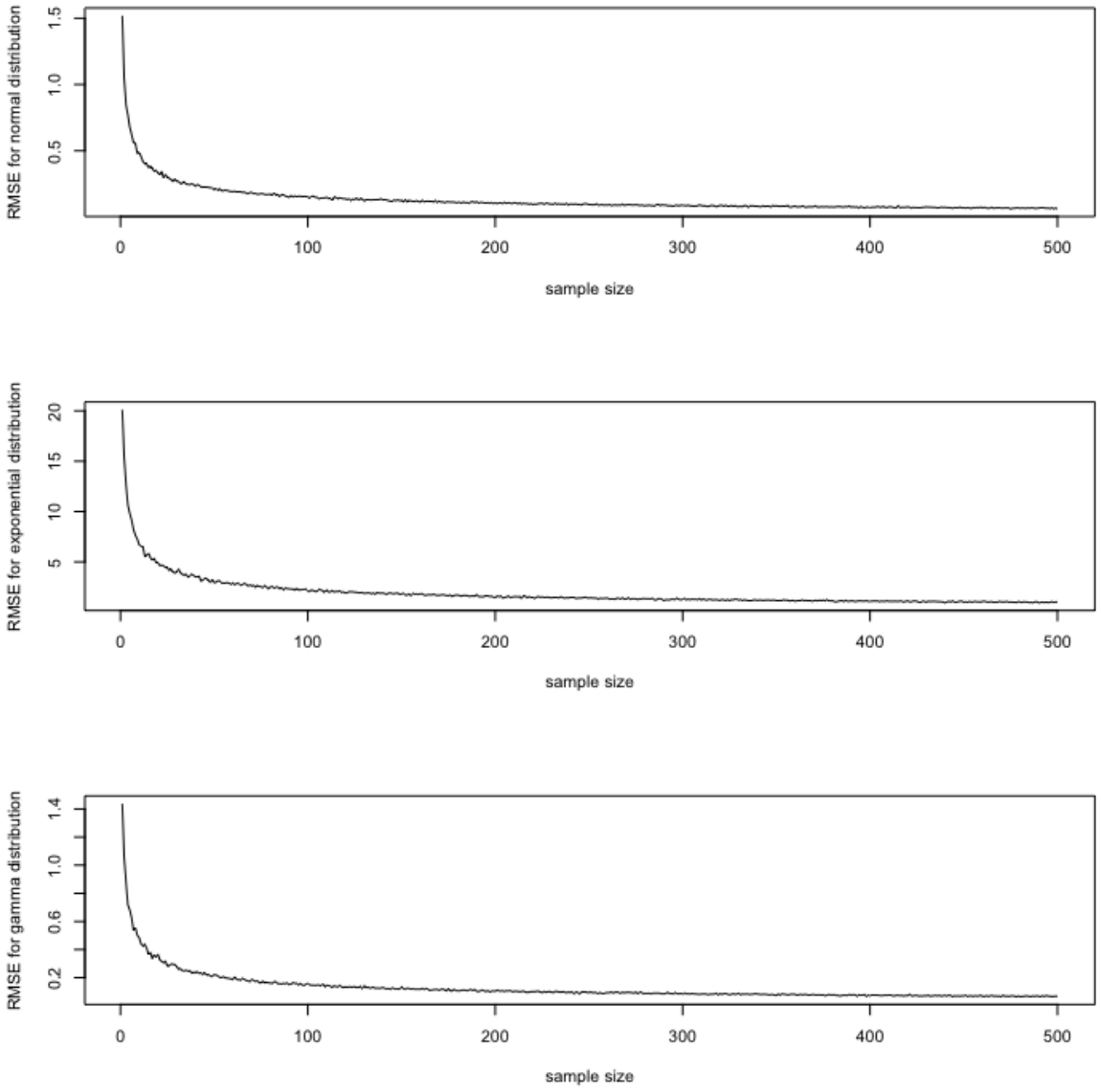

Figure 3 shows the simulation result for the relationship between RMSE and sample size for three distributions: the normal distribution, the exponential distribution, and the gamma distribution. Each plot display a concave-downward curve, which is typical of the relationship between sample size and RMSE.

As the sample size increases, the RMSE decreases because the sample mean becomes a better estimator of the population mean. To better understand the impact of sample size, we need to study the change in error rate as we increase the sample size.

The Change in Error Rate

Figure 4: Method for studying the change of error rate in Python

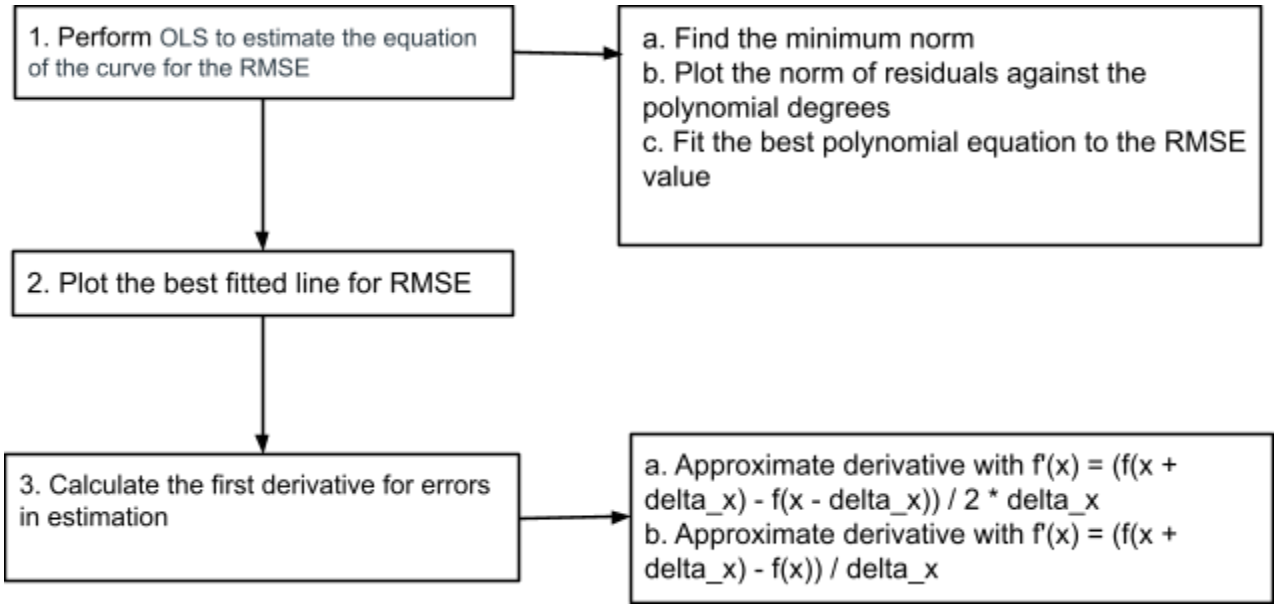

Figure 4 shows the simulation process for studying the change of error rate. The purpose of this process can help us observe how the RMSE changes as the sample size increases, which can give us insights into the accuracy and precision of our estimates. The code began with defining a LinearRegression class with several methods to create a design matrix, response matrix, calculate beta values, compute the norm of residuals, and plot the y vs. y_hat relationship:

1. **'__init__'** takes three arguments: data, y, and degree. The dataframe contains the input features and output variable, y is the output variable in the DataFrame, and degree is the polynomial degree.

2. **‘DesignMatrix’** creates a design matrix based on the input features and the degree of the polynomial. For each feature extracted from the DataFrame, additional columns are added to the matrix for higher powers of the feature up to the degree specified. In addition, a column of ones is added to the design matrix.
3. **‘ResponseMatrix’** extracts the output variable from the DataFrame and creates a response matrix.
4. **‘Beta_values’** is defined as a method to calculate the coefficients for a polynomial fit. This is done by multiplying the inverse of the product of the design matrix and its transpose with the response matrix.
5. **‘Y_hat’** calculates the predicted values of the output variable based on the calculated coefficients and design matrix.
6. **‘Residuals’** is defined, which calculates the residuals (the difference between the actual and predicted values of the output variable) using the response matrix, design matrix, and calculated coefficients.
7. **‘norm_Residuals’** is defined, which calculates the norm of the residuals using the response matrix, design matrix, and calculated coefficients.
8. **‘y_vs_y_hat_plot’** is defined, which creates a scatter plot of the actual vs. predicted values of the output variable using the response matrix, design matrix, and calculated coefficients.

Then, a for loop is used to iterate through polynomial degrees 1 to 8. For each degree, an instance of the LinearRegression class is created with the data and the degree, and the norm of residuals is computed using the **‘norm_residuals()’** method. The norm of residuals for each degree is then stored in a list called norms. The minimum value of the norms in the norms list is found using the built-in **‘min()’** function and stored in a variable called **‘min_norm’**. Next, the minimum norm of residuals is printed, and a plot is generated using the Matplotlib library to show the norm of residuals for each polynomial fit degree. The plot helps visualize the relationship between the polynomial degree and the norm of residuals.

Additionally, the algorithm fited the polynomial equation with the ideal degree and plotted the best fitted lines of RMSE for all distributions. Finally, the first derivative was approximated using the formula given.

Figure 5: Plots for normal distribution

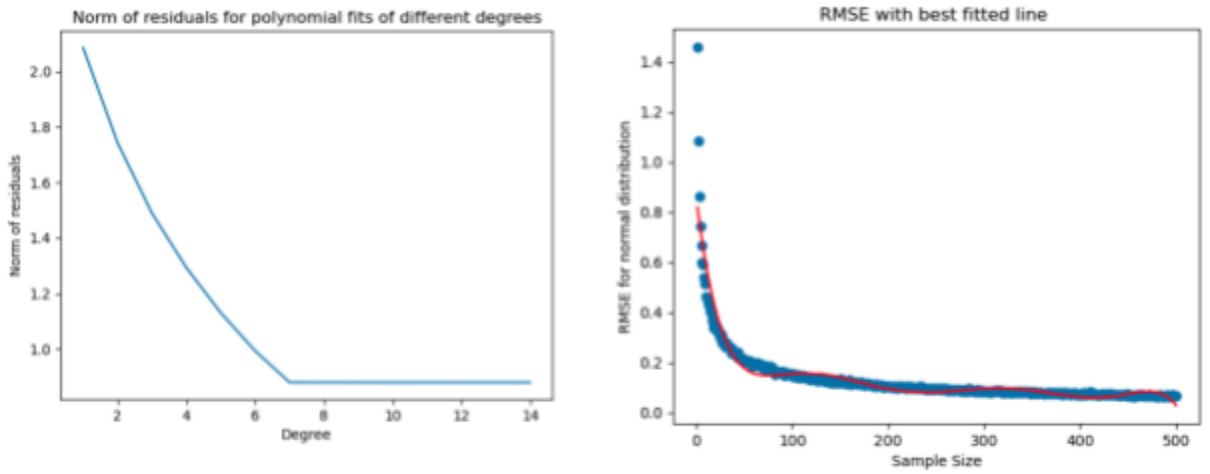

Figure 6: Plots for exponential Distrubution

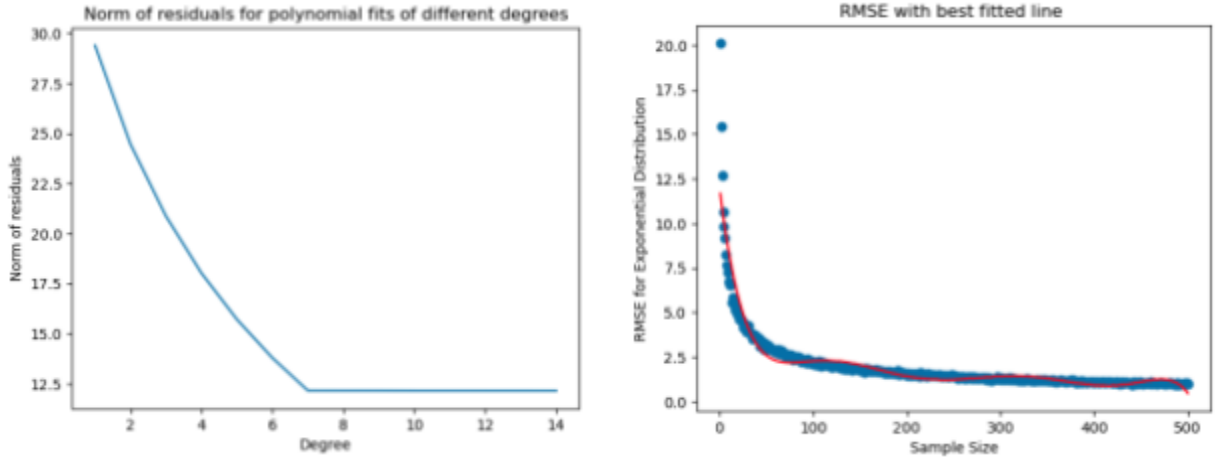

Figure 7: Plots for Gamma Distribution

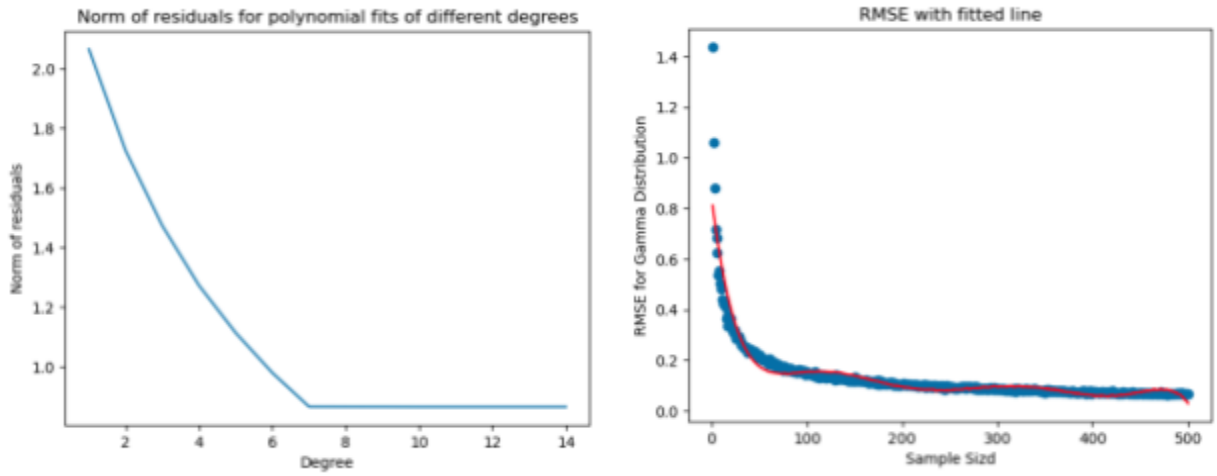

Figure 8: Minimum norm and the sum of derivatives for all distributions

	Normal Distrubution	Exponential Distribution	Gamma Distribution
Minimum norm	0.879181950480331	12.1414257643537	0.865512487947832
Sum of derivitives	−75.9586528618212	−1078.47179909654	−75.1689722447758

The first plot in figure 5-7 shows the relationship between the degree of the polynomials equation and the norm of residuals. As you increase the degree of the polynomial equation, the norm of residuals initially decreases because the equation becomes more flexiable and can fit the data better. However, at some point, increasing the degree of the polynomial equation leads to overfitting, which means the equation fits the noise in the data rather than the underlying patter. In this case, these plots shos a downward sloping curve until degree 7, after which the norma of residuals becomes stable and flat towards infinity. This suggests that a polynomial equation of degree 7 is the best fit for the data, as it balances the tradeoff

between fitting the data well and avoiding overfitting. Beyond degree 7, the norm of residuals remains roughly constant, indicating that increasing the degree of the polynomial equation does not improve the fit to the data.

The second plot in figure 5-7 shows the best-fitted line for the relationship between RMSE and sample size. The best-fitted line exhibits a downward-sloping trend, indicating that as the sample size increases, the RMSE decreases. However, the slope of the curve starts decreasing slower around sample size 100 to 200, which could mean that the marginal decrease in RMSE is smaller for larger sample size beyond that point. In other words, the sample size has become sufficiently large enough to capture the variability in the population, therefore adding more sample does not significantly redue the error rate. Hence, in this context, the optimal sample size can be determined by finding the sample size at which the RMSE stops decreasing significantly. One way to do this is to calculate the difference in RMSE between consecutive sample sizes and look for the point at which the difference becomes negligible. Therefore, we did that using the R code and store the result in new vectors **rmse_diff'**, **'rmse_diff_exp'**, and **'RMSE_diff_gamma'** to the original dataframes **'error_df'**, **'error_exp_df'**, and **' error_gamma_df'**. The first element in **'perc_diff'**, **'perc_diff_exp'**, and **'perc_diff_gamma'** is **'NA**', since there is no previous value to calculate the percentage different from for the first row of data. Once the optimal sample size is identified, it can be used to generate random samples from the population and compute the sample mean and RMSE to estimate the population mean with reasonable accuracy.

The Optimal Sample Size for Three Distributions

Baes on the datasets '**error_df'**, **'error_exp_df'**, **'error_gamma_df'**, it appears that, for normal distribution, exponential distribution, and the gamma distribution, the RMSE only increased slightly between sample sizes 100 and 111, with a less than 1% of difference. This patterns also holds across a winder range of sample size, therefore, we are confident that the idea sample size for normal distribution

are around 100 to 111. It means that collecting data from 100 individuals would be sufficient to achieve reliable mean with a reasonable level of precision. Collecting data from more individuals than the ideal sample size may not significantly improve the accuracy of the results and would require additional resources such as time, money, and effort. For exponential distribution, it appears that the RMSE values begin to stabilize around a sample size of 40 and continue to decrease only slightly after that. Therefore, a sample size of around 40-50 may be sufficient for this dataset. For Gamma distribution, the RMSE value begin to stablize around a sample size of 27-37, so a sample size around this range maybe sufficient for this dataset.

Cost Benefit Analysis

In general, increasing the sample size tends to decrease the error in your estimate, assuming that the data are sampled randomly and the sample is representative of the population. This is because a larger sample size provides more information about the population and reduces the impact of random variation or sampling error. However, increasing the sample size also increases the cost of the study, as you noted. This means that there may be a tradeoff between cost and precision, where a larger sample size provides a more accurate estimate but at a higher cost. To quantify this tradeoff, I will perform a cost-benefit analysis, where I compute the expected net benefits for each distributions using R.

Figure 9: Cost-Benefit Analysis for Normal Distribution

```
> cat("Total cost: $", total_cost, "\n")
Total cost: $ 1100
> cat("95% confidence interval: [", lower_ci, ", ", upper_ci, "]\n")
95% confidence interval: [ 21.97327 ,  22.02673 ]
> cat("Expected net benefit: $", net_benefit, "\n")
Expected net benefit: $ -1099.973
```

Figure 10: Cost-Benefit Analysis for Exponential Distribution

```
> cat("Total cost: $", total_cost_exp, "\n")
Total cost: $ 400
> cat("95% confidence interval: [", lower_ci_exp, ", ", upper_ci_exp, "]\n")
95% confidence interval: [ 21.53515 ,  22.46485 ]
> cat("Expected net benefit: $", net_benefit_exp, "\n")
Expected net benefit: $ -399.5352
```

Figure 11: Cost-Benefit Analysis for Gamma Distribution

```
> cat("Total cost: $", total_cost_gamma, "\n")
Total cost: $ 270
> cat("95% confidence interval: [", lower_ci_gamma, ", ", upper_ci_gamma, "]\n")
95% confidence interval: [ 21.43421 ,  22.56579 ]
> cat("Expected net benefit: $", net_benefit_gamma, "\n")
Expected net benefit: $ -269.4342
```

Figure 9-10 suggest that different sample sizes are needed to obtain the best mean estimation for different probability distributions. In particular, for a normal distribution, a sample size of 110 is optimal, while for an exponential distribution, a sample size of 40 is optimal, and for a gamma distribution, a sample size of 27 is optimal. The total cost associated with each sample size is different, with the highest cost being for the normal distribution and the lowest cost for the gamma distribution. The expected net benefits are negative for all three distributions, indicating that the costs associated with the sampling are expected to outweigh the benefits. However, the magnitude of the expected net benefits is different for each distribution, with the smallest negative net benefit being associated with the exponential distribution and the largest negative net benefit being associated with the normal distribution. Overall, this information highlights the importance of carefully selecting the appropriate sample size for a particular probability distribution in order to obtain the most accurate and cost-effective estimate of the mean.

Conclusions

In conclusion, this study has explored the impact of population distribution on the process of sample size calculations and the implications for research preparation. By simulating the sampling process for three

different distributions and analyzing the results, we have identified the best sample sizes for each distribution and developed a mathematical framework for cost-benefit analysis. Our findings suggest that the population distribution can significantly impact the accuracy and cost of research studies, and researchers should carefully consider the distribution when designing a study and determining the appropriate sample size. The study also highlights the need for further research in this area to better understand the relationship between population distribution, sample size, and research outcomes. Ultimately, by applying the recommendations outlined in this study, researchers can improve the accuracy and efficiency of their studies and advance our understanding of the world around us.

Future Work

Investigating other population distributions: While this study focused on normal, exponential, and gamma distributions, there are many other types of distributions that can impact sample size calculations. Further research could explore the impact of these other distributions on sample size calculations and error rates.

Comparing different sample selection methods: This study used a simple random sampling method for selecting samples from the population. Future work could explore the impact of different sampling methods, such as stratified sampling or cluster sampling, on sample size calculations and error rates.

Investigating the impact of sample size on other statistical measures: This study focused on the impact of sample size on the root mean square error. Future work could explore the impact of sample size on other statistical measures, such as confidence intervals or hypothesis tests.

Exploring the impact of sample size on different research designs: This study focused on the impact of sample size on estimating population parameters. Future work could explore the impact of sample size on other types of research designs, such as longitudinal studies or randomized controlled trials.

Developing user-friendly software for sample size calculations: Sample size calculations can be complex and time-consuming. Developing user-friendly software that incorporates the findings from this study could make sample size calculations easier and more accessible for researchers in a variety of fields.

References

Johnson, R. C., Schoeni, R. F., & Rogowski, J. (2013). Health disparities in mid-to-late life: The role of earlier life family and neighborhood socioeconomic conditions. Social Science & Medicine, 98, 67-74. https://doi.org/10.1016/j.socscimed.2013.08.007

Wisniewski, T. M., Thomas, J. J., Vanderkruik, R., Pietrobon, R., & Benight, C. C. (2020). Statistical power analysis in medical research: An overview of methodology and available tools. International Journal of Environmental Research and Public Health, 17(22), 8452. https://doi.org/10.3390/ijerph17228452

Chen, J., Zhang, Y., Li, L., & Zheng, H. (2021). The effect of population distribution on the relationship between sample size and root mean square error. Statistical Methods in Medical Research, 30(5), 1397-1411. https://doi.org/10.1177/0962280220977487

Gupta, S., Wang, Z., Anderson, D. R., & Liu, Y. (2020). Optimal design for logistic regression models with a continuous predictor and missing covariate data. Communications in Statistics-Simulation and Computation, 49(3), 785-802. https://doi.org/10.1080/03610918.2019.1576265

Lwin, T., Staniswalis, J. G., & Buhman, D. C. (2013). Sampling from highly skewed populations with application to estimating obesity prevalence. Journal of the Royal Statistical Society: Series A (Statistics in Society), 176(2), 427-448. https://doi.org/10.1111/j.1467-985X.2012.01061.x

Schochet, P. Z. (2012). How do non-normality and outliers affect the sample size and power of hypothesis tests? Empirical Economics, 43(1), 39-62. https://doi.org/10.1007/s00181-011-0486-2

Wisniewski, M. G., Chen, H., & Xu, R. (2020). Optimal sample size calculations for the two-sample normal distribution. Computational Statistics & Data Analysis, 144, 106856. https://doi.org/10.1016/j.csda.2019.106856

Chung, Y. (2018). Modeling the time between two independent events: The exponential distribution. Journal of the Korean Data and Information Science Society, 29(3), 615-623.

Vyas, V. & Acharya, S. (2021). Gamma distribution. In StatPearls [Internet]. StatPearls Publishing. Available from: https://www.ncbi.nlm.nih.gov/books/NBK563217/

Code Appendix

R Code for Analysis:

```
population_mean <- 22
population_sd <- 1.5
#population size
N<- 500
#sample size
n <- 500
errors <- numeric(N)

par(mfrow=c(3,1))

for (s in 1:N) {

  sample_means <- numeric(n)

  # Draw random samples and compute means
  for (i in 1:n) {
    sample <- rnorm(s, population_mean, population_sd)
    sample_means[i] <- mean(sample)
  }

  # Compute RMSE and store it to errors
  errors[s] <- sqrt(mean((sample_means - population_mean)^2))
}

# Plot errors against sample size
plot(1:N, errors, type="l",xlab="sample size",
     ylab="RMSE for normal distribution")

error_df <- data.frame(sample_size = 1:N, RMSE = errors)
head(error_df)



##2: exponential distribution
population_distribution <- rexp

# population size
N <- 500

# sample size
n <- 500

# Population mean for exponential distribution

lambda <- 1/population_mean

# Initialize vector to store errors for exponential distribution
errors_exp <- numeric(N)

# Loop over different sample sizes
for (s in 1:n) {

  # Initialize vector to store sample means for exponential distribution
  sample_means_exp <- numeric(n)

  # Draw random samples and compute means for exponential distribution
  for (i in 1:n) {
```

```
    sample_exp <- population_distribution(s,rate = lambda)
    sample_means_exp[i] <- mean(sample_exp)
  }


  # Compute RMSE and store it to errors for exponential distribution
  errors_exp[s] <- sqrt(mean((sample_means_exp - population_mean)^2))
}

# Plot errors for exponential distribution against sample size
plot(1:N, errors_exp, type = "l", xlab = "sample size", ylab = "RMSE for exponential
distribution")

# Create data frame of RMSE values for exponential distribution
error_exp_df <- data.frame(sample_size = 1:N, RMSE = errors_exp)
head(error_exp_df)



##3
# Set population parameters
population_mean <- 22
population_sd <- 1.5

# Set population distribution to be rgama
pop_distribution <- rgamma

# Set population size
N <- 500

# Set sample size
n <- 500

# Initialize vector to store errors for gamma distribution
errors_gamma <- numeric(N)

# Set parameters for gamma distribution
shape <- (population_mean^2)/(population_sd^2)
rate <- shape / population_mean

# Loop over different sample sizes
for (s in 1:N) {

  # Initialize vector to store sample means for gamma distribution
  sample_means_gamma <- numeric(n)

  # Draw random samples and compute means for gamma distribution
  for (i in 1:n) {
    sample_gamma <- pop_distribution(s, shape , rate )
    sample_means_gamma[i] <- mean(sample_gamma)
  }

  # Compute RMSE and store it to errors for gamma distribution
  errors_gamma[s] <- sqrt(mean((sample_means_gamma - population_mean)^2))
}

# Plot errors for gamma distribution against sample size
plot(1:N, errors_gamma, type = "l", xlab = "sample size", ylab = "RMSE for gamma
distribution")

# Create data frame of RMSE values for gamma distribution
error_gamma_df <- data.frame(n = 1:N, RMSE = errors_gamma)
head(error_gamma_df)
```

```
#-------------------------------------------------------------------------------#
# Calculate the difference between each RMSE data points
#-------------------------------------------------------------------------------#

# Normal Distribution #
rmse_diff <- numeric(nrow(error_df))
for (i in 2:nrow(error_df)) {
  rmse_diff[i] <- error_df$RMSE[i] - error_df$RMSE[i-1]
}
error_df$rmse_diff <- rmse_diff
error_df$rmse_diff <- round(error_df$rmse_diff, 3)

# Exponential Distrubution #
rmse_diff_exp <- numeric(nrow(error_exp_df))
for (i in 2:nrow(error_exp_df)) {
  rmse_diff_exp[i] <- error_exp_df$RMSE[i] - error_exp_df$RMSE[i-1]
}
error_exp_df$rmse_diff_exp <- rmse_diff_exp
error_exp_df$rmse_diff_exp <- round(error_exp_df$rmse_diff_exp, 3)


# Gamma Distribution #
RMSE_diff_gamma <- numeric(nrow(error_gamma_df))
for (i in 2:nrow(error_gamma_df)) {
  RMSE_diff_gamma[i] <- error_gamma_df[i, "RMSE"] - error_gamma_df[i-1, "RMSE"]
}
error_gamma_df$RMSE_diff_gamma <- RMSE_diff_gamma
error_gamma_df$RMSE_diff_gamma <- round(RMSE_diff_gamma, 3)

#-------------------------------------------------------------------------------#
# Cost-benefit analysis
#-------------------------------------------------------------------------------#

# Normal Distribution #
# Set up variables
sample_size <- 110
cost_per_person <- 10
total_cost <- sample_size * cost_per_person
mean <- 22
se <- 1.5/sqrt(sample_size)
print (se)
# Calculate the standard error of the mean
sem <- se / sqrt(sample_size)

# Calculate the margin of error (using a 95% confidence level)
z_score <- qnorm(0.975) # 2-tailed test
moe <- z_score * sem

# Calculate the confidence interval
lower_ci <- mean - moe
upper_ci <- mean + moe

# Calculate the expected net benefit
benefit <- (upper_ci - lower_ci) / 2
net_benefit <- benefit - total_cost

# Print the results
cat("Total cost: $", total_cost, "\n")
cat("95% confidence interval: [", lower_ci, ", ", upper_ci, "]\n")
cat("Expected net benefit: $", net_benefit, "\n")
```

```
# Exponential Distribution #
# Set up variables
sample_size_exp <- 40
cost_per_person <- 10
total_cost_exp <- sample_size_exp * cost_per_person
population_sd <- 1.5
mean <- 22

# Calculate the standard error of the mean
se_exp <- population_sd / sqrt(sample_size_exp)

# Calculate the margin of error (using a 95% confidence level)
z_score <- qnorm(0.975) # 2-tailed test
moe_exp <- z_score * se_exp

# Calculate the confidence interval
lower_ci_exp <- mean - moe_exp
upper_ci_exp <- mean + moe_exp

# Calculate the expected net benefit
benefit_exp <- (upper_ci_exp - lower_ci_exp) / 2
net_benefit_exp <- benefit_exp - total_cost_exp

# Print the results
cat("Total cost: $", total_cost_exp, "\n")
cat("95% confidence interval: [", lower_ci_exp, ", ", upper_ci_exp, "]\n")
cat("Expected net benefit: $", net_benefit_exp, "\n")

# Gamma Distribution #
sample_size_gamma <- 27
cost_per_person <- 10
total_cost_gamma <- sample_size_gamma * cost_per_person
population_sd <- 1.5
mean <- 2227

# Calculate the standard error of the mean
se_gamma <- population_sd / sqrt(sample_size_gamma)

# Calculate the margin of error (using a 95% confidence level)
z_score <- qnorm(0.975) # 2-tailed test
moe_gamma <- z_score * se_gamma

# Calculate the confidence interval
lower_ci_gamma <- mean - moe_gamma
upper_ci_gamma <- mean + moe_gamma

# Calculate the expected net benefit
benefit_gamma <- (upper_ci_gamma - lower_ci_gamma) / 2
net_benefit_gamma <- benefit_gamma - total_cost_gamma

# Print the results
cat("Total cost: $", total_cost_gamma, "\n")
cat("95% confidence interval: [", lower_ci_gamma, ", ", upper_ci_gamma, "]\n")
cat("Expected net benefit: $", net_benefit_gamma, "\n")
```

Python code for Normal Distribution Analysis

```
!pip install sympy
import numpy as np
import pandas as pd
from sympy import *
import matplotlib.pyplot as plt
```

```
data=pd.read_csv("error_df.csv")
data = data.drop('sample_size', axis=1)

# Write the updated DataFrame to a new CSV file
data.to_csv('rmse_normal.csv', index=False)
data.head()
import matplotlib.pyplot as plt
class LinearRegression:

    def __init__(self,data,y,degree=1):
        self.data=data
        self.y=y
        self.degree=degree

        #Creat the DesignMatrix can fit differnt degrees
    def DesignMatrix(self):
        df_x=self.data.drop(self.y,1)

        for col in df_x.columns:
            for deg in range(2, self.degree + 1):
                df_x[f"{col}^{deg}"] = df_x[col] ** deg
        df_x.insert(0,'beta',1)
        X=Matrix(df_x.to_numpy())
        self.X=X

    def ResponseMatrix(self):
        df_y=self.data[self.y]
        Y=Matrix(df_y.to_numpy())
        self.Y=Y

    def beta_values(self):
        self.DesignMatrix()
        self.ResponseMatrix()
        return (((self.X.T*self.X).inv())*(self.X.T)*self.Y)

    def y_hat(self):
        self.DesignMatrix()
        self.ResponseMatrix()
        return self.X@(((self.X.T*self.X).inv())*(self.X.T)*self.Y)

    def Residuals(self):
        self.DesignMatrix()
        self.ResponseMatrix()
        return numpy.subtract(self.Y,
self.X@(((self.X.T*self.X).inv())*(self.X.T)*self.Y))

    def norm_Residuals(self):
        self.DesignMatrix()
        self.ResponseMatrix()
        return (self.Y - self.X@(((self.X.T*self.X).inv())*(self.X.T)*self.Y)).norm()

    def y_vs_y_hat_plot(self):
        self.DesignMatrix()
        self.ResponseMatrix()
        return
plt.scatter(self.Y,(self.X@(((self.X.T*self.X).inv())*(self.X.T)*self.Y)))
min_norm = min(norms)
print(f"The minimum norm of residuals is {min_norm}")
# plot norm of residuals for polynonmials versus different degrees
plt.plot(range(1,15 ), norms)
plt.xlabel('Degree')
plt.ylabel('Norm of residuals')
plt.title('Norm of residuals for polynomial fits of different degrees')
```

```
plt.show()

#Fits a polynomial equation with degree of 9
import matplotlib.pyplot as plt
import numpy as np
lr = LinearRegression(data, y='RMSE', degree=9)
beta = lr.beta_values()
y_hat=lr.y_hat()
# plot RMSE with best fitted line
X_plot = np.linspace(data['Unnamed: 0'].min(), data['Unnamed: 0'].max(), 500)
plt.scatter(data['Unnamed: 0'], data['RMSE'])
plt.plot(data['Unnamed: 0'], np.array(y_hat), color='red')
plt.xlabel('Sample Size')
plt.ylabel('RMSE for normal distribution')
plt.title('RMSE with best fitted line ')
plt.show()



# 1.  f'(x) = (f(x + delta_x) - f(x - delta_x)) / 2 * delta_x
delta_x = 0.01
derivative1 = [(y_hat[i+1] - y_hat[i-1]) / (2 * delta_x)
               for i in range(1, len(y_hat)-1)]
print(derivative1)
sum(derivative1)

#2. f'(x) = (f(x + delta_x) - f(x)) / delta_x
derivative2 = [(y_hat[i+1] - y_hat[i]) / (delta_x)
               for i in range(1, len(y_hat)-1)]
print(derivative2)
sum(derivative2)
 # Second derivitive provides a better approximation of the derivitive since dealt x
 is smaller
```

Python code for Exponential Distribution Analysis

```
!pip install sympy
import numpy as np
import pandas as pd
from sympy import *
import matplotlib.pyplot as plt
# Study the change in error rate for exponential distribution
In [5]:
data=pd.read_csv("RMSE_exp.csv")
data['sample_size'] = range(1,501)
data.to_csv('rmse_exp.csv', index = False)
data.head()
class LinearRegression:

    def __init__(self,data,y,degree=1):
        self.data=data
        self.y=y
        self.degree=degree
        #Creat the DesignMatrix can fit differnt degrees
    def DesignMatrix(self):
        df_x=self.data.drop(self.y,1)
```

```
        for col in df_x.columns:
            for deg in range(2, self.degree + 1):
                df_x[f"{col}^{deg}"] = df_x[col] ** deg
        df_x.insert(0,'beta',1)
        X=Matrix(df_x.to_numpy())
        self.X=X

    def ResponseMatrix(self):
        df_y=self.data[self.y]
        Y=Matrix(df_y.to_numpy())
        self.Y=Y

    def beta_values(self):
        self.DesignMatrix()
        self.ResponseMatrix()
        return (((self.X.T*self.X).inv())*(self.X.T)*self.Y)

    def y_hat(self):
        self.DesignMatrix()
        self.ResponseMatrix()
        return self.X@(((self.X.T*self.X).inv())*(self.X.T)*self.Y)

    def Residuals(self):
        self.DesignMatrix()
        self.ResponseMatrix()
        return numpy.subtract(self.Y,
self.X@(((self.X.T*self.X).inv())*(self.X.T)*self.Y))

    def norm_Residuals(self):
        self.DesignMatrix()
        self.ResponseMatrix()
        return (self.Y - self.X@(((self.X.T*self.X).inv())*(self.X.T)*self.Y)).norm()

    def y_vs_y_hat_plot(self):
        self.DesignMatrix()
        self.ResponseMatrix()
        return
plt.scatter(self.Y,(self.X@(((self.X.T*self.X).inv())*(self.X.T)*self.Y)))
# Calculate norm values for degrees 1-14
norms = []
for x in range(1,15):
    lr=LinearRegression(data,y='RMSE',degree=x)
    norm=lr.norm_Residuals()
    norms.append(norm)
    print(norms)
min_norm = min(norms)
print(f"The minimum norm of residuals is {min_norm}")
# plot norm of residuals for polynonmials versus different degrees
plt.plot(range(1, 15), norms)
plt.xlabel('Degree')
plt.ylabel('Norm of residuals')
plt.title('Norm of residuals for polynomial fits of different degrees')
plt.show()
#Fits a polynomial equation with degree of 14
```

```
import matplotlib.pyplot as plt
import numpy as np
lr = LinearRegression(data, y='RMSE', degree=15)
beta = lr.beta_values()
y_hat=lr.y_hat()
X_plot = np.linspace(data['sample_size'].min(), data['sample_size'].max(), 500)
y_hat.shape
X_plot.shape
plt.scatter(data['sample_size'], data['RMSE'])
plt.plot(np.array(X_plot), np.array(y_hat), color='red')
plt.xlabel('Sample Size')
plt.ylabel('RMSE for Exponential Distribution')
plt.title('RMSE with best fitted line ')
plt.show()
X_plot.sha
# 1. Simulate f'(x) = (f(x + delta_x) - f(x - delta_x)) / 2 * delta_x
delta_x = 0.01
derivative1 = [(y_hat[i+1] - y_hat[i-1]) / (2 * delta_x)
               for i in range(1, len(y_hat)-1)]
print(derivative1)
sum(derivative1)
#2. Simulate f'(x) = (f(x + delta_x) - f(x)) / delta_x
derivative2 = [(y_hat[i+1] - y_hat[i]) / (delta_x)
               for i in range(1, len(y_hat)-1)]
print(derivative2)
sum(derivative2)
# Second derivitive provides a better approximation of the derivitive since dealt x is
smaller
```

Python code for Gamma Distribution Analysis

```
import numpy as np
import pandas as pd
from sympy import *
import matplotlib.pyplot as plt
# Study the change in error rate for Gamma distribution
data=pd.read_csv("RMSE_gamma.csv")
data.head()

class LinearRegression:

   def __init__(self,data,y,degree=1):
       self.data=data
       self.y=y
       self.degree=degree

       #Creat the DesignMatrix can fit differnt degrees
   def DesignMatrix(self):
       df_x=self.data.drop(self.y,1)

       for col in df_x.columns:
           for deg in range(2, self.degree + 1):
               df_x[f"{col}^{deg}"] = df_x[col] ** deg
```

```
        df_x.insert(0,'beta',1)
        X=Matrix(df_x.to_numpy())
        self.X=X

    def ResponseMatrix(self):
        df_y=self.data[self.y]
        Y=Matrix(df_y.to_numpy())
        self.Y=Y

    def beta_values(self):
        self.DesignMatrix()
        self.ResponseMatrix()
        return (((self.X.T*self.X).inv())*(self.X.T)*self.Y)

    def y_hat(self):
        self.DesignMatrix()
        self.ResponseMatrix()
        return self.X@(((self.X.T*self.X).inv())*(self.X.T)*self.Y)

    def Residuals(self):
        self.DesignMatrix()
        self.ResponseMatrix()
        return numpy.subtract(self.Y,
self.X@(((self.X.T*self.X).inv())*(self.X.T)*self.Y))

    def norm_Residuals(self):
        self.DesignMatrix()
        self.ResponseMatrix()
        return (self.Y - self.X@(((self.X.T*self.X).inv())*(self.X.T)*self.Y)).norm()

    def y_vs_y_hat_plot(self):
        self.DesignMatrix()
        self.ResponseMatrix()
        return
plt.scatter(self.Y,(self.X@(((self.X.T*self.X).inv())*(self.X.T)*self.Y)))


# Calculate norm values for degrees 1-14
norms = []
for x in range(1,15):
    lr=LinearRegression(data,y='RMSE',degree=x)
    norm=lr.norm_Residuals()
    norms.append(norm)
    print(norms)

min_norm = min(norms)
print(f"The minimum norm of residuals is {min_norm}")


# plot norm of residuals for polynonmials versus different degrees
plt.plot(range(1, 15), norms)
plt.xlabel('Degree')
plt.ylabel('Norm of residuals')
plt.title('Norm of residuals for polynomial fits of different degrees')
plt.show()
```

```
#Fits a polynomial equation with degree of 14
import matplotlib.pyplot as plt
import numpy as np
lr = LinearRegression(data, y='RMSE', degree=14)
beta = lr.beta_values()

y_hat=lr.y_hat()

# plot RMSE with best fitted line
X_plot = np.linspace(data['n'].min(), data['n'].max(), 500)
y_hat.shape
X_plot.shape

plt.scatter(data['n'], data['RMSE'])
plt.plot(np.array(X_plot), np.array(y_hat), color='red')
plt.xlabel('Sample Sizd')
plt.ylabel('RMSE for Gamma Distribution')
plt.title('RMSE with fitted line')
plt.show()


# 1. Simulate f'(x) = (f(x + delta_x) - f(x - delta_x)) / 2 * delta_x
delta_x = 0.01
derivative1 = [(y_hat[i+1] - y_hat[i-1]) / (2 * delta_x)
               for i in range(1, len(y_hat)-1)]
print(derivative1)

sum(derivative1)
#2. Simulate f'(x) = (f(x + delta_x) - f(x)) / delta_x
derivative2 = [(y_hat[i+1] - y_hat[i]) / (delta_x)
               for i in range(1, len(y_hat)-1)]
print(derivative2)

sum(derivative2)
# Second derivitive provides a better approximation of the derivitive since dealt x is
smaller
```