trends in higher education

MDPI

Article

Artificial Intelligence in Statistics Education: Leveraging LLMs for Analysis and Learning

Enrico di Bella † **and Sara Preti** *,†

Department of Political and International Sciences, University of Genoa, 16125 Genoa, Italy; enrico.dibella@unige.it

* Correspondence: sara.preti@edu.unige.it

† These authors contributed equally to this work.

Abstract

Large Language Models (LLMs), such as GPT (GPT-5.5) by OpenAI and Gemini (Gemini 3.2 Pro) by Google DeepMind, have shown impressive capabilities in text generation and code assistance. This study evaluates their performance in generating R code—that is, computer scripts written in the R programming language for statistical analysis—using both classic educational datasets, including "The Lady Tasting Tea", "Titanic", "Iris", and more recent datasets likely not included in the models' training data. We assess the accuracy, readability, and educational relevance of the generated code, providing both quantitative and qualitative evaluations that highlight strengths and limitations of LLMs. Our findings suggest that while LLMs generate correct and interpretable R code in many cases, critical human oversight remains essential when integrating AI into educational contexts to ensure rigor and avoid potential misuse.

Keywords: artificial intelligence; large language models; GPT; Gemini; R programming; code generation; statistics education

1. Introduction

Artificial Intelligence (AI) emerged as a formal field in 1956 during the Dartmouth Summer Research Project, where the term was coined by John McCarthy [1]. Although its roots extend to earlier philosophical and computational ideas [2], AI has evolved into a branch of computer science focused on developing machines capable of mimicking and enhancing human cognitive functions. Through advanced algorithms and data structures, AI systems can solve complex problems, recognize patterns, learn from past experiences, and make decisions [3]. Over the decades, AI has undergone significant evolution, experiencing alternating periods of rapid advancement and temporary stagnation [4]. Today, AI is integrated across many sectors, from industrial automation to medicine, finance, and entertainment.

A breakthrough in AI development was the rise of Machine Learning (ML), a technique that allows computers to learn from data without being explicitly programmed [5]. Further advances led to Deep Learning (DL), a subcategory of ML based on deep neural networks, which dramatically improved capabilities in areas such as image recognition and natural language understanding [6].

In recent years, the emergence of Large Language Models (LLMs) has expanded the boundaries of AI capabilities. Examples include Generative Pre-trained Transformer (GPT) models, a family of transformer-based language models developed by OpenAI and widely

Academic Editors: Hani Morgan, Carlos Hervás-Gómez, María Dolores Díaz-Noguera, Liliana Mâță and Nadia Barkoczi

Received: 19 July 2025
Revised: 19 February 2026
Accepted: 16 March 2026
Published: 7 May 2026

used for natural language generation and tasks [7–9], and Gemini, a multimodal large language model developed by Google DeepMind, designed to process and generate content across multiple data types [10]. These models exhibit unprecedented abilities in generating human-like text and code across a wide range of contexts. Their capacity to produce syntactically correct and functional code has significant implications in many fields. By automating tasks such as code completion, debugging, and even algorithm design, LLMs can accelerate workflows and enhance productivity. These capabilities extend beyond technical domains, like software development, and have profound implications for education. LLMs are increasingly transforming how learning and teaching take place, offering new opportunities for data analysis, personalized feedback, and instructional support [11]. Recent studies distinguish between two complementary perspectives: Artificial Intelligence Literacy (AIL), which focuses on helping people understand how AI systems work and their broader social implications, and Artificial Intelligence in higher Education (AIED), which examines how AI technologies can be responsibly integrated to enhance teaching and learning processes [12,13]. Within the AIED framework, the emphasis is placed on using AI tools, particularly LLMs, in educational courses in an effective and responsible manner, while maintaining human oversight and ethical integrity [14]. This view aligns with the "humans in the loop" paradigm, which envisions educators and AI systems as collaborative partners, each complementing the other's strengths to create more engaging, inclusive, and effective learning experiences [15].

A notable application of LLMs is their use in statistical analysis. Particularly, LLMs can access data stored in shared folders (such as Google Drive, OneDrive, etc.), generate code (e.g., in Python or R) to analyze these data, and automatically produce reports with detailed explanations from natural language prompts. These features enable the integration of AI systems into research and educational settings, where students and researchers can conduct complex analyses more efficiently.

Despite these promising capabilities, the increasing reliance on LLMs raises important concerns regarding accuracy and the need for appropriate oversight. At the same time, statistical methods play an increasingly visible role across a wide range of research disciplines [16]. Simple indicators, such as keyword frequencies in bibliographic databases, can serve as illustrative signals of this broad presence, although they should not be interpreted as evidence of systematic trends. For instance, a search on Scopus shows that the number of papers mentioning the word "ANOVA" in their abstracts increased from 2333 in 2004 to 14,260 in 2023. This example is not intended as proof of a general trend, but merely as an illustration of the widespread use of statistical terminology in contemporary research. While the number of professional statisticians has grown in some countries (e.g., according to US Bureau of Labor Statistics occupational data [17]) the scale, diversity and timing of analytical demand across research communities may still result in uneven access to specialized statistical expertise or to timely expert review. Importantly, many researchers receive additional statistical training or collaborate with professional statisticians; our argument is not that such avenues do not exist, but rather that the combination of (i) increasing use of advanced methods, (ii) variable access to expert support across institutions and countries, and (iii) the growing availability of automated analytical tools underscores the continued importance of careful oversight and targeted statistics education.

Although LLMs, such as GPT and Gemini, show potential in supporting these tasks, analyses performed without proper oversight pose risks. Automated analyses that are not critically reviewed by experts can lead to poorly conducted studies and undermine the reliability of research findings. This situation underscores the need for a balanced approach: integrating these tools with critical human evaluation to ensure scientific rigor and reliability.

Statistics education faces similar challenges. Many students view statistics as a difficult subject [16,18,19], often taught in limited hours and perceived as merely a requirement for publication rather than a fundamental skill. This situation can push learners to rely heavily on LLMs for convenience and immediacy, especially when professional statisticians are unavailable. However, if guided appropriately, LLMs can bridge the gap between the growing demand for statistical skills and the limited availability of expert support, provided they are used critically and responsibly.

In this study, we evaluate the capabilities of LLMs, specifically GPT and Gemini, in generating statistical analysis code in R. R is an open-source programming language widely used in education due to its extensive set of packages for statistical learning and visualization. To test LLMs' coding capabilities, we utilize both classic and more recent, less familiar datasets. By analyzing the accuracy and readability of the generated code, we aim to understand how LLMs can enhance statistical learning and research. The subsequent sections will provide an overview of LLMs, detail our datasets, experimental design, and evaluation methodology, present case studies, and conclude with a discussion of our findings.

2. Large Language Models: An Overview

Large Language Models are advanced artificial intelligence systems designed to process and generate natural language by learning statistical regularities from large text corpora drawn from different sources, including books, articles, websites, and social media. Most contemporary LLMs are based on transformer architectures, such as GPT and BERT (Bidirectional Encoder Representations from Transformers (BERT is a transformer-based language model introduced by Google in 2018 to enhance query understanding in Google search engine. Unlike autoregressive models, it learns bidirectional contextual representations by considering both preceding and following words in a sentence), which rely on self-attention mechanisms to model contextual relationships between tokens (tokens are the basic units—such as words, character sets, or combinations of words and punctuation—used by LLMs to represent and process text) and capture long-range dependencies in text. This architectural design enables LLMs to generate coherent and contextually appropriate outputs across a wide range of tasks.

In practice, LLMs are typically trained in two stages. During pre-training, models learn general linguistic knowledge from large-scale text data using unsupervised objectives, such as predicting missing or subsequent tokens. This general knowledge is then refined through task-specific fine-tuning using supervised learning, often supplemented by human feedback, to improve usefulness and alignment with user expectations [20,21]. When generating text or code, LLMs operate probabilistically, producing outputs token by token based on the given prompt and learned representations.

LLMs are currently available both as commercial, subscription-based systems and as open-source models that can be locally deployed. While commercial models often provide higher performance due to extensive training and continuous updates, open-source alternatives offer greater transparency and control.

As advanced AI tools, LLMs are applied across a wide range of contexts. In natural language processing, they support tasks such as text generation, summarization, translation, and conversational agents. In statistical analysis, LLMs can assist in data interpretation, report generation, and insight extraction by producing human-readable summaries of analytical results. They are also used in content creation, customer support, and, increasingly, in specialized fields such as healthcare.

Of particular relevance to this study is the ability of LLMs to generate and explain programming code. Recent research has shown that LLMs can function effectively as

coding assistants across several programming languages, supporting tasks such as code completion, debugging, and explanation [22,23]. However, existing evaluations have focused predominantly on languages such as Python, JavaScript, and C++ [24–33], while comparatively little attention has been given to R, despite its central role in statistics education and data analysis [34–37]. Beyond its widespread adoption, R offers pedagogical advantages particularly relevant for statistics education: as an open-source and code-based environment, it promotes reproducibility, transparency of analytical steps, and the explicit representation of statistical reasoning. These characteristics are especially important in educational settings, where making analytical processes visible supports conceptual learning rather than mere procedural execution [34]. To date, however, relatively few studies have systematically examined the performance of LLMs in generating R code within authentic statistics education scenarios, where both computational accuracy and conceptual reasoning are pedagogically relevant outcomes.

Against this background, the present study evaluates the performance of LLMs in generating R code for educationally relevant statistical tasks. Rather than assessing general language capabilities, we examine how effectively these models—specifically GPT and Gemini—support statistical programming and reasoning in learning-oriented scenarios, using a combination of quantitative accuracy measures and qualitative error analysis.

3. Materials and Methods

3.1. Data and Applications

We used several datasets to evaluate the capabilities of LLMs in generating R code for statistical analysis. First, we employed classic datasets widely used in statistics education: the "Lady Testing Tea", "Titanic", and "Iris" datasets. These allow for an initial assessment of the models' performance on familiar educational examples.

To explore how LLMs handle more contemporary and less familiar data, we included the 2024 OECD Better Life Index dataset. Finally, to test the models on a more challenging and novel scenario, we used the publicly available "Extrovert vs. Introvert Behavior" dataset from Kaggle. This dataset is unlikely to have been part of the LLMs' training data, providing a rigorous test of their generalization and coding ability.

3.1.1. Lady Tasting Tea Dataset: Fisher's Exact Test

The "Lady Testing Tea" experiment was designed by Ronald A. Fisher in 1925 [38]. During a tea party, Muriel Bristol (the lady) claimed she could identify the sequence of pouring in a tea-milk mixture. Fisher tested this assertion by preparing eight cups of tea, four with milk poured first and four with tea poured first, presented in random order. The hypotheses can be formulated as follows:

- H_0: The lady's predictions are random (no ability)
- H_1: The lady's predictions are not random (some ability)

This experiment is historically significant, as it was used by Fisher to introduce concepts of null hypothesis, p-value, and for the subsequent development of Fisher's Exact Test for 2×2 contingency tables. In brief, the experiment can be analyzed using the test statistic for Fisher's Exact Test given by the hypergeometric probability mass function, as follows:

$$P(X = x) = \frac{C(m, x) \cdot C(N - m, n - x)}{C(N, n)} \quad (1)$$

where C represents the binomial coefficient, N is the total number of cups, n is the number of successes (correct predictions), and m is the total number of cups prepared one way. In this work, we assume that the lady correctly identified three out of four cups presented in either pouring mix (i.e., tea before milk or milk before tea).

3.1.2. Titanic Dataset: Logistic Regression

The "Titanic" dataset contains information on the passengers of the Titanic, the famous ship that sank in 1912 after hitting an iceberg. There are various versions of this dataset, all derived from the British Board of Trade Inquiry [39]. The dataset of the `titanic` R package is split into a training (`titanic_train`, 891 passengers) and a test (`titanic_test`, 481 passengers) set. The main variables in the dataset are:

1. PassengerId: A unique identifier assigned to each passenger.
2. Survived: An indicator if the passenger survived or not (0 = No, 1 = Yes).
3. Pclass: The class of the ticket purchased by the passenger (1 = first, 2 = second, 3 = third). This can also be an indicator of the passenger's socio-economic status.
4. Name: The name of the passenger.
5. Sex: The sex of the passenger.
6. Age: The age of the passenger.
7. SibSp: The number of siblings or spouses of the passenger aboard the ship.
8. Parch: The number of parents or children of the passenger aboard the ship.
9. Ticket: The ticket number of the passenger.
10. Fare: The fare paid by the passenger for the journey.
11. Cabin: The cabin number of the passenger.
12. Embarked: The port of embarkation of the passenger (C = Cherbourg; Q = Queenstown; S = Southampton).

The Titanic dataset is widely used in data science and statistics education due to its historical relevance, ease of understanding, and inherent complexity. Despite its accessibility, it presents numerous data analysis challenges, such as missing values and varied variable types, making it a valuable tool for teaching and learning data analysis and machine learning techniques. One of the most immediate models that can be applied to the Titanic dataset is the logistic regression to predict the probability of survival based on passenger characteristics (input features). In logistic regression, the probability (P) of a particular event occurring is related to the input features $(x_1, x_2, \ldots, x_n)$ through the following relationship (see, among the others, Hastie et al., (2009) [40]):

$$P(Y = 1) = \frac{1}{1 + e^{-(b_0 + b_1 x_1 + b_2 x_2 + \ldots + b_n x_n)}} \quad (2)$$

where Y is the binary output (Survived or Not survived), b_0 is the intercept, and $(b_1, b_2, \ldots, b_n)$ are the coefficients corresponding to input features $(x_1, x_2, \ldots, x_n)$ (such as Age, Sex, Fare, etc.). The coefficients are typically estimated using a method called maximum likelihood estimation.

3.1.3. Iris Dataset: Plotting Clusters

The "Iris" dataset, created by the British statistician and biologist Ronald Fisher [41], includes measurements of 50 samples from each of three Iris flower species: Iris setosa, Iris virginica, and Iris versicolor. Each sample records includes four features: sepal length, sepal width, petal length, and petal width, all measured in centimeters. The dataset is widely used in classification and clustering examples. In this study, we tasked the LLMs with identifying possible clusters and evaluating cluster quality. The main objective of the clustering methods is to identify homogeneous subgroups among observations. Essentially, there are two common clustering methods:

- K-means clustering: The algorithm begins by randomly assigning each observation to a predefined number of clusters K. It then iteratively calculates the centroid (the centroid is the vector of the averages of the p variables for the observations in the k-th cluster) for each cluster and reassigns each observation to the cluster with the

nearest centroid (proximity is usually measured by the quadratic Euclidean distance: $d_{i,j} = \sqrt{\sum_{k=1}^{n}(x_{ik} - y_{jk})^2}$). This process repeats until the cluster assignments stabilize and no longer change.

- Hierarchical clustering: Builds a hierarchy of clusters, most commonly through an agglomerative (bottom-up) approach where individual observations are successively merged into larger clusters based on their proximity. This process continues until all observations are merged into a single cluster (conversely, a divisive (top-down) method starts with a single large cluster and recursively splits it until each observation forms its own cluster).

3.1.4. Better Life Index 2024 Dataset: Principal Component Analysis

We utilized the 2024 OECD Better Life Index (BLI) dataset, downloaded from the OECD website. This dataset allows to measure a country's well-being beyond the traditional Gross Domestic Product by aggregating a wide and heterogeneous range of social and economic metrics. The data comprise 24 indicators across 11 topics, detailed in Table 1. The extensive information contained within the BLI makes it an excellent test bed for LLMs in generating R code for complex statistical analysis. Specifically, given the extensive number of indicators, we prompted the LLMs to perform dimensionality reduction and efficiently summarize the information, providing illustrative R code. It is worth noting that, although the 2024 data are recent, earlier editions of the Better Life Index may have been included in the LLM training data, despite potential variations across annual releases.

Table 1. OECD's topics and indicators for Better Life Index.

Topics	Indicators
Housing	Dwellings without basic facilities Rooms per person Housing expenditure
Income	Household net adjusted disposable income Household net financial wealth
Jobs	Labour market insecurity Employment rate Long term unemployment rate Personal earnings
Education	Educational attainment Students' cognitive skills Expected years in education
Environment	Air pollution Satisfaction with water quality
Civic Engagement	Stakeholder engagement for regulations Voter turnout
Health	Life expectancy at birth Self-reported health status
Safety	Feeling safe walking alone at night Homicide rates
Work-Life Balance	Employees working very long hours Time devoted to leisure and personal care
Community	Social network support
Life Satisfaction	Life satisfaction

Potential pitfalls for students could relate to the data pre-processing steps (verification of indicator types and their standardization) which, if not performed before the dimensionality reduction method, would provide biased or incorrect results.

3.1.5. Extrovert vs. Introvert Behavior Dataset: Supervised Machine Learning

To further challenge the LLMs, we focus on their ability to generate R code for a supervised machine learning problem. We used a dataset from `Kaggle` containing behavioral and social data for 2900 individuals, labeled as either "Extrovert" or "Introvert" [42]. This dataset is valuable as it offers concrete insights into these psychological constructs through observable metrics, detailed below:

1. Time_spent_alone: Number of hours an individual typically spends alone daily, ranging from 0 to 11 h.
2. Stage_fear: Experience of stage fright (Yes) or not (No).
3. Social_event_attendance: Frequency of attending social events, on a scale from 0 to 10.
4. Going_outside: Frequency of going outside, with a range from 0 to 7.
5. Drained_after_socializing: Feeling drained after socializing (Yes/No).
6. Friends_circle_size: Number of close friends, ranging from 0 to 15.
7. Post_frequency: Frequency of posting on social media, on a scale from 0 to 10.
8. Personality: Target variable for classification, designating individuals as either an Extrovert or an Introvert.

We expected the LLMs to generate R code covering data pre-processing, data splitting into training and test sets, model selection and tuning, and evaluation of performance metrics. Compared to prior case studies, this particular application introduces several layers of complexity, necessitating the use of various methodologies and R packages (R 4.4.1). Furthermore, given the recent publication of these data, it is unlikely they were included in the LLM training process.

Consequently, the inherent difficulty of this multifaceted classification task demands student expertise beyond basic data preparation (e.g., handling heterogeneous data types and missing values), also requiring critical judgment in model selection, hyperparameter tuning, and performance evaluation.

3.2. Experimental Design and Evaluation Criteria

We evaluated the performance of GPT and Gemini, specifically GPT-3.5, GPT-4o (an optimized version of GPT-4 with improved efficiency and processing capabilities), and Gemini Pro (the freely available version based on Google's Pro model), which are widely used and share similar core functionalities.

These LLMs can simplify programming tasks for non-expert users and support learning processes [43]. For this reason, the models were provided with identical prompts designed to simulate typical queries from students or researchers with limited programming and statistical backgrounds. This approach ensures comparability across models and allow us to assess their ability to interpret and respond effectively in educational contexts. The prompts followed a standard template. Specifically, each prompt included the following elements: (i) Scenario and task description: a brief description of the context and the analytical goal (e.g., "Identify possible clusters and a measure of the goodness of the clustering"); (ii) Dataset description: the name of the data and a list of the main available variables; (iii) Desired output format: the models were asked to produce only executable R code. In general, this information corresponds to what instructors or textbooks typically provide to allow students to correctly address a statistical problem. No further specific instructions were given to the LLMs, such as requesting explanatory comments or

the use of particular R packages, in order to evaluate both the readability and accuracy of the generated code under minimal guidance conditions.

To objectively evaluate the LLMs' generated R code, we used the R package `lintr` [44]. This static analysis tool assesses code for style, syntax, and structural consistency, promoting readability and adherence to coding standards by flagging common issues like improper spacing, naming conventions, and line length, as well as potential programming or stylistic errors or warnings. However, because lintr perform static analysis, it cannot detect semantic errors related to logic or functionality (such as missing arguments or the absence of required package imports). Furthermore, no existing tool can determine whether the generated R code fully satisfies the user's original prompt.

To address this limitation, we qualitatively assessed each response's usefulness and overall quality using a 1 to 5 rating scale. Specifically, we evaluated the following dimensions:

- Accuracy: Correctness of code and statistical analysis, accuracy of outputs, ability to handle edge cases or unusual inputs, range of tasks supported, proper use of libraries, and the model's understanding of the prompt.
- Readability: Code clarity, inclusion of helpful comments or documentation, and adherence to style guidelines (this includes verifying whether the code aligns with commonly accepted R programming conventions and best practices).

This combined approach allowed us to objectively and subjectively evaluate the educational usefulness of LLM-generated code.

4. Results

In this section, we present each case study, detailing the prompts submitted to the LLMs, their generated outputs, and a commentary on each response.

4.1. Case Studies

4.1.1. Results for the Lady Tasting Tea Dataset

Question to LLMs: *A lady declares that by tasting a cup of tea made with milk, she can discriminate whether the milk or tea infusion was first added to the cup. We will consider the problem of designing an experiment through which this assertion can be tested. It consists of mixing eight cups of tea, four in one way and four in the other, and presenting them to the subject for judgment in random order. The subject has been told that the test will consist, namely, that she will be asked to taste eight cups, four of each kind. The lady in question answered correctly in six out of eight trials. The results are the following: (1) tea poured first: three correct answers out of four cups; (2) milk poured first: three correct answers out of four cups.*
With these results, what should be concluded about the lady's ability to discriminate whether milk or tea was poured first? Propose a method to assess this statistically and the R code to run the analyses. Provide the response formatted in Latex, including the R code.

Below are the responses provided by the LLMs and the R code they generated (Listings 1–3), while Table 2 reports the contingency table used for the implementation of Fisher's exact test proposed by GPT-4o.

GPT-3.5 response: To assess the lady's ability to discriminate whether milk or tea was poured first, we can use a binomial test. The null hypothesis would be that the lady's ability to discriminate is no better than random chance (50% correct), and the alternative hypothesis would be that she can discriminate with better accuracy. To perform the statistical analysis in R, we can use the `binom.test()` function. Here's the R code to run the analyses:

Listing 1. R code generated by GPT-3.5 for the Lady Tasting Tea experiment.

```
% Number of correct answers for tea poured first and milk poured first
tea_correct <- 3
milk_correct <- 3

% Number of trials for tea poured first and milk poured first
tea_trials <- 4
milk_trials <- 4

% Perform the binomial test
tea_result <- binom.test(tea_correct, tea_trials, p = 0.5, alternative = "greater")

milk_result <- binom.test(milk_correct, milk_trials, p = 0.5, alternative = "greater")

% Print the test results
print(tea_result)
print(milk_result)
```

The `binom.test()` function calculates the p-value for the binomial test, comparing the observed number of correct answers to the expected number under the null hypothesis of random chance (50% probability). By specifying `alternative = ‘‘greater’’` we are testing whether the lady's accuracy is greater than random chance. The test results will provide the p-values for both conditions (tea poured first and milk poured first). If the p-value is less than the chosen significance level (e.g., 0.05), we can reject the null hypothesis and conclude that the lady's ability to discriminate is significantly better than the random chance.

Note: in this specific case, the lady answered correctly in six out of eight trials, so it seems that she has some ability to discriminate. However, the statistical test will provide a formal assessment of the significance of her ability.

GPT-4o response: The experimental data can be analyzed using Fisher's Exact Test. We first construct a 2×2 contingency table as follows:

Table 2. Contingency table for the tea testing experiment.

	Correct Guess	**Wrong Guess**
Milk First	3	1
Tea First	3	1

Fisher's Exact Test can be used to determine if there are nonrandom associations between two categorical variables. In this case, we're interested in whether there's an association between the type of preparation (milk or tea poured first) and the lady's ability to guess correctly. This can be conducted in R using the following code:

Listing 2. R code generated by GPT-4o for the Lady Tasting Tea experiment.

```
# Create 2x2 contingency table
data <- matrix(c(3, 1, 3, 1), nrow = 2)

# Assign row and column names
dimnames(data) <- list(c("Milk First","Tea First"), c("Correct", "Wrong"))

# Perform Fisher's exact test
result <- fisher.test(data)

# Print the p-value
print(result$p.value)
```

This code will output the *p*-value of the test, which gives the probability of obtaining a result as extreme as the observed one under the null hypothesis. If this *p*-value is smaller than a chosen significance level (usually 0.05), then we can reject the null hypothesis, that the lady's guesses are random, and conclude that the lady does indeed have some ability to discern whether milk or tea was poured first.

Gemini response: The lady's claim of being able to discriminate between milk-first and tea-first tea can be tested statistically.
We can use a binomial test to assess if the lady's success rate (6 out of 8) deviates significantly from what we would expect by chance alone (i.e., guessing). The binomial test assumes:

- Only two outcomes are possible (correct or incorrect) for each trial (cup).
- Trials are independent (the outcome of one cup doesn't influence another).
- The probability of success (correct guess) remains constant across trials.

In this case:

- Success = Correctly identifying milk-first or tea-first tea.
- Trials (n) = 8 cups of tea.
- Expected probability of success (p) = 0.5 (assuming pure chance, guessing).

We want to see if the observed success rate (6/8) provides evidence against the null hypothesis (H_0): p = 0.5 (the lady cannot discriminate).

Listing 3. R code generated by Gemini for the Lady Tasting Tea experiment.

```
# Define success (number of correct answers) and trials
success <- 6
trials <- 8

# Perform binomial test
binom.test(success, trials, p = 0.5, alternative = "two.sided")
```

The R output will provide the *p*-value associated with the test. A low *p*-value (<0.05) suggests we can reject the null hypothesis and conclude the lady's success rate is unlikely due to chance alone. Conversely, a high *p*-value (>0.05) suggests we fail to reject the null hypothesis and there's insufficient evidence to claim the lady can discriminate.

Comment to LLMs' responses on the "Lady Tasting Tea" dataset: GPT-3.5 and GPT-4o propose different statistical tests for analyzing the results. GPT-3.5 recommends a binomial test, which is inappropriate in this context. The proposed R code performed two separate binomial tests against a $p = 0.5$ alternative, rather than a unified test for the two tea-milk pouring mixes. While a binomial test could assess if guessing was better than chance, our objective is to verify whether the pouring order (milk first or tea first) influences the lady's correct guesses. For this, the Fisher exact test proposed by GPT-4o is the more fitting choice, as it considers more the success rates in both conditions, making it a more powerful test than the binomial test, which only considers the overall success rate. However, despite the correct table provided in the text by GPT-4, its implementation in the R code is wrong. As additional remarks, GPT-3.5 misuses the "%" character for R comments and includes LaTeX syntax errors such as "50%" instead of "50 \%". Despite these issues, both models produce functioning R code. Gemini, like GPT-3.5, selects a binomial test, which is less appropriate for this experimental design. However, it appears more precise in its model explanation and result interpretation than GPT.

4.1.2. Results for the Titanic Dataset

Question to LLMs: *I have the R datatable "titanic_data" referring to the shipwreck of a large cruise ship. The data contains information about the passengers and whether they survived the*

shipwreck. In particular, the fields of the data table are: PassengerId: A unique identifier assigned to each passenger. Survived: An indicator if the passenger survived or not (0 = No, 1 = Yes). Pclass: The class of the ticket purchased by the passenger (1 = first class, 2 = second class, 3 = third class). This can also be an indicator of the passenger's socio-economic status. Name: The name of the passenger. Sex: The sex of the passenger. Age: The age of the passenger. SibSp: The number of siblings or spouses of the passenger aboard the ship. Parch: The number of parents or children of the passenger aboard the ship. Ticket: The ticket number of the passenger. Fare: The fare paid by the passenger for the journey. Cabin: The cabin number of the passenger. Embarked: The port of embarkation of the passenger (C = Cherbourg; Q = Queenstown; S = Southampton). Suggest which could be relevant fields to include in a statistical model to predict the passenger probability of survival, propose the statistical method you think is most opportune given the information I provided, and display the R code to estimate the model.

Below are the responses provided by the LLMs and the R code they generated (Listings 4–6).

GPT-3.5 response: To predict the passengers' probability of survival, we can consider a logistic regression model using the following relevant fields from the `titanic_data` data table:

- Survived (target variable): This is the indicator variable that represents whether the passenger survived or not. It will be the dependent variable in the statistical model.
- Pclass: The class of the ticket purchased by the passenger can be an important indicator of socioeconomic status and potential survival probabilities.
- Sex: Gender may have played a significant role in the survival rates during the shipwreck.
- Age: Age could be a relevant factor as it might have affected the priority given to certain age groups during the rescue efforts.
- SibSp: The number of siblings or spouses aboard the ship might have influenced the chances of survival.
- Parch: The number of parents or children aboard the ship could have affected the survival rates.
- Fare: The fare paid by the passenger might be an indicator of socio-economic status and could potentially impact survival probabilities.
- Embarked: The port of embarkation might be associated with different socio-economic backgrounds and could be a relevant factor in the model.

We can estimate the logistic regression model in R using the `glm` function:

Listing 4. R code generated by GPT-3.5 for logistic regression on the Titanic dataset.

```
# Fit the logistic regression model
model <- glm(Survived ~ Pclass + Sex + Age + SibSp + Parch + Fare + Embarked, data =
    ↪ titanic_data, family = binomial)

# Display the model summary
summary(model)
```

In the above code, we are using the `glm` function to fit the logistic regression model. The formula *Survived~Pclass* + *Sex* + *Age* + *SibSp* + *Parch* + *Fare* + *Embarked* specifies the relationship between the dependent variable (`Survived`) and the predictor variables (`Pclass, Sex, Age, SibSp, Parch, Fare, Embarked`). The `family = binomial` argument specifies the binomial family, which is appropriate for logistic regression. After fitting the model, you can use the `summary` function to display the model summary, which provides information about the estimated coefficients, their significance, and other statistical measures.

GPT-4o response: Given the variables in the "titanic_data" data table, the following fields could be pertinent in predicting a passenger's survival probability:

- Pclass: The ticket class might be an indicator of the passenger's socio-economic status.
- Sex: Gender might have played a role in survival rates due to the "women and children first" protocol.
- Age: Similar to gender, age might have played a role in survival rates.
- SibSp and Parch: The number of family members could have influenced the survival rate.
- Fare: This might correlate with "Pclass" and socio-economic status.
- Embarked: The port of embarkation might be a proxy for socio-economic status.

We could use Logistic Regression for this problem, which is suitable for binary classification problems. The following R code can be used to build the Logistic Regression model using the selected variables.

Listing 5. R code generated by GPT-4o for logistic regression on the Titanic dataset.

```
# Import required library
library(data.table)

# Ensure data is in data.table format
titanic_data <- as.data.table(titanic_data)

#Build the Logistic Regression model
logistic_model <- glm(Survived ~ Pclass + Sex + Age + SibSp + Parch + Fare + Embarked,
    ↪ family = binomial(link=logit), data = titanic_data)

# Display the model summary
summary(logistic_model)
```

In this R code, the `glm` function is used to fit the logistic regression model. The family argument is set to binomial to specify that we are modeling a binary outcome, and link='logit' to use the logit function.

Remember to handle missing values in your data, either by imputation or exclusion, depending on the amount and nature of the missing data.

Also, please make sure to convert Sex and Embarked to factor variables if they are not already in that format. These variables are categorical in nature and should be treated as such in the model.

Please, ensure that the 'Sex' and 'Embarked' fields are in factor format before running the model. Also, you should handle missing values appropriately, either by imputation or exclusion. Please note that PassengerId, Name, Ticket, and Cabin fields have been excluded from the model as they're less likely to be of relevance in predicting survival.

Gemini response: Here are the fields from your "titanic" data table that could be relevant for predicting passenger survival probability:

- Survived (Target Variable): This is the variable we want to predict (0 = No, 1 = Yes).
- Pclass: Ticket class can be an indicator of socio-economic status, potentially impacting access to lifeboats.
- Sex: Men and women may have had different survival rates due to social norms and lifeboat capacity.
- Age: Age could influence survival chances due to physical capabilities.
- SibSp: Number of siblings/spouses aboard might influence survival if they acted together.
- Parch: Number of parents/children aboard could impact survival decisions based on family responsibility.
- Fare: Higher fare might indicate access to better cabins closer to lifeboats.

- Embarked: Embarkation port might influence proximity to lifeboats when the evacuation started.

The most appropriate statistical method for this scenario is Logistic Regression. It's a supervised learning method used for binary classification problems like this, where the outcome variable (Survived) has two possible values (0 or 1). It models the relationship between the independent variables (Pclass, Sex, Age, etc.) and the probability of a specific outcome (survival).

Listing 6. R code generated by Gemini for logistic regression on the Titanic dataset.

```
# Load libraries (if not already loaded)
library(dplyr)

# Load the data
data("titanic")  # Assuming "titanic" is a data frame

# Select relevant features
data <- titanic %>%
  select(Survived, Pclass, Sex, Age, SibSp, Parch, Fare, Embarked)

# Convert categorical variables to factors (if needed)
data$Sex <- factor(data$Sex)
data$Embarked <- factor(data$Embarked)

# Build the logistic regression model
model <- glm(Survived ~ Pclass + Sex + Age + SibSp + Parch + Fare + Embarked, data =
    ↪ data, family = binomial)

# Print the model summary
summary(model)
```

This code performs the following steps:

1. Loads the `dplyr` library for data manipulation (if not already loaded).
2. Loads the "titanic" data assuming it's a data frame.
3. Selects the relevant features for the model.
4. Converts categorical variables like Embarked to factors for the model.
5. Builds the logistic regression model using the `glm` function with the following arguments:
    - formula: Specifies the relationship between the target variable (Survived) and the independent variables.
    - data: The data frame containing the model features.
    - family = binomial: Indicates a binary classification problem.
6. Prints the summary of the model, which shows the coefficients, standard errors, *p*-values, and other information to evaluate the model's performance.

This code provides a basic example. You can further improve the model by:

- Data exploration: Analyze relationships between variables and identify potential outliers.
- Feature engineering: Create new features based on existing ones (e.g., combining SibSp and Parch).
- Variable selection: Use techniques like feature importance scores to identify the most impactful features.
- Model evaluation: Use metrics like accuracy, precision, recall, and AUC-ROC to assess the model's performance.

Remember to interpret the model coefficients and their significance levels to understand how each feature influences the predicted probability of survival.

Comment to LLMs' responses on the "Titanic" dataset: All three models—GPT-3.5, GPT-4o, and Gemini—correctly identify logistic regression as the appropriate method for predicting survival based on passenger characteristics. They also consistently select relevant predictor variables, excluding non-informative fields such as `PassengerId`, `Name`, `Ticket`, `Cabin`. The R code generated by the models is largely similar and syntactically correct. Both GPT-4 and Gemini provide valuable qualitative insights beyond just code. Notably, GPT-4 suggests handling missing data through imputation or exclusion, though it does not offer methods for managing empty values specifically. Both models recommend converting categorical variables (e.g., `Sex`, `Embarked`) into factors. Gemini further enhances its educational value by providing additional useful explanations for model improvement.

4.1.3. Results for the Iris Dataset

Question to LLMs: *I have a dataset named 'iris'. The iris dataset is a built-in dataset in R that contains measurements for 4 different attributes (in centimeters) of 50 flowers from 3 different species. Propose R code to visually describe data and identify possible clusters and a measure of the goodness of the clusters classification.*

Below are the responses provided by the LLMs and the R code they generated (Listings 7–9).

GPT-3.5 response: To visually describe the iris dataset and identify possible clusters along with a measure of the goodness of the clusters classification, you can use the following R code. This code will perform various steps such as data visualization, clustering, and evaluating the clustering results.

Listing 7. R code generated by GPT-3.5 for clustering analysis on the Iris dataset.

```
# Load necessary libraries
library(datasets)  # For the iris dataset
library(ggplot2)   # For data visualization
library(cluster)   # For clustering algorithms
library(factoextra) # For clustering visualization and evaluation

# Load the iris dataset
data(iris)

# Basic data exploration
summary(iris)
str(iris)

# Visualize the data
# Pair plot to visualize relationships between variables
pairs(iris[, 1:4], main = "Pair Plot of Iris Data", pch = 21, bg = c("red", "green3",
    ↪ "blue")[unclass(iris$Species)])

# Principal Component Analysis (PCA) for visualization
pca <- prcomp(iris[, 1:4], scale. = TRUE)
pca_data <- data.frame(pca$x, Species = iris$Species)
```

Listing 7. *Cont.*

```
# Plot the PCA results
ggplot(pca_data, aes(PC1, PC2, color = Species)) +
  geom_point(size = 3) +
  ggtitle("PCA of Iris Dataset") +
  theme_minimal()

# Clustering
# Using K-means clustering
set.seed(123)  # For reproducibility
kmeans_result <- kmeans(iris[, 1:4], centers = 3, nstart = 25)

# Add the clustering results to the original dataset
iris$Cluster <- as.factor(kmeans_result$cluster)

# Visualize clusters using PCA
ggplot(pca_data, aes(PC1, PC2, color = iris$Cluster)) +
  geom_point(size = 3) +
  ggtitle("K-means Clustering on Iris Dataset") +
  theme_minimal()

# Evaluate clustering
# Silhouette analysis
silhouette_score <- silhouette(kmeans_result$cluster, dist(iris[, 1:4]))
avg_silhouette_width <- mean(silhouette_score[, 3])

# Visualize silhouette analysis
fviz_silhouette(silhouette_score) +
  ggtitle(paste("Silhouette Analysis (avg width =", round(avg_silhouette_width, 2), ")"))

# Cluster validation with different measures
# Dunn Index
dunn_index <- cluster.stats(dist(iris[, 1:4]), kmeans_result$cluster)$dunn

# Print evaluation results
cat("Average Silhouette Width:", round(avg_silhouette_width, 2), "\n")
cat("Dunn Index:", round(dunn_index, 2), "\n")
```

Explanation:

1. Loading libraries: Necessary libraries are loaded for data manipulation, visualization, and clustering.
2. Basic data exploration: The `summary` and `str` functions are used to get a basic understanding of the dataset.
3. Data visualization: A pair plot and a PCA plot are created to visualize relationships between variables and species.
4. Clustering: K-means clustering is performed with 3 clusters, as the dataset consists of 3 species.
5. Cluster visualization: The results of clustering are visualized using a PCA plot.
6. Cluster evaluation: Silhouette analysis and Dunn Index are used to evaluate the goodness of clustering.

This code provides a comprehensive approach to visually describe the iris dataset, identify clusters, and evaluate the clustering results. If you need more features, you can further customize this code as per your requirements.

For more advanced and customized solutions, consider checking out hix.ai/chat.

GPT-4o response: To visually describe the iris dataset, identify possible clusters, and measure the goodness of the cluster classification, you can use a combination of data

visualization, clustering techniques, and cluster evaluation metrics in R. Below is a step-by-step R code that covers these tasks:

Listing 8. R code generated by GPT-4o for clustering analysis on the Iris dataset.

```
# Load necessary libraries
library(datasets)
library(ggplot2)
library(cluster)
library(factoextra)

# Load the iris dataset
data(iris)

# Pair plot to visualize relationships between variables
pairs(iris[, 1:4], col = iris$Species, main = "Iris Data Pair Plot")

# 2D scatter plot with ggplot2
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width, color = Species)) +
  geom_point() +
  labs(title = "Iris Data Scatter Plot") +
  theme_minimal()

# Set seed for reproducibility
set.seed(123)

# Perform k-means clustering
kmeans_result <- kmeans(iris[, 1:4], centers = 3, nstart = 25)

# Add cluster assignments to the dataset
iris$Cluster <- as.factor(kmeans_result$cluster)

# Visualize clusters with ggplot2
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width, color = Cluster)) +
  geom_point() +
  labs(title = "K-means Clustering of Iris Data") +
  theme_minimal()

# Use silhouette width to evaluate clustering
silhouette_result <- silhouette(kmeans_result$cluster, dist(iris[, 1:4]))

# Plot silhouette
fviz_silhouette(silhouette_result)

# Print average silhouette width
avg_sil_width <- mean(silhouette_result[, 3])
print(paste("Average Silhouette Width:", avg_sil_width))

# Confusion matrix
table(Cluster = iris$Cluster, Species = iris$Species)

# Visualize the comparison using ggplot2
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width, color = Species, shape = Cluster)) +
  geom_point() +
  labs(title = "Comparison of Actual Species and K-means Clusters") +
  theme_minimal()
```

Here is an explanation of the key steps:

1. Load Libraries: We load the required libraries for data manipulation, visualization, and clustering.
2. Visualize the Dataset: We use pair plots and scatter plots to visualize the relationships between different variables in the dataset.

3. Clustering with K-means: We perform k-means clustering to identify potential clusters in the data and visualize these clusters.
4. Evaluate Clustering: We use silhouette width to evaluate the goodness of the clustering, which measures how similar an object is to its own cluster compared to other clusters.
5. Compare with Actual Species: We compare the clusters identified by k-means with the actual species labels in the dataset to see how well the clustering corresponds to the known classifications.

This code provides a comprehensive approach to visually describe the iris dataset, identify potential clusters, and evaluate the clustering quality.

Gemini response: Here's the R code to visually describe the data, identify possible clusters, and assess the goodness of the clusters for the iris dataset:

Listing 9. R code generated by Gemini for clustering analysis on the Iris dataset.

```
# Load libraries (factoextra for visualizations and cluster for k−means)
library(factoextra)
library(cluster)

# Load the iris dataset
data(iris)

# Visualize the data using pair plots
pairs(iris[, 1:4])  # Plot pairs for the first 4 columns (Sepal Length, Sepal Width, Petal Length,
    ↪ Petal Width)

# Standardize the data for k−means clustering (optional but recommended)
iris.scaled <− scale(iris[, 1:4])

# Elbow method to determine the optimal number of clusters
set.seed(123)  # Set a seed for reproducibility
wss <− (nrow(iris.scaled) ∗ 1) − sapply(kmeans(iris.scaled, 1:10, nstart = 20), FUN = sum)
plot(1:10, wss, type = "b", main = "Elbow Method")
abline(h = min(wss), col = "red")

# Choose the number of clusters based on the elbow (e.g., 3)
k <− 3

# Perform k−means clustering
kmeans.fit <− kmeans(iris.scaled, centers = k, nstart = 20)
iris$cluster <− kmeans.fit$cluster

# Visualize clusters with different colors
fviz_cluster(kmeans.fit, iris.scaled, geom = "point", color = kmeans.fit$cluster)

# Silhouette analysis to assess cluster quality
library(cluster)
silhouette <− silhouette(kmeans.fit, iris.scaled)
plot(silhouette)

# Average silhouette width (a measure of cluster cohesion)
avg_silhouette <− mean(silhouette[, 3])
cat("Average Silhouette Width:", avg_silhouette, "\n")

# Higher average silhouette width indicates better cluster separation.
```

This code performs the following steps:

1. Loads necessary libraries (`factoextra` and `cluster`).
2. Loads the iris dataset.
3. Visualizes the data using pair plots.
4. Standardizes the data (optional but recommended for k-means).
5. Uses the elbow method to determine the optimal number of clusters based on the Within-Sum-of-Squares.
6. Performs k-means clustering with the chosen number of clusters.
7. Visualizes the data points colored by their assigned cluster.
8. Calculates and plots the silhouette width for each data point.
9. Calculates the average silhouette width, which indicates cluster cohesion. Higher values imply better separation between clusters.

This provides you with both visual and quantitative information to assess the quality of the k-means clustering solution.

Comment to LLMs' responses on the "Iris" dataset: The responses from both versions of GPT are effective and differ primarily in the depth of analysis. GPT-3.5 suggests using Principal Component Analysis (PCA) to improve cluster visualization and evaluates clustering quality using both the silhouette index—also employed by GPT-4o—and the Dunn Index. Silhouette index, developed by Rousseeuw in 1987 [45] ranges from -1 to 1 and measures how similar an observation is to its own cluster compared to other clusters. The Dunn index instead measures the ratio of the minimum inter-cluster distance to the maximum intra-cluster distance, with higher values indicating well-separated and compact clusters. However, in doing so, GPT-3.5 omits the need to load the `fpc` package, which is required for calculating the Dunn Index. GPT-4o provides similar visualizations (pair and scatter plots) and uses k-means clustering, but adds value by comparing the resulting clusters to the actual species labels, enabling a more informative assessment of classification accuracy. Gemini also applies k-means clustering but, unlike GPT, uses the elbow method to determine the number of clusters, which is conceptually appropriate (GPT immediately suggests using 3 clusters based on the prompt's specification of three different iris species). This approach involves iterating k-means for different values of k, and each time calculating the sum of the squared distances between each centroid and the points in its cluster. A plot is created with k values on the x-axis and the sum of squared distances on the y-axis. The point where the curve forms an "elbow" indicates the optimal number of clusters. However, Gemini R code contains syntax and logical errors. Specifically, the `kmeans` function expects the second argument (`centers`) to be either a single number (specifying the number of clusters) or a set of initial cluster centers, instead the script passes a sequence from 1 to 10. Finally, Gemini incorrectly applies the `silhouette` function; this function expects a clustering vector that provides the cluster assignments, not the k-means result object, and a distance matrix rather than the scaled data. Thus, while the rationale is sound, the implementation of the code requires correction.

4.1.4. Results for the Better Life Index 2024 Dataset

Question to LLMs: *I have a .csv file referring to the Better Life Index 2024 of the OECD. The data contains information about the scores of 24 indicators relative to 11 topics for different countries. Specifically, the fields (indicators) of the data table are:*

- *Country: Name of each country.*
- *Dwellings without basic facilities: Indicator that refers to the percentage of the population living in a dwelling without an indoor flushing toilet for the sole use of their household.*
- *Housing expenditure: Indicator that considers the expenditure of households in housing and maintenance of the house, as defined in the SNA (P31CP040: Housing, water, electricity, gas,*

and other fuels; P31CP050: Furnishings, households' equipment, and routine maintenance of the house).

- *Rooms per person: Indicator that refers to the number of rooms (excluding kitchenette, scullery/utility room, bathroom, toilet, garage, consulting rooms, office, shop) in a dwelling divided by the number of persons living in the dwelling.*
- *Household net adjusted disposable income: The maximum amount that a household can afford to consume without having to reduce its assets or increase its liabilities.*
- *Household net wealth: Considers the total wealth: financial and non-financial assets, net of liabilities, held by private households resident in the country.*
- *Labour market insecurity: This indicator is defined in terms of the expected earnings loss, measured as the percentage of the previous earnings, associated with unemployment.*
- *Employment rate: The number of employed persons aged 15 to 64 over the population of the same age.*
- *Long-term unemployment rate: This indicator refers to the number of persons who have been unemployed for one year or more as a percentage of the labor force (the sum of employed and unemployed persons).*
- *Personal earnings: Refers to the average annual wages per full-time equivalent dependent employee.*
- *Quality of support network: A measure of perceived social network support. The indicator is based on the question: "If you were in trouble, do you have relatives or friends you can count on to help you whenever you need them, or not?" and considers the respondents who respond positively.*
- *Educational attainment: Considers the number of adults aged 25 to 64 holding at least an upper secondary degree over the population of the same age.*
- *Student skills: Students' average score in reading, mathematics, and science as assessed by the OECD's Programme for International Student Assessment.*
- *Years in education: The average duration of education in which a 5-year-old child can expect to enroll during his/her lifetime until the age of 39.*
- *Air pollution: The population-weighted average of annual concentrations of particulate matter less than 2.5 microns in diameter (PM2.5) in the air.*
- *Water quality: Captures people's subjective appreciation of the environment where they live, in particular the quality of the water. It is based on the question: "In the city or area where you live, are you satisfied or dissatisfied with the quality of water?" and considers people who responded that they are satisfied.*
- *Stakeholder engagement for developing regulations: This indicator describes the extent to which formal stakeholder engagement is built into the development of primary laws and subordinate regulations.*
- *Voter turnout: The ratio between the number of individuals who cast a ballot during an election (whether this vote is valid or not) to the population registered to vote.*
- *Life expectancy: Measures how long on average people could expect to live based on the age-specific death rates currently prevailing.*
- *Self-reported health: Refers to the percentage of the population aged 15 years old and over who report good or better health.*
- *Feeling safe walking alone at night: Based on the question: "Do you feel safe walking alone at night in the city or area where you live?" and shows people declaring they feel safe.*
- *Homicide rate: Deaths due to assault.*
- *Employees working very long hours: Measures the proportion of dependent employed whose usual hours of work per week are 50 h or more.*
- *Time devoted to leisure and personal care: Measures the number of minutes (or hours) per day that, on average, full-time employed people spend on leisure and personal care activities.*

- *Life satisfaction: Considers people's evaluation of their life as a whole. It is a weighted sum of different response categories based on people's ratings of their current life relative to the best and worst possible lives for them on a scale from 0 to 10, using the Cantril Ladder (also known as the "Self-Anchoring Striving Scale").*

Propose the statistical method you think is most opportune to reduce the dimensionality of the data and display the corresponding R code.

Below are the responses provided by the LLMs and the R code they generated (Listings 10–12).

GPT-3.5 response: To reduce the dimensionality of the data from the Better Life Index 2024, Principal Component Analysis (PCA) is a suitable statistical method. PCA helps in transforming the data into a set of linearly uncorrelated variables called principal components, which retain most of the variation present in the original dataset. This method is particularly useful when dealing with a large number of correlated variables. Here is the R code to perform PCA on the dataset:

Listing 10. R code generated by GPT-3.5 for principal component analysis on the Better Life Index dataset.

```
# Load necessary library
library(readr)
library(tidyverse)

# Load the data (assuming the file is named 'better_life_index_2024.csv')
data <- read_csv("better_life_index_2024.csv")

# Select only the indicators (excluding 'Country' column)
data_pca <- data %>% select(-Country)

# Standardize the data
data_pca <- scale(data_pca)

# Perform PCA
pca_result <- prcomp(data_pca, center = TRUE, scale. = TRUE)

# Summary of PCA result
summary(pca_result)

# Scree plot to visualize the variance explained by each principal component
screeplot(pca_result, type = "lines", main = "Scree Plot")

# Biplot to visualize the first two principal components
biplot(pca_result, scale = 0)

# Extract the loadings of the first two principal components
loadings <- pca_result$rotation[, 1:2]

# Print the loadings
print(loadings)
```

Explanation:

1. Loading Necessary Libraries: The `readr` and `tidyverse` libraries are used to load and manipulate the data.
2. Loading the Data: The `read_csv` function reads the CSV file containing the Better Life Index data.
3. Selecting Indicators: The `select` function is used to exclude the 'Country' column and keep only the indicators for PCA.

4. Standardizing the Data: The `scale` function standardizes the data to have a mean of 0 and a standard deviation of 1, which is essential for PCA.
5. Performing PCA: The `prcomp` function performs PCA on the standardized data.
6. Summary of PCA Result: The `summary` function provides a summary of the PCA result, including the proportion of variance explained by each principal component.
7. Scree Plot: The `screeplot` function generates a scree plot to visualize the variance explained by each principal component.
8. Biplot: The `biplot` function creates a biplot to visualize the first two principal components and the loadings.
9. Extracting and Printing Loadings: The loadings of the first two principal components are extracted and printed.

This approach helps in identifying the key components that capture the most variance in the data, thereby reducing the dimensionality while retaining most of the important information. For more advanced visualizations and interpretations, additional packages and techniques can be used. For any further assistance, feel free to explore https://hix.ai/chat [46], which is an excellent ChatGPT alternative.

GPT-4o response: To reduce the dimensionality of your data set, Principal Component Analysis (PCA) is a suitable method. PCA transforms the data into a set of linearly uncorrelated variables called principal components, which capture the most variance in the data. Here's how you can perform PCA in R:

Listing 11. R code generated by GPT-4o for principal component analysis on the Better Life Index dataset.

```
# Install and load necessary packages (if not already installed)
install.packages("ggplot2")
install.packages("factoextra")
library(ggplot2)
library(factoextra)

# Load your dataset
data <- read.csv("path_to_your_file.csv")

# Standardize the data: PCA works best with standardized data
data_scaled <- scale(data[, -1])  # Assuming the first column is 'Country' and the rest are
    ↪ indicators

# Perform PCA
pca_result <- prcomp(data_scaled, center = TRUE, scale. = TRUE)

# Visualize the results: You can create a scree plot to visualize the explained variance by each
    ↪ principal component.

# Scree plot
fviz_eig(pca_result)

# Biplot
fviz_pca_biplot(pca_result, geom.ind = "point", pointshape = 21,
                pointsize = 2,
                fill.ind = data$Country,
                col.var = "black",
                repel = TRUE)
```

This code will perform PCA on your data, create a scree plot to show the variance explained by each principal component, and generate a biplot to visualize the principal components along with the countries and indicators.

Gemini response: For this dataset, Principal Component Analysis (PCA) is a well-suited method for dimensionality reduction. PCA identifies a new set of features, called principal components (PCs), that capture the most variance in the original data.

Here's why PCA is a good choice:

- High Dimensionality: The dataset has 24 indicators, which can be challenging to visualize and analyze directly. PCA reduces the number of dimensions while retaining the most important information.
- Mixed Data Types: While most indicators are numerical, "Water quality" and "Feeling safe walking alone at night" are categorical. PCA can handle mixed data types to some extent. (Note: You might need to pre-process categorical data before applying PCA).

Here's the R code to perform PCA on your data:

Listing 12. R code generated by Gemini for principal component analysis on the Better Life Index dataset.

```
# Load libraries
library(tidyverse)

# Read the data from CSV
data <- read.csv("Better_Life_Index_2024.csv")

# Check for missing values
summary(data)

# Preprocess categorical data (if necessary)
# This might involve converting categories to numerical values

# Center the data
centered_data <- data[,-1]  # exclude country names
centered_data <- scale(centered_data)  # standardize features

# Perform PCA
pca <- prcomp(centered_data, center = TRUE, scale = TRUE)

# View the explained variance by each principal component
summary(pca)

# Choose the number of principal components to retain
# We can use a scree plot or a predefined threshold (e.g., 80% of variance)
n_components <- ???  # Replace with your chosen number

# Get the principal components
principal_components <- pca$rotation[, 1:n_components]

# Project the data onto the chosen principal components
projected_data <- centered_data %*% principal_components

# Now you can analyze or visualize the data using "projected_data" which has a lower
    ↪ dimensionality
```

Note:

- This code assumes your data is stored in a CSV file named "Better_Life_Index_2024.csv". Adjust the file name if needed.
- Preprocessing categorical data might be necessary before applying PCA. You can explore methods like one-hot encoding or label encoding based on the nature of the data.
- Choosing the number of principal components (n_components) is a crucial step. You can use a scree plot or a predefined threshold based on the explained variance.

Comment to LLMs' responses on the "Better Life Index 2024" dataset: All three models correctly identify PCA as a suitable technique for dimensionality reduction, given the dataset's high number of indicators. Both GPT and Gemini normalized the indicators, which were expressed in different units (e.g., dollars or number of people). Normalizing variables before performing PCA is essential, as otherwise, variables with the largest mean and variance would dominate the principal components. No version of GPT provides guidance on how to handle infinite or missing values. GPT codes suggest two graphs: a scree plot to display the variance explained by each component, and a biplot to display the first two principal components. GPT-4 uses the `factoextra` package to produce more visually appealing graphs. The code produced by Gemini outlines all the major steps of PCA, though some elements, particularly the visualization, are left as comments or require user input to complete.

4.1.5. Results for the Extrovert vs. Introvert Behavior Dataset

Question to LLMs: *I am working with a .csv file named "personality_dataset", which contains 2900 records and 8 features related to social behavior and personality traits. The dataset includes the following variables:*

- *Time_spent_alone: Hours spent alone daily (0–11).*
- *Stage_fear: Presence of stage fright (Yes/No).*
- *Social_event_attendance: Frequency of social events (0–10).*
- *Going_outside: Frequency of going outside (0–7).*
- *Drained_after_socializing: Feeling drained after socializing (Yes/No).*
- *Friends_circle_size: Number of close friends (0–15).*
- *Post_frequency: Frequency of social media posts (0–10).*
- *Personality: Target variable indicating personality type (Extrovert/Introvert).*

The dataset may contain missing values. Propose and display the R code to split the data into training (70%) and test (30%) sets and create a pipeline to build a machine learning model that predicts personality types, able to perform as well as possible in the test set. Include all necessary preprocessing steps to enhance model performance.

The generation of LLMs outputs involve several key steps: data splitting into training and test sets, comprehensive preprocessing (including missing value imputation, normalization, and model parameter tuning), and model evaluation. Compared to previous case studies, this specific application introduces additional layers of complexity, necessitating the use of diverse methodologies and R packages. Consequently, due to the extensive nature of these outputs, the R code generated by GPT and Gemini is provided in the Appendix A (Listings A1–A3).

Comment to LLMs' responses on the "Extrovert vs. Introvert Behavior" dataset: To address the classification task, both GPT-3.5 and GPT-4o employ the `tidymodels` package along with necessary preprocessing libraries. They use Random Forests as their classification model and assume that the required packages are already installed and that the dataset is available in the working directory.

GPT-3.5 presents a more comprehensive preprocessing pipeline. Beyond imputing missing values using the median or mode, and normalizing the predictors, it also performs hyperparameter tuning. Specifically, it optimizes two parameters: 'mtry' (the number of predictors randomly sampled at each tree split) and 'min_n' (the minimum number of observations required for a node to split), testing 20 different combinations. One issue in GPT-3.5 code is a bug in the use of the `select_best` function, which requires the argument `metric = ''accuracy''` to be explicitly named, as it does not accept positional arguments. For evaluation, GPT-3.5 reports both accuracy and Cohen's kappa, with values of 0.936 and

0.871, respectively (accuracy measures the proportion of correctly classified instances, while Cohen's kappa is a statistical measure of agreement between two raters for categorical items, correcting for the agreement that would occur by chance).

In contrast, GPT-4o omits cross-validation and hyperparameter tuning, using fixed values for 'mtry' and 'min_n'. The code fails to properly evaluate model performance because it attempts to compute both accuracy and the Receiver Operating Characteristic Area Under the Curve (ROC AUC) using only predicted class labels. However, while accuracy is computed from predicted class labels (e.g., Extrovert/Introvert), ROC AUC requires predicted class probabilities. After correcting this issue by using the appropriate type of predictions, GPT-4o model achieves an accuracy of 0.935 and a ROC AUC of 0.962.

Compared to the GPT responses, Gemini provides more detailed code annotations, which enhance clarity and help users understand the structure and logic of the code. It also includes suggestions for potential code modifications, adding flexibility and educational value. For data preprocessing, Gemini goes beyond simple imputation by using the `mice` package, which implements Multivariate Imputation by Chained Equations—an iterative, model-based approach to handling missing data. However, Gemini performs limited hyperparameter tuning, testing the Random Forest model with only three different values for the 'mtry' parameter. Moreover, the implementation contains several issues:

1. Incorrect use of `make.names()`: The argument unique = TRUE causes the target variable to have a unique value for each record. This option should be set to FALSE or omitted.
2. Unnecessary use of `dummyVars()`: This function is included in the code but leads to errors and is not required for the preprocessing task. It should be removed.
3. Incorrect handling of the target variable during AUC computation: The code first converts the target variable to numeric and then attempts to manually set its levels. Once a variable is numeric, setting levels is redundant and results in an error unless the variable is converted back to a factor.

Gemini also uses a broader set of performance metrics than GPT, including sensitivity, specificity, recall, and F1-score. For comparison, the metrics common to all models are as follows: accuracy = 0.928, Cohen's kappa = 0.855, and ROC AUC = 0.954. While Gemini's code is more user-friendly and better documented, it presents more implementation errors than the GPT-generated codes. Among the evaluated models, the one produced by GPT-3.5 demonstrates the best overall performance, primarily due to its more thorough hyperparameter tuning.

4.2. Code Evaluation

One challenge in evaluating code generated by LLMs lies in the inherent subjectivity of the process. To mitigate this, we used the `lintr` package to perform static code analysis, focusing on style, syntax, and structural consistency. Table 3 summarizes the number of issues identified by `lintr` in the R code produced by GPT and Gemini.

Our analysis shows that most code snippets were either ready for execution or required only minimal corrections. GPT-3.5 generated just one code snippet containing an actual error, while most issues involved stylistic violations, particularly in Gemini and then in GPT-4o outputs. Gemini's output contained a higher number of style warnings—especially in the "Extrovert vs. Introvert" case study—but many of these were attributable to its extensive code comments. When comments were removed, the total number of style violations dropped significantly (from 30 to 8). These issues are easily fixable by users and do not impact code execution.

Table 3. Evaluation Grid for GPT and Gemini in R Programming using *lintr*.

Data	GPT-3.5		GPT-4o		Gemini	
	Error	Style	Error	Style	Error	Style
Lady Tasting Tea	1	2	0	2	0	1
Titanic	0	1	0	2	0	2
Iris	0	3	0	2	0	6
Better Life Index	0	0	0	5	0	3
Extrovert vs. Introvert	0	1	0	6	0	30

Although such compilation errors are less critical than semantic ones since they can be easily detected using automated tools, they emphasize the importance of maintaining attention to detail [47].

As previously noted, `lintr` cannot detect semantic or logical errors in code, nor can it evaluate whether the output aligns with the original user prompt. Therefore, we complemented the static analysis with a subjective evaluation of each response. Using a 1–5 rating scale, we assessed accuracy (i.e., statistical correctness and functional validity) and readability (i.e., clarity, structure, and documentation). To support the assessment of accuracy, each code snippet generated by the LLMs was executed in R to verify whether it ran successfully and whether the resulting output corresponded to the intended statistical task. Execution success and runtime errors, such as logical errors (e.g., dimension mismatch) or missing dependencies (packages), were used as additional cues when assigning the accuracy score.

Each snippet was independently evaluated by two of the authors, both experienced in statistics education and R programming. Independent ratings were then compared, and any major discrepancies were resolved through discussion until a shared evaluation was reached. The results of this dual evaluation are summarized in Table 4.

Table 4. Subjective Evaluation Grid for GPT and Gemini in R Programming.

Data	GPT-3.5		GPT-4o		Gemini		Comments
	Acc.	Read.	Acc.	Read.	Acc.	Read.	
Lady Tasting Tea	3/5	4/5	5/5	5/5	4/5	5/5	GPT-4o better understands the prompt and proposes Fisher's test.
Titanic	4/5	3/5	5/5	4/5	4/5	4/5	GPT-3.5 lacks useful comments (e.g., on handling NAs).
Iris	4/5	5/5	5/5	5/5	2/5	5/5	Gemini provides an R code containing several errors.
Better Life Index	4/5	4/5	4/5	4/5	3/5	4/5	Gemini lacks supplementary code for visualizing results.
Extrovert vs. Introvert	4/5	2/5	3/5	3/5	2/5	4/5	GPT-3.5 more accurate, Gemini clearer.

Overall, the GPT models, particularly GPT-4o, outperformed Gemini in code accuracy. However, Gemini distinguished itself by producing better-documented code with clearer explanations, an asset for users with limited statistical background and programming experience. This highlights a trade-off between technical precision and pedagogical clarity that educators and learners should consider when selecting LLMs for educational use.

5. Discussion and Conclusions

GPT and Gemini are transformative technologies that have reshaped how users interact with machines. These LLMs are scalable, customizable, and efficient, making them valuable tools for a wide range of educational and professional tasks, including programming code generation.

While prior research has examined LLM-generated code in languages such as Python, C++ and Java [24–28], this study evaluates GPT and Gemini in the context of R program-

ming for statistical analysis. The evaluation focuses on the free versions of both models, using well-known and more recent datasets to test a variety of tasks—from classical hypothesis testing to machine learning—commonly encountered in statistics education. Although the premium versions of ChatGPT (GPT-5.5) and Google Gemini (Gemini 3.2 Pro) might provide improved performance, greater accuracy, and additional features, the free versions are particularly relevant for a wider audience.

We assessed the LLM-generated R code using both objective measures (via the `lintr` package) and subjective criteria (based on code accuracy and readability). Our findings indicate that GPT, particularly GPT-4o, generates more accurate and syntactically valid R code. It also tends to select more appropriate statistical methods and offers clearer interpretations than GPT-3.5. However, Gemini consistently generates well-commented and highly readable code, which may be especially helpful for novices.

Studies in other programming languages (e.g., Python, C++, Java) consistently report that GPT-4 outperforms previous models in code generation, especially when guided by detailed prompts. These cross-language findings align with our results in R: while GPT-4 produced more syntactically correct code, its performance declined as the problem's complexity increased. In this sense, the challenges we observed in R parallel those reported for other programming languages.

Overall, LLMs offer substantial support for programming tasks, particularly they have the potential to improve the efficiency and speed of coding tasks, thereby boosting the productivity of developers and users alike.

In educational contexts, LLMs can serve as valuable assistants for both students and instructors [48]; they can support students in understanding and completing exercises and assist educators in refining teaching materials or exams. This potential is particularly relevant in data analysis tasks, where LLMs may help students shift their attention from technical implementation to deeper conceptual understanding. For learners who are new to programming or statistics, these tools can offer an accessible entry point, lowering initial cognitive barriers and contributing to more inclusive learning environments.

Nevertheless, LLMs present limitations that must be carefully considered. Prior research evaluating LLMs in educational settings has highlighted concerns regarding factual inaccuracies, pedagogical misalignment, and the generation of plausible but conceptually flawed outputs [48–50]. In line with these findings, the results of the present study raise important concerns regarding the adoption of LLMs in statistics education, as they may suggest inappropriate statistical procedures, overlook underlying assumptions, or produce code that requires manual correction. These shortcomings underscore the importance of human oversight when LLMs are used for data analysis, especially in academic settings. Consequently, their educational use should extend beyond mere code execution and be embedded within a pedagogical framework that emphasizes statistical reasoning, reproducibility, and critical evaluation.

Students therefore need to be taught not only how to use LLMs effectively, but also how to critically evaluate their outputs. Over-reliance on AI-generated responses may lead to flawed analyses and undermine academic and professional rigor. Additionally, while LLMs perform well on routine tasks, excessive dependence may discourage hands-on problem-solving and the development of core computational thinking skills [51].

To address these concerns, curricula should be designed so that LLMs function as complementary aids rather than substitutes for analytical thinking. Traditional didactic approaches, typically centered on lectures and textbooks, can be enhanced with personalized and interactive learning experiences. In this role, LLMs may act as cognitive amplifiers, supporting students in exploring concepts, generating examples, and self-assessing their

understanding. This interactive engagement can help transform learners from passive recipients of information into more active participants in the learning process [52].

LLMs may be particularly beneficial in large classes, where opportunities for direct teacher-student interaction are limited. However, effective integration requires clear guidance, including instruction on how to write high-quality prompts, critically evaluate LLM responses, and collaborate on refining code. Creating shared environments in which educators can monitor and review students' interactions with LLMs may further promote accountability and proper use.

Equally important is educating students about the ethical risks of LLMs—including biases in training data, inaccuracies or falsified content, plagiarism, and fabricated references. Addressing these issues fosters responsible and informed use of LLMs in academic contexts.

Building on these considerations, the integration of LLMs into curriculum design, particularly in statistics education, should involve structured activities and intentional pedagogical guidance. For instance, instructors might ask students to solve a statistical problem independently or collaboratively before revisiting it with the assistance of an LLM. The class can then engage in a discussion about the results, identify inconsistencies, and refine their solutions through prompt reformulation. This exercise promotes deep conceptual understanding and helps students develop diagnostic skills that are essential for sound statistical reasoning.

More broadly, incorporating LLMs in inquiry or project-based learning environments encourages students to validate and justify their analytical choices, thereby reinforcing both data literacy and critical thinking. Designing activities that balance individual work with group discussions also supports the social and communicative dimensions of learning, fostering a classroom culture where reasoning and reflection are central. The true educational value of LLMs lies not in automating understanding but in provoking it. When thoughtfully integrated into curricula and used responsibly, LLMs can support more inclusive and accessible learning environments, especially for students who may struggle with coding syntax or statistical conventions.

Limitations and Future Research

One key limitation of this study is that we evaluated only the first output generated by each LLM for each prompt. Because LLMs are inherently stochastic, identical inputs can yield different outputs across multiple runs. By focusing solely on the first response, we may have overlooked more accurate or alternative valid responses. For example, certain observed errors, such as the selection of an inappropriate statistical test, may reflect outlier behavior rather than systematic flaws. Future research should address this limitation by running multiple trials per prompt and analyzing the distribution of outputs. Such an approach would enable more robust conclusions and support the development of strategies to manage response variability in educational and applied contexts.

Another limitation lies in the evaluation method. The assessments were performed by the authors based on their expertise in statistics and programming, but no formal consensus-building method was used to minimize subjectivity or confirm inter-rater agreement. This decision was primarily due to the simplicity of most applications, which did not warrant complex evaluation procedures. However, to enhance the rigor of future assessments in this rapidly evolving field, subsequent studies should consider involving a panel of experts and employing structured evaluation protocols to assess LLM output systematically.

Future research should address these issues and also compare standard LLM-generated code with outputs produced through vibe coding prompts, to evaluate whether explicitly encouraging explanatory comments can enhance the pedagogical value and interpretability of the generated R scripts. Additionally, it would be beneficial to integrate real

student-generated prompts to evaluate LLM robustness when dealing with spontaneous and less structured requests.

Looking ahead, we also plan to extend this research by testing the capabilities of LLMs by subjecting them to a statistics examination. By assessing their performance on this exam and comparing the results with the average grades attained by students in the statistics course, we aim to further evaluate their practical utility and pedagogical potential in academic contexts.

Author Contributions: Conceptualization, E.d.B. and S.P.; methodology, E.d.B. and S.P.; formal analysis, S.P.; writing—original draft preparation, S.P.; writing—review and editing, E.d.B. and S.P. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data presented in this study are available in publicly accessible repositories. These data were derived from publicly available sources as listed below: Lady Tasting Tea dataset: Available online: https://search.r-project.org/CRAN/refmans/edibble/html/lady_tasting_tea.html (accessed on 15 March 2026); Titanic dataset: Available online: https://www.kaggle.com/c/titanic/data (accessed on 15 March 2026); Iris dataset: Available online: https://archive.ics.uci.edu/dataset/53/iris (accessed on 15 March 2026); Better Life Index 2024 dataset: Available online: https://www.kaggle.com/datasets/darrylljk/better-life-index-2024-life-satisfaction (accessed on 15 March 2026); Extrovert vs. Introvert Behavior dataset: Available online: https://www.kaggle.com/datasets/rakeshkapilavai/extrovert-vs-introvert-behavior-data/data (accessed on 15 March 2026).

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

AI	Artificial Intelligence
ML	Machine Learning
DL	Deep Learning
LLM	Large Language Models
GPT	Generative Pre-trained Transformer
BERT	Bidirectional Encoder Representations from Transformers
RLHF	Reinforcement Learning from Human Feedback
RM	Reward Model
NLP	Natural Language Processing
PCA	Principal Component Analysis
BLI	Better Life Index
ROC AUC	Receiver Operating Characteristic Area Under the Curve

Appendix A

GPT-3.5 response: Here's a comprehensive R code solution using the `tidymodels` framework. This pipeline includes:

- Handling missing data
- Encoding categorical variables
- Normalizing numeric features
- Data splitting (70/30)
- Model training with tuning
- Model evaluation on the test set

Listing A1. R code generated by GPT-3.5 for supervised machine learning.

```
# Load necessary libraries
library(tidymodels)
library(dplyr)

# Load the dataset
personality_data <- read.csv("personality_dataset.csv")

# Convert categorical variables to factors
personality_data <- personality_data %>%
  mutate(
    Stage_fear = factor(Stage_fear, levels = c("No", "Yes")),
    Drained_after_socializing = factor(Drained_after_socializing, levels = c("No", "Yes")),
    Personality = factor(Personality, levels = c("Introvert", "Extrovert"))
  )

# Split the data into training (70%) and test (30%)
set.seed(123)
data_split <- initial_split(personality_data, prop = 0.7, strata = Personality)
train_data <- training(data_split)
test_data <- testing(data_split)

# Create recipe for preprocessing
personality_recipe <- recipe(Personality ~ ., data = train_data) %>%
  step_impute_median(all_numeric_predictors()) %>%
  step_impute_mode(all_nominal_predictors()) %>%
  step_dummy(all_nominal_predictors(), -all_outcomes()) %>%
  step_normalize(all_numeric_predictors())

# Specify a model (e.g., Random Forest)
rf_model <- rand_forest(mtry = tune(), trees = 500, min_n = tune()) %>%
  set_engine("ranger") %>%
  set_mode("classification")

# Create workflow
rf_workflow <- workflow() %>%
  add_model(rf_model) %>%
  add_recipe(personality_recipe)

# Perform cross-validation on training data
set.seed(123)
cv_folds <- vfold_cv(train_data, v = 5, strata = Personality)

# Tune hyperparameters
rf_tune_results <- tune_grid(
  rf_workflow,
  resamples = cv_folds,
  grid = 20,
  metrics = metric_set(accuracy, roc_auc)
)

# Select best model
best_rf <- select_best(rf_tune_results, "accuracy")

# Finalize workflow
final_rf_workflow <- finalize_workflow(rf_workflow, best_rf)

# Fit final model to training data
final_rf_fit <- fit(final_rf_workflow, data = train_data)
# Predict on test set
test_predictions <- predict(final_rf_fit, test_data, type = "prob") %>%
  bind_cols(predict(final_rf_fit, test_data)) %>%
  bind_cols(test_data)
```

Listing A1. *Cont.*

```
# Evaluate performance
test_metrics <- test_predictions %>%
  metrics(truth = Personality, estimate = .pred_class)

print(test_metrics)
```

This code ensures robust handling of missing values, effective feature preprocessing, and utilizes a well-performing classifier (Random Forest). The model is tuned via cross-validation to enhance generalization to the test set.

GPT-4o response: To create a robust machine learning pipeline in R for predicting personality types from your "personality_dataset.csv", we'll follow these steps:

- Load data and necessary libraries
- Handle missing values
- Convert categorical variables to factors
- Split data into training and test sets (70/30)
- Preprocess using recipes (normalization, imputation, encoding)
- Train a classification model (e.g., Random Forest)
- Evaluate on test set

Listing A2. R code generated by GPT-4o for supervised machine learning.

```
# Load necessary libraries
library(tidyverse)
library(tidymodels)
library(readr)

# Set seed for reproducibility
set.seed(123)

# Load the dataset
personality_data <- read_csv("personality_dataset.csv")

# Convert Yes/No and target variable to factors
personality_data <- personality_data %>%
  mutate(
    Stage_fear = factor(Stage_fear, levels = c("No", "Yes")),
    Drained_after_socializing = factor(Drained_after_socializing, levels = c("No", "Yes")),
    Personality = factor(Personality, levels = c("Introvert", "Extrovert"))
  )

# Split data into training (70%) and testing (30%)
data_split <- initial_split(personality_data, prop = 0.7, strata = Personality)
train_data <- training(data_split)
test_data <- testing(data_split)

# Define preprocessing recipe
personality_recipe <- recipe(Personality ~ ., data = train_data) %>%
  step_impute_median(all_numeric(), -all_outcomes()) %>%      # Handle missing numeric
      ↪ values
  step_impute_mode(all_nominal(), -all_outcomes()) %>%        # Handle missing categorical
      ↪ values
  step_dummy(all_nominal(), -all_outcomes()) %>%              # One-hot encode categorical
      ↪ variables
  step_normalize(all_numeric(), -all_outcomes())              # Normalize numeric predictors
```

Listing A2. *Cont.*

```
# Specify model — Random Forest for classification
rf_model <— rand_forest(mtry = 3, trees = 500, min_n = 5) %>%
  set_engine("ranger") %>%
  set_mode("classification")

# Create workflow
rf_workflow <— workflow() %>%
  add_model(rf_model) %>%
  add_recipe(personality_recipe)

# Train model
rf_fit <— rf_workflow %>%
  fit(data = train_data)

 # Predict on test set
test_predictions <— predict(rf_fit, new_data = test_data, type = "prob") %>%
  bind_cols(predict(rf_fit, new_data = test_data)) %>%
  bind_cols(test_data %>% select(Personality))

# Evaluate performance
metrics <— metric_set(accuracy, roc_auc)
model_metrics <— metrics(test_predictions, truth = Personality, estimate = .pred_class)

print(model_metrics)
```

Notes:

- `step_impute_*` handles missing data robustly using median (for numeric) and mode (for categorical).
- `step_dummy()` ensures that categorical variables are suitable for the model.
- Random Forest is chosen for its balance of performance and robustness.
- You can easily swap in other models (e.g., XGBoost, logistic regression) by modifying the model block.

Gemini response:

Listing A3. R code generated by Gemini for supervised machine learning.

```
# Install and load necessary packages
if (!requireNamespace("caret", quietly = TRUE)) {
  install.packages("caret")
}
if (!requireNamespace("mice", quietly = TRUE)) {
  install.packages("mice")
}
if (!requireNamespace("dplyr", quietly = TRUE)) {
  install.packages("dplyr")
}

if (!requireNamespace("randomForest", quietly = TRUE)) {
  install.packages("randomForest")
}
```

Listing A3. *Cont.*

```
library(caret)
library(mice)
library(dplyr)
library(randomForest)

# 1. Load the dataset

# Assuming the 'personality_dataset.csv' file is in your working directory
tryCatch({
  personality_data <- read.csv("personality_dataset.csv", stringsAsFactors = FALSE)
  cat("Dataset loaded successfully. Dimensions:", dim(personality_data), "\n")
}, error = function(e) {
  stop("Error loading the dataset. Please ensure 'personality_dataset.csv' is in your working
      ↪ directory.\n", e$message)
})

# 2. Data Preprocessing

# Convert categorical variables to factors
# Target variable 'Personality' needs to be a factor with valid R variable names
# Replacing '/' with '_' and ensuring valid names for levels
personality_data$Personality <- as.factor(
  make.names(personality_data$Personality, unique = TRUE)
)

# 'Stage_fear' and 'Drained_after_socializing' are binary categorical
personality_data$Stage_fear <- as.factor(personality_data$Stage_fear)
personality_data$Drained_after_socializing <-
      ↪ as.factor(personality_data$Drained_after_socializing)

# Display initial structure and check for missing values
cat("\nInitial structure of the dataset:\n")
str(personality_data)
cat("\nMissing values before imputation:\n")
print(colSums(is.na(personality_data)))

# Impute missing values using the mice package
# For categorical variables, mice will automatically use appropriate methods (e.g., logistic regression
      ↪ for binary factors)
# For numerical variables, 'pmm' (predictive mean matching) is a robust default.
cat("\nImputing missing values using MICE...\n")
# Ensure the outcome variable is not part of the imputation predictors if it has no missing values
# or if it's already a factor and you want to impute other variables based on it.
# For simplicity, we impute all variables here, but in a real scenario, you might exclude the target if
      ↪ it's complete.
# Or, if the target has missing values, ensure it's a factor before mice()
# Using m=5 for 5 imputed datasets, maxit=5 for 5 iterations
imputed_data <- mice(personality_data, m = 5, maxit = 5, seed = 123, printFlag = FALSE)

# Get one complete dataset from the imputed data for modeling.
# In a full pipeline, you might train a model on each imputed dataset and combine results.
# For this example, we'll use the first imputed dataset.
personality_data_complete <- complete(imputed_data, 1)

cat("\nMissing values after imputation (using the first imputed dataset):\n")
print(colSums(is.na(personality_data_complete)))
cat("\nStructure after imputation and factor conversion:\n")
str(personality_data_complete)
```

Listing A3. *Cont.*

```
# 3. Split the data into training (70%) and test (30%) sets

# Use stratified sampling to ensure similar proportions of 'Personality' in both sets
set.seed(123) # for reproducibility
index <- createDataPartition(
  personality_data_complete$Personality,
  p = 0.7,
  list = FALSE,
  times = 1
)

train_data <- personality_data_complete[index, ]
test_data <- personality_data_complete[-index, ]

cat("\nData split complete.\n")
cat("Training set dimensions:", dim(train_data), "\n")
cat("Test set dimensions:", dim(test_data), "\n")
cat("\nProportion of 'Personality' in training set:\n")
print(prop.table(table(train_data$Personality)))
cat("\nProportion of 'Personality' in test set:\n")
print(prop.table(table(test_data$Personality)))

# 4. Create a Machine Learning Pipeline using caret

# Define preprocessing steps within the caret `train` function using `preProcess`
# - "knnImpute": Imputes missing values using k-nearest neighbors (alternative to mice if
    ↪ preferred within caret)
#               (Note: We already used mice, so this step might be redundant if data is already
    ↪ complete.
#               However, if there were new NAs introduced or if you prefer caret's imputation, it's
    ↪ here.)
# - "center" and "scale": Normalizes numerical features (mean 0, std dev 1)
# - "dummyVars": Creates dummy variables for categorical predictors (one-hot encoding)
#               This is crucial for many models that don't handle factors internally.

# For this specific problem, we've handled NAs with `mice`, so `knnImpute` in preProcess is not
    ↪ strictly needed
# unless you want to re-impute or use caret's method. I'll remove it for clarity, assuming mice is
    ↪ sufficient.

# However, for a robust pipeline, if `mice` was *not* used, `preProcess` would be:
# preproc_methods <- c("knnImpute", "center", "scale", "dummyVars")
# Since we used `mice`, let's just focus on centering, scaling, and dummy variables.

preproc_methods <- c("center", "scale", "dummyVars")

# Define training control for resampling (e.g., 10-fold cross-validation)
# This helps in robustly evaluating model performance and tuning hyperparameters.
train_control <- trainControl(
  method = "cv",          # Cross-validation
  number = 10,            # 10 folds
  summaryFunction = twoClassSummary, # For binary classification
  classProbs = TRUE,      # Needed for roc_auc or similar metrics
  savePredictions = "final" # Save predictions for analysis
)
```

Listing A3. *Cont.*

```
# Train the Random Forest model
# Random Forest is a good choice for this type of dataset as it handles mixed data types
# and is relatively robust to outliers and feature scaling.
cat("\nTraining the Random Forest model with preprocessing pipeline...\n")
set.seed(123) # for reproducibility of model training

model_pipeline <- train(
  Personality ~ ., # Predict 'Personality' using all other features
  data = train_data,
  method = "rf",           # Random Forest model
  preProcess = preproc_methods, # Preprocessing steps
  trControl = train_control,
  metric = "ROC",          # Optimize for Area Under the ROC Curve
  tuneLength = 3           # Number of unique 'mtry' values to try. Increase for more tuning.
)

cat("\nModel training complete.\n")
print(model_pipeline)

# 5. Evaluate the model on the test set

cat("\nEvaluating the model on the test set...\n")

# Make predictions on the test data
predictions <- predict(model_pipeline, newdata = test_data)
probabilities <- predict(model_pipeline, newdata = test_data, type = "prob")

# Confusion Matrix
confusion_matrix <- confusionMatrix(predictions, test_data$Personality, mode = "everything")
cat("\nConfusion Matrix on the Test Set:\n")
print(confusion_matrix)

# Calculate ROC AUC (if 'Extrovert' is the positive class, otherwise adjust positive = "")
# Ensure the levels of the factor match the order of columns in probabilities
# (caret usually handles this by default based on the factor levels)
roc_auc <- roc(
  response = as.numeric(test_data$Personality) - 1, # Convert factor to 0/1 (assuming Extrovert
      ↪ is 1, Introvert is 0)
  predictor = probabilities[, "Extrovert"], # Probability of the positive class (Extrovert)
  levels = levels(test_data$Personality)
)
cat("\nROC AUC on the Test Set:", auc(roc_auc), "\n")

# Display important variables (for Random Forest)
cat("\nVariable Importance (from Random Forest):\n")
print(varImp(model_pipeline))

# Summary of the model
cat("\nSummary of the trained model pipeline:\n")
summary(model_pipeline)

# You can save the trained model for future use
# saveRDS(model_pipeline, "personality_prediction_model.rds")
# To load: loaded_model <- readRDS("personality_prediction_model.rds")
```

References

1. Moor, J. The Dartmouth College Artificial Intelligence Conference: The Next Fifty years. *AI Mag.* **2006**, *27*, 87–89.
2. Grzybowski, A.; Pawlikowska-Łagód, K.; Lambert, W.C. A history of artificial intelligence. *Clin. Dermatol.* **2024**, *42*, 221–229. [CrossRef]
3. Russell, S.J.; Norvig, P. *Artificial Intelligence: A Modern Approach*; Pearson: London, UK, 2016.

4. McCorduck, P.; Cfe, C. *Machines Who Think: A Personal Inquiry into the History and Prospects of Artificial Intelligence*; AK Peters: Boca Raton, FL, USA; CRC Press: Boca Raton, FL, USA, 2004.
5. Mitchell, T. *Machine Learning*; McGraw-Hill International Editions; McGraw-Hill: Columbus, OH, USA, 1997.
6. LeCun, Y.; Bengio, Y.; Hinton, G. Deep learning. *Nature* **2015**, *521*, 436–444. [CrossRef] [PubMed]
7. Radford, A.; Wu, J.; Child, R.; Luan, D.; Amodei, D.; Sutskever, I. Language models are unsupervised multitask learners. *OpenAI Blog* **2019**, *1*, 9.
8. Brown, T.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.D.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. Language Models are Few-Shot Learners. In *Proceedings of the Advances in Neural Information Processing Systems*; Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., Lin, H., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2020; Volume 33, pp. 1877–1901.
9. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, L.u.; Polosukhin, I. Attention is All you Need. In *Proceedings of the Advances in Neural Information Processing Systems*; Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2017, Volume 30.
10. Kalla, D.; Smith, N.; Samaah, F.; Kuraku, S. Study and Analysis of chat GPT and its Impact on Different Fields of Study. *Int. J. Innov. Sci. Res. Technol.* **2023**, *8*, 827–833.
11. Dai, W.; Lin, J.; Jin, H.; Li, T.; Tsai, Y.S.; Gašević, D.; Chen, G. Can large language models provide feedback to students? A case study on ChatGPT. In *Proceedings of the 2023 IEEE International Conference on Advanced Learning Technologies (ICALT), 2023*; IEEE: Piscataway, NJ, USA, 2023; pp. 323–325.
12. Ng, D.T.K.; Leung, J.K.L.; Chu, S.K.W.; Qiao, M.S. Conceptualizing AI literacy: An exploratory review. *Comput. Educ. Artif. Intell.* **2021**, *2*, 100041. [CrossRef]
13. Colonna, L. Artificial Intelligence in Education (AIED): Towards More Effective Regulation. *Eur. J. Risk Regul.* **2025**, *17*, 161–181. [CrossRef]
14. Zawacki-Richter, O.; Marín, V.I.; Bond, M.; Gouverneur, F. Systematic review of research on artificial intelligence applications in higher education—Where are the educators? *Int. J. Educ. Technol. High. Educ.* **2019**, *16*, 39. [CrossRef]
15. Ifenthaler, D.; Majumdar, R.; Gorissen, P.; Judge, M.; Mishra, S.; Raffaghelli, J.; Shimada, A. Artificial intelligence in education: Implications for policymakers, researchers, and practitioners. *Technol. Knowl. Learn.* **2024**, *29*, 1693–1710. [CrossRef]
16. Wongvorachan, T.; Srisuttiyakorn, S.; Sriklaub, K. Optimizing Learning: Predicting Research Competency via Statistical Proficiency. *Trends High. Educ.* **2024**, *3*, 540–559. [CrossRef]
17. U.S. Bureau of Labor Statistics. Occupational Employment and Wage Statistics. Available online: https://www.bls.gov/oes/current/oes152041.htm (accessed on 15 March 2026).
18. Macher, D.; Paechter, M.; Papousek, I.; Ruggeri, K.; Freudenthaler, H.H.; Arendasy, M. Statistics anxiety, state anxiety during an examination, and academic achievement. *Br. J. Educ. Psychol.* **2013**, *83*, 535–549. [CrossRef]
19. McGrath, A.L. Content, affective, and behavioral challenges to learning: Students' experiences learning statistics. *Int. J. Scholarsh. Teach. Learn.* **2014**, *8*, 6. [CrossRef]
20. Lund, B.D.; Wang, T. Chatting about ChatGPT: How may AI and GPT impact academia and libraries? *Libr. Tech News* **2023**, *40*, 26–29. [CrossRef]
21. Budzianowski, P.; Vulić, I. Hello, it's GPT-2—How can I help you? Towards the use of pretrained language models for task-oriented dialogue systems. *arXiv* **2019**, arXiv:1907.05774.
22. Tamkin, A.; Brundage, M.; Clark, J.; Ganguli, D. Understanding the capabilities, limitations, and societal impact of large language models. *arXiv* **2021**, arXiv:2102.02503. [CrossRef]
23. Liu, J.; Xia, C.S.; Wang, Y.; Zhang, L. Is Your Code Generated by ChatGPT Really Correct? Rigorous Evaluation of Large Language Models for Code Generation. In *Proceedings of the Advances in Neural Information Processing Systems*; Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., Levine, S., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2023; Volume 36, pp. 21558–21572.
24. Bucaioni, A.; Ekedahl, H.; Helander, V.; Nguyen, P.T. Programming with ChatGPT: How far can we go? *Mach. Learn. Appl.* **2024**, *15*, 100526. [CrossRef]
25. Moussiades, L.; Zografos, G.; Papakostas, G. GPT-4 vs. GPT-3.5 as coding assistants. *preprint* **2024**. Available online: https://www.researchsquare.com/article/rs-3920214/v1 (accessed on 15 March 2026).
26. Heitz, L.B.; Chamas, J.; Scherb, C. Evaluation of the Programming Skills of Large Language Models. *arXiv* **2024**, arXiv:2405.14388. [CrossRef]
27. Elgedawy, R.; Sadik, J.; Dutta, S.; Gautam, A.; Georgiou, K.; Gholamrezae, F.; Ji, F.; Lim, K.; Liu, Q.; Ruoti, S. Ocassionally secure: A comparative analysis of code generation assistants. *arXiv* **2024**, arXiv:2402.00689. [CrossRef]
28. Hou, W.; Ji, Z. A systematic evaluation of large language models for generating programming code. *arXiv* **2024**, arXiv:2403.00894. [CrossRef]
29. Song, X.; Xie, K.; Lee, L.; Chen, R.; Clark, J.M.; He, H.; He, H.; Min, J.; Zhang, X.; Zheng, S.; et al. Performance Evaluation of Large Language Models in Statistical Programming. *arXiv* **2025**, arXiv:2502.13117. [CrossRef]

30. Beer, R.; Feix, A.; Guttzeit, T.; Muras, T.; Müller, V.; Rauscher, M.; Schäffler, F.; Löwe, W. Examination of Code generated by Large Language Models. *arXiv* **2024**, arXiv:2408.16601. [CrossRef]
31. Górecki, J. Pair programming with ChatGPT for sampling and estimation of copulas. *Comput. Stat.* **2024**, *39*, 3231–3261. [CrossRef]
32. Buscemi, A. A comparative study of code generation using chatgpt 3.5 across 10 programming languages. *arXiv* **2023**, arXiv:2308.04477. [CrossRef]
33. Tian, H.; Lu, W.; Li, T.O.; Tang, X.; Cheung, S.C.; Klein, J.; Bissyandé, T.F. Is ChatGPT the ultimate programming assistant—How far is it? *arXiv* **2023**, arXiv:2304.11938.
34. Tucker, M.C.; Shaw, S.T.; Son, J.Y.; Stigler, J.W. Teaching Statistics and Data Analysis with R. *J. Stat. Data Sci. Educ.* **2023**, *31*, 18–32. [CrossRef]
35. Baumer, B.; Cetinkaya-Rundel, M.; Bray, A.; Loi, L.; Horton, N.J. R Markdown: Integrating a reproducible analysis tool into introductory statistics. *arXiv* **2014**, arXiv:1402.1894. [CrossRef]
36. Nolan, D.; Lang, D.T. *Data Science in R: A Case Studies Approach to Computational Reasoning and Problem Solving*; CRC Press: Boca Raton, FL, USA, 2015.
37. Mascaró, M.; Sacristán, A.I.; Rufino, M.M. For the love of statistics: appreciating and learning to apply experimental analysis and statistics through computer programming activities. *Teach. Math. Its Appl. Int. J. IMA* **2016**, *35*, 74–87. [CrossRef]
38. Fisher, R.A. *The Design of Experiments*; Oliver and Boyd: Edinburgh, UK, 1937; Volume 2.
39. British Board of Trade. *Report on the Loss of the 'Titanic' (S.S.)*; Allan Sutton Publishing: Gloucester, UK, 1990.
40. Hastie, T.; Tibshirani, R.; Friedman, J.H.; Friedman, J.H. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*; Springer: Berlin, Germany, 2009; Volume 2.
41. Fisher, R.A. The use of multiple measurements in taxonomic problems. *Ann. Eugen.* **1936**, *7*, 179–188. [CrossRef]
42. Kaggle. Extrovert vs. Introvert Behavior Data. Available online: https://www.kaggle.com/datasets/rakeshkapilavai/extrovert-vs-introvert-behavior-data/data (accessed on 15 March 2026).
43. Evtikhiev, M.; Bogomolov, E.; Sokolov, Y.; Bryksin, T. Out of the bleu: How should we assess quality of the code generation models? *J. Syst. Softw.* **2023**, *203*, 111741. [CrossRef]
44. CRAN. lintr: A 'Linter' for R Code. Available online: https://cran.r-project.org/web/packages/lintr/index.html (accessed on 15 March 2026).
45. Rousseeuw, P.J. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *J. Comput. Appl. Math.* **1987**, *20*, 53–65. [CrossRef]
46. HIX.AI. Chat Interface. Available online: https://hix.ai/chat (accessed on 15 March 2026).
47. Zheng, Q.; Xia, X.; Zou, X.; Dong, Y.; Wang, S.; Xue, Y.; Wang, Z.; Shen, L.; Wang, A.; Li, Y.; et al. Codegeex: A pre-trained model for code generation with multilingual evaluations on humaneval-x. *arXiv* **2023**, arXiv:2303.17568.
48. Cheng, Y.; Sanders, M.; Le, A.T. Using LLM to Generate Questions and Answers for Statistics Education: Is It Adequate? In *Artificial Intelligence in Education—Creating an Equitable, Creative, and Effective Learning Environment*; IntechOpen: London, UK, 2026.
49. Cheung, B.H.H.; Lau, G.K.K.; Wong, G.T.C.; Lee, E.Y.P.; Kulkarni, D.; Seow, C.S.; Wong, R.; Co, M.T.H. ChatGPT versus human in generating medical graduate exam multiple choice questions—A multinational prospective study (Hong Kong SAR, Singapore, Ireland, and the United Kingdom). *PLoS ONE* **2023**, *18*, e0290691. [CrossRef] [PubMed]
50. Powell, W.; Courchesne, S. Opportunities and risks involved in using ChatGPT to create first grade science lesson plans. *PLoS ONE* **2024**, *19*, e0305337. [CrossRef]
51. Lee, U.; Kim, Y.; Lee, S.; Park, J.; Mun, J.; Lee, E.; Kim, H.; Lim, C.; Yoo, Y.J. Can we Use GPT-4 as a Mathematics Evaluator in Education?: Exploring the Efficacy and Limitation of LLM-based Automatic Assessment System for Open-ended Mathematics Question. *Int. J. Artif. Intell. Educ.* **2024**, *35*, 1560–1596. [CrossRef]
52. Rudolph, J.; Tan, S.; Tan, S. ChatGPT: Bullshit spewer or the end of traditional assessments in higher education? *J. Appl. Learn. Teach.* **2023**, *6*, 342–363. [CrossRef]