MONASH University

Advances in Artificial Intelligence for Data Visualization: Developing Computer Vision Models to Automate Reading of Data Plots, with Application to Regression Diagnostics

Weihao Li

PhD, Monash University

A thesis submitted for the degree of
Doctor of Philosophy
at Monash University in 2024
Department of Econometrics & Business Statistics

Table of contents

Copyright notice

Produced on 23 August 2024.

Abstract

The abstract should outline the main approach and findings of the thesis and must not be more than 500 words.

Declaration

Use only one of the following declarations (Standard thesis or Thesis including published works declaration) and remove the other.

Standard thesis

This thesis is an original work of my research and contains no material which has been accepted for the award of any other degree or diploma at any university or equivalent institution and that, to the best of my knowledge and belief, this thesis contains no material previously published or written by another person, except where due reference is made in the text of the thesis.

Student name:

Student signature:

Date:

Publications during enrolment

Remove this section if you do not have publications.

The material in Chapter 1 has been submitted to the journal *Journal of Impossible Results* for possible publication.

The contribution in **?@sec-litreview** of this thesis was presented in the International Symposium on Nonsense held in Dublin, Ireland, in July 2022.

Reproducibility statement

This thesis is written using Quarto with renv (**?**) to create a reproducible environment. All materials (including the data sets and source files) required to reproduce this document can be found at the Github repository `github.com/SusanSu/thesis`.

Thesis including published works declaration

I hereby declare that this thesis contains no material which has been accepted for the award of any other degree or diploma at any university or equivalent institution and that, to the best of my knowledge and belief, this thesis contains no material previously published or written by another person, except where due reference is made in the text of the thesis.

This thesis includes ?? original papers published in peer reviewed journals and ?? submitted publications. The core theme of the thesis is ??. The ideas, development and writing up of all the papers in the thesis were the principal responsibility of myself, the student, working within the Department of Econometrics & Business Statistics under the supervision of ??

(The inclusion of co-authors reflects the fact that the work came from active collaboration between researchers and acknowledges input into team-based research.)

In the case of (??insert chapter numbers) my contribution to the work involved the following:

Thesis chapter	Publication title	Status	Nature and % of student contribution	Nature and % of coauthors' contribution	Coauthors are Monash students
2	The life cycle of Mongolian crickets	Submitted	Concept and data analysis, writing first draft: 60%	Shu Xu, input into manuscript: 25%; Eddie Betts, input into manuscript: 15%	Shu Xu: No; Eddie Betts: Yes

I have / have not renumbered sections of submitted or published papers in order to generate a consistent presentation within the thesis.

Student name:

Student signature:

Date:

I hereby certify that the above declaration correctly reflects the nature and extent of the student's and co-authors' contributions to this work. In instances where I am not the responsible author I have consulted with the responsible author to agree on the respective contributions of the authors.

Main Supervisor name:

Main Supervisor signature:

Date:

Acknowledgements

I would like to thank my pet goldfish for ...

In accordance with Chapter 7.1.4 of the research degrees handbook, if you have engaged the services of a professional editor, you must provide their name and a brief description of the service rendered. If the professional editor's current or former area of academic specialisation is similar your own, this too should be stated as it may suggest to examiners that the editor's advice to the student has extended beyond guidance on English expression to affect the substance and structure of the thesis.

If you have used generative artificial intelligence (AI) technologies, you must include a written acknowledgment of the use and its extent. Your acknowledgement should at a minimum specify which technology was used, include explicit description on how the information was generated, and explain how the output was used in your work. Below is a suggested format:

"I acknowledge the use of [insert AI system(s) and link] to [specific use of generative artificial intelligence]. The output from these was used to [explain use]."

Free text section for you to record your acknowledgment and gratitude for the more general academic input and support such as financial support from grants and scholarships and the non-academic support you have received during the course of your enrolment. If you are a recipient of the "Australian Government Research Training Program Scholarship", you are required to include the following statement:

> "This research was supported by an Australian Government Research Training Program (RTP) Scholarship."

You may also wish to acknowledge significant and substantial contribution made by others to the research, work and writing represented and/or reported in the thesis. These could include significant contributions to: the conception and design of the project; non-routine

technical work; analysis and interpretation of research data; drafting significant parts of the work or critically revising it so as to contribute to the interpretation.

Chapter 1

Introduction

This is where you introduce the main ideas of your thesis, and an overview of the context and background.

In a PhD, Chapter 2 would normally contain a literature review. Typically, Chapters 3–5 would contain your own contributions. Think of each of these as potential papers to be submitted to journals. Finally, Chapter 6 provides some concluding remarks, discussion, ideas for future research, and so on. Appendixes can contain additional material that don't fit into any chapters, but that you want to put on record. For example, additional tables, output, etc.

1.1 Quarto

In this template, the rest of the chapter shows how to use quarto. The big advantage of using quarto is that it allows you to include your R or Python code directly into your thesis, to ensure there are no errors in copying and pasting, and that everything is reproducible. It also helps you stay better organized.

For details on using Quarto, see http://quarto.org.

1.2 Data

Included in this template is a file called `sales.csv`. This contains quarterly data on Sales and Advertising budget for a small company over the period 1981–2005. It also contains the GDP (gross domestic product) over the same period. All series have been adjusted for inflation. We can load in this data set using the following code:

```
sales <- readr::read_csv(here::here("data/sales.csv")) |>
  rename(Quarter = `...1`) |>
```

```
  mutate(
    Quarter = as.Date(paste0("01-", Quarter), "%d-%b-%y"),
    Quarter = yearquarter(Quarter)
  ) |>
  as_tsibble(index = Quarter)
```

Any data you use in your thesis can go into the `data` directory. The data should be in exactly the format you obtained it. Do no editing or manipulation of the data prior to including it in the `data` directory. Any data munging should be scripted and form part of your thesis files (possibly hidden in the output).

1.3 Figures

Figure 1.1 shows time plots of the data we just loaded. Notice how figure captions and references work. Chunk names can be used as figure labels with `fig-` prefixed. Never manually type figure numbers, as they can change when you add or delete figures. This way, the figure numbering is always correct.

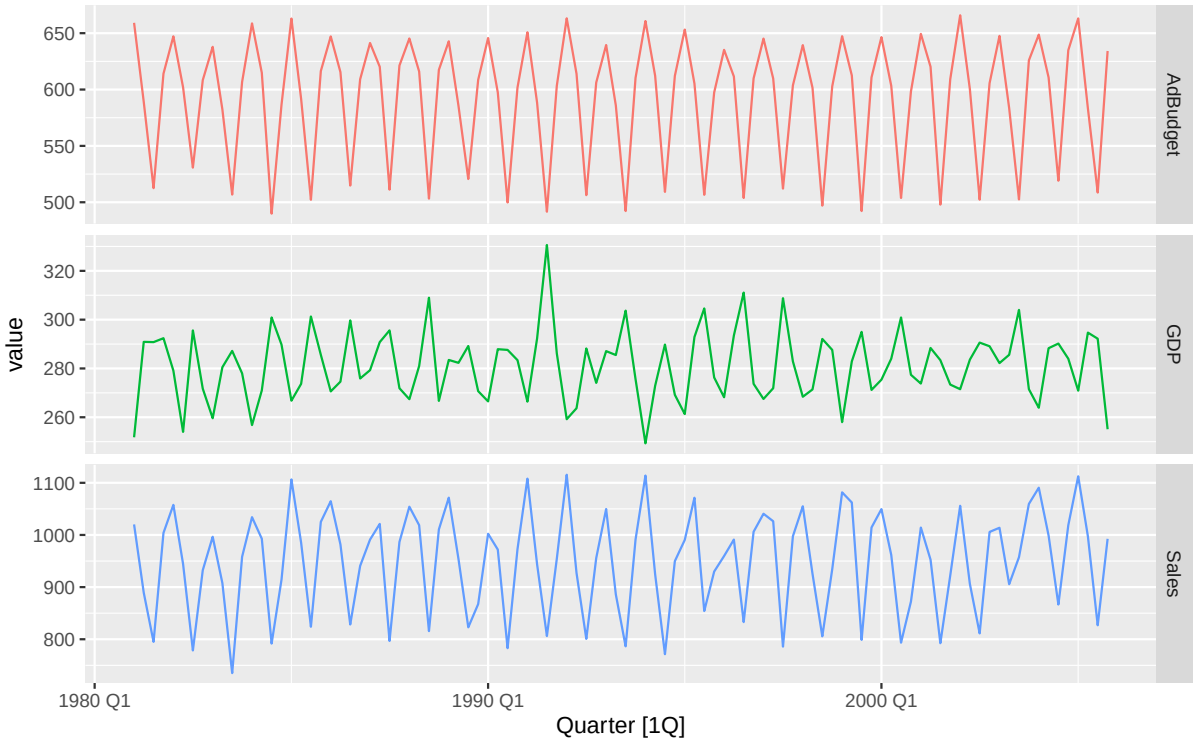


Figure 1.1: *Quarterly sales, advertising and GDP data.*

1.4 Results from analyses

We can fit a regression model to the sales data.

If y_t denotes the sales in quarter t, x_t denotes the corresponding advertising budget and z_t denotes the GDP, then the resulting model is:

$$y_t = \beta x_t + \gamma z_t + \varepsilon_t \tag{1.1}$$

where $\hat{\beta} = 1.85$, and $\hat{\gamma} = 1.04$. We can reference this equation using Equation 1.1.

1.5 Tables

We can also make a nice summary table of the coefficients, as shown in Table 1.1

Table 1.1: *Coefficients from the fitted model.*

Coefficient	Estimate	P value
(Intercept)	-438.98	0.02
GDP	1.04	0.02
AdBudget	1.85	0.00

Again, notice the use of labels and references to automatically generate table numbers.

Chapter 2

A Plot is Worth a Thousand Tests: Assessing Residual Diagnostics with the Lineup Protocol

Regression experts consistently recommend plotting residuals for model diagnosis, despite the availability of many numerical hypothesis test procedures designed to use residuals to assess problems with a model fit. Here we provide evidence for why this is good advice using data from a visual inference experiment. We show how conventional tests are too sensitive, which means that too often the conclusion would be that the model fit is inadequate. The experiment uses the lineup protocol which puts a residual plot in the context of null plots. This helps generate reliable and consistent reading of residual plots for better model diagnosis. It can also help in an obverse situation where a conventional test would fail to detect a problem with a model due to contaminated data. The lineup protocol also detects a range of departures from good residuals simultaneously. Supplemental materials for the article are available online.

2.1 Introduction

> *"Since all models are wrong the scientist must be alert to what is importantly wrong."* (Box 1976)

Diagnosing a model is an important part of building an appropriate model. In linear regression analysis, studying the residuals from a model fit is a common diagnostic activity. Residuals summarise what is not captured by the model, and thus provide the capacity to identify what might be wrong.

We can assess residuals in multiple ways. To examine the univariate distribution, residuals may be

plotted as a histogram or normal probability plot. Using the classical normal linear regression model as an example, if the distribution is symmetric and unimodal, we would consider it to be well-behaved. However, if the distribution is skewed, bimodal, multimodal, or contains outliers, there would be cause for concern. We can also inspect the distribution by conducting a goodness-of-fit test, such as the Shapiro-Wilk normality test (Shapiro and Wilk 1965).

Scatterplots of residuals against the fitted values, and each of the explanatory variables, are commonly used to scrutinize their relationships. If there are any visually discoverable associations, the model is potentially inadequate or incorrectly specified. We can also potentially discover patterns not directly connected to a linear model assumption from these residual plots, such as the discreteness or skewness of the fitted values, and outliers. To read residual plots, one looks for noticeable departures from the model such as non-linear pattern or heteroskedasticity. A non-linear pattern would suggest that the model needs to have some additional non-linear terms. Heteroskedasticity suggests that the error is dependent on the predictors, and hence violates the independence assumption. Statistical tests were developed to provide objective assessment, for example, of non-linear patterns (e.g. Ramsey 1969), and heteroskedasticity (e.g. Breusch and Pagan 1979).

The common wisdom of experts is that plotting the residuals is indispensable for diagnosing model fits (Cook and Weisberg 1982; Draper and Smith 1998; Montgomery et al. 1982). The lack of empirical evidence for the ubiquitous advice is *curious*, and is what this article tackles.

Additionally, relying solely on the subjective assessment of a single plot can be problematic. People will almost always see a pattern (see Kahneman 2011), so the question that really needs answering is whether any pattern perceived is consistent with randomness, or sampling variability, or noise. Correctly judging whether *no* pattern exists in a residual plot is a difficult task. Loy (2021) emphasizes that this is especially difficult to teach to new analysts and students, and advocates to the broader use of the lineup protocol (Buja et al. 2009a).

The lineup protocol places a data plot in a field of null plots, allowing for a comparison of patterns due purely by chance to what is perceived in the data plot. For residual analysis this is especially helpful for gauging whether there is *no* pattern. (Figure 2.1 shows an example of a lineup of residual plots.) In its strict use, one would insist that the data plot is not seen before seeing the lineup, so that the observer does not know which is the true plot. When used this way, it provides an objective test for data plots. Majumder et al. (2013a) validated that results from lineups assessed by human observers performed similarly to conventional tests. One would not use a lineup when a conventional test exists and is adequate because it is more manually expensive to conduct. However, where no adequate conventional test exists, it is invaluable, as shown by Loy and Hofmann (2013). Here we use

the lineup as a vehicle to rigorously explore why experts advise that residual plots are indispensable despite the prevalence of numerical tests.

The paper is structured as follows. Section 2.2 describes the background on the types of departures that one expects to detect, and outlines a formal statistical process for reading residual plots, called visual inference. Section 2.3 describes the calculation of the statistical significance and power of the test. Section 2.4 details the experimental design to compare the decisions made by formal hypothesis testing, and how humans would read diagnostic plots. The results are reported in Section 2.5. We conclude with a discussion of the presented work, and ideas for future directions.

2.2 Background

2.2.1 Departures from Good Residual Plots

Graphical summaries where residuals are plotted against fitted values, or other functions of the predictors (expected to be approximately orthogonal to the residuals) are considered to be the most important residual plots by Cook and Weisberg (1999). Figure 2.2A shows an example of an ideal residual plot where points are symmetrically distributed around the horizontal zero line (red), with no discernible patterns. There can be various types of departures from this ideal pattern. Non-linearity, heteroskedasticity and non-normality, shown in Figure 2.2B, Figure 2.2C, and Figure 2.2D, respectively, are three commonly checked departures.

Model misspecification occurs if functions of predictors that needed to accurately describe the relationship with the response are incorrectly specified. This includes instances where a higher-order polynomial term of a predictor is wrongfully omitted. Any non-linear pattern visible in the residual plot could be indicative of this problem. An example residual plot containing visual pattern of non-linearity is shown in Figure 2.2B. One can clearly observe the “S-shape” from the residual plot, which corresponds to the cubic term that should have been included in the model.

Heteroskedasticity refers to the presence of non-constant error variance in a regression model. It indicates that the distribution of residuals depends on the predictors, violating the independence assumption. This can be seen in a residual plot as an inconsistent spread of the residuals relative to the fitted values or predictors. An example is the “butterfly” shape shown in Figure 2.2C, or a “left-triangle” and “right-triangle” shape where the smallest variance occurs at one side of the horizontal axis.

Figure 2.2D shows a scatterplot where the residuals have a skewed distribution, as seen by the uneven vertical spread. Unlike non-linearity and heteroskedasticity, non-normality is usually detected with a different type of residual plot: a histogram or a normal probability plot. Because we focus on

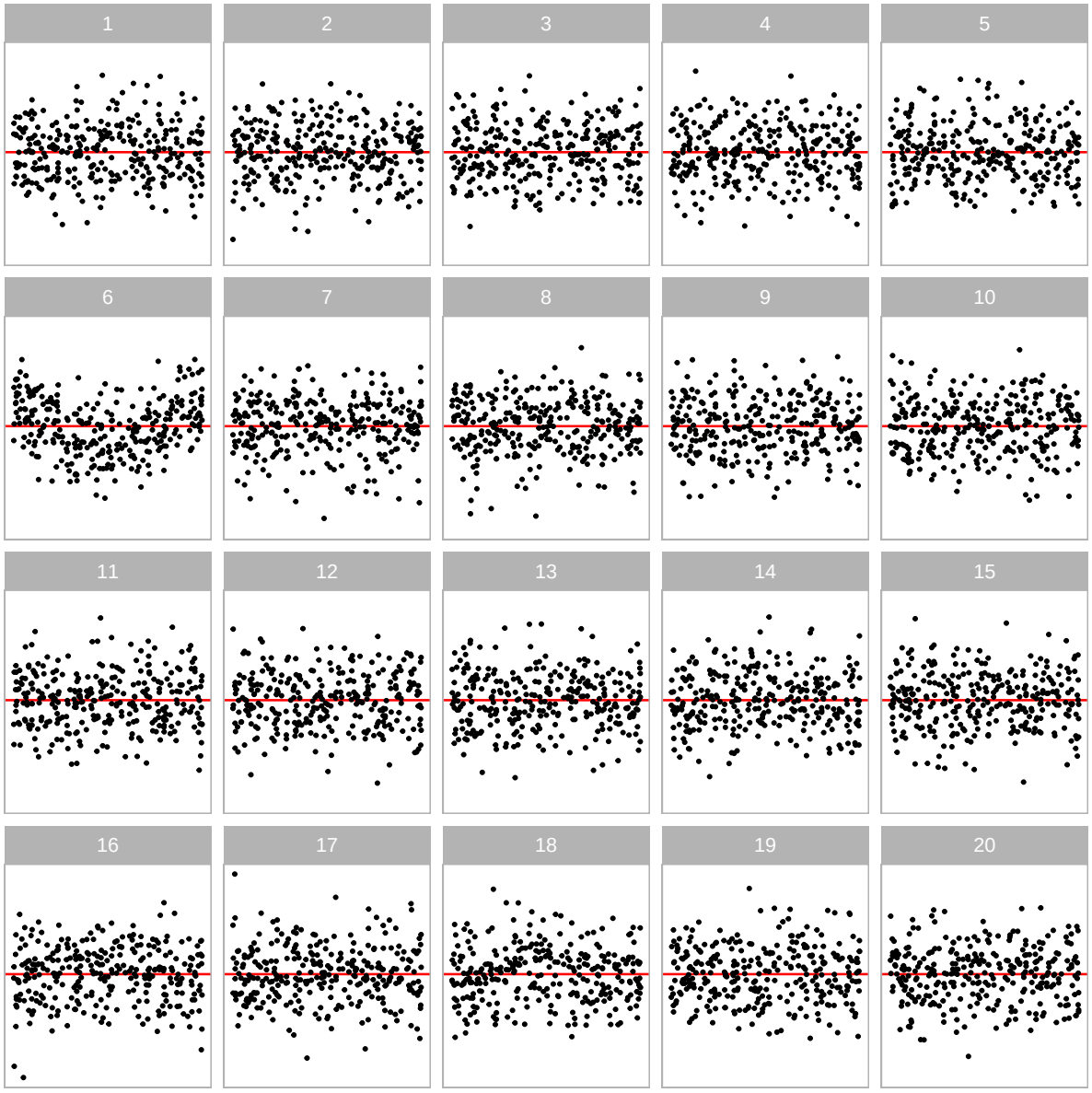


Figure 2.1: *Visual testing is conducted using a lineup, as in the example here. The residual plot computed from the observed data is embedded among 19 null plots, where the residuals are simulated from a standard error model. Computing the p-value requires that the lineup be examined by a number of human judges, each asked to select the most different plot. A small p-value would result from a substantial number selecting the data plot (at position 6, exhibiting non-linearity).*

scatterplots, non-normality is not one of the departures examined in depth in this paper. (Loy et al. 2016 discuss related work on non-normality checking.)

2.2.2 Conventionally Testing for Departures

Many different hypothesis tests are available to detect specific model defects. For example, the presence of heteroskedasticity can usually be tested by applying the White test (White 1980) or the Breusch-Pagan (BP) test (Breusch and Pagan 1979), which are both derived from the Lagrange

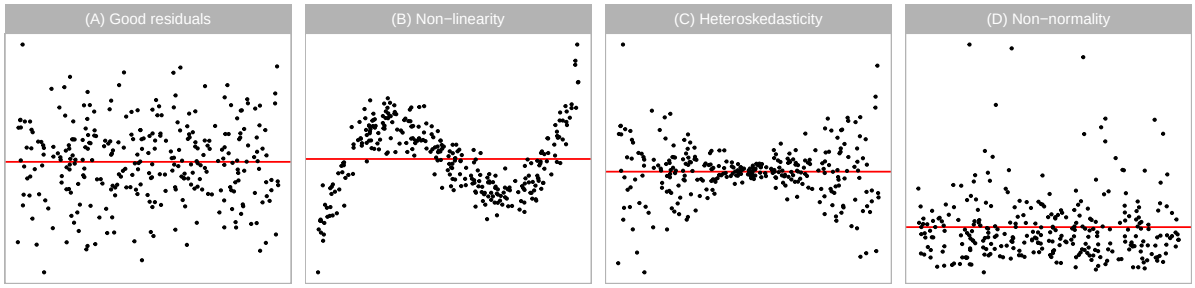


Figure 2.2: *Example residual vs fitted value plots (horizontal line indicates 0): (A) classically good looking residuals, (B) non-linear pattern indicates that the model has not captured a non-linear association, (C) heteroskedasticity indicating that variance around the fitted model is not uniform, and (D) non-normality where the residual distribution is not symmetric around 0. The latter pattern might best be assessed using a univariate plot of the residuals, but patterns B and C need to be assessed using a residual vs fitted value plot.*

multiplier test (Silvey 1959) principle that relies on the asymptotic properties of the null distribution. To test specific forms of non-linearity, one may apply the F-test as a model structural test to examine the significance of a specific polynomial and non-linear forms of the predictors, or the significance of proxy variables as in the Ramsey Regression Equation Specification Error Test (RESET) (Ramsey 1969). The Shapiro-Wilk (SW) normality test (Shapiro and Wilk 1965) is the most widely used test of non-normality included by many of the statistical software programs. The Jarque-Bera test (Jarque and Bera 1980) is also used to directly check whether the sample skewness and kurtosis match a normal distribution.

Table 2.1 displays the p-values from the RESET, BP and SW tests applied to the residual plots in Figure 2.2. The RESET test and BP test were computed using the `resettest` and `bptest` functions from the R package `lmtest`, respectively. The SW test was computed using the `shapiro.test` from the core R package `stats`. [1] The RESET test requires the selection of a power parameter. Ramsey (1969) recommends a power of four, which we adopted in our analysis.

For residual plots in Figure 2.2, we would expect the RESET test for non-linearity to reject residual plot B, the BP test for heteroskedasticity to reject the residual plot C, and the SW test for non-normality to reject residual plot D, which they all do and all tests also correctly fail to reject residual plot A. Interestingly, the BP and SW tests also reject the residual plots exhibiting structure that they were not designed for. Cook and Weisberg (1982) explain that most residual-based tests for a particular departure from the model assumptions are also sensitive to other types of departures. This could be considered a Type III error (Kimball 1957), where the null hypothesis of good residuals is correctly rejected but for the wrong reason. Also, some types of departure can have elements of other types

[1] Although we did not use it, it is useful to know that the R package `skedastic` (Farrar 2020) also contains a large collection of functions to test for heteroskedasticity.

Table 2.1: *Statistical significance testing for departures from good residuals for plots in Figure 2.2. Shown are the p-values calculated for the RESET, the BP and the SW tests. The good residual plot (A) is judged a good residual plot, as expected, by all tests. The non-linearity (B) is detected by all tests, as might be expected given the extreme structure.*

Plot	Departures	RESET	BP	SW
A	None	0.779	0.133	0.728
B	Non-linearity	*0.000*	*0.000*	*0.039*
C	Heteroskedasticity	0.658	*0.000*	*0.000*
D	Non-normality	0.863	0.736	*0.000*

of departure, for example, non-linearity can appear like heteroskedasticity. Additionally, other data problems such as outliers can trigger rejection (or not) of the null hypothesis (Cook and Weisberg 1999).

With large sample sizes, hypothesis tests may reject the null hypothesis when there is only a small effect. (A good discussion can be found in Kirk (1996).) While such rejections may be statistically correct, their sensitivity may render the results impractical. A key goal of residual plot diagnostics is to identify potential issues that could lead to incorrect conclusions or errors in subsequent analyses, but minor defects in the model are unlikely to have a significant impact and may be best disregarded for practical purposes. The experiment discussed in this paper specifically addresses this tension between statistical significance and practical significance.

2.2.3 Visual Test Procedure based on Lineups

The examination of data plots to infer signals or patterns (or lack thereof) is fraught with variation in the human ability to interpret and decode the information embedded in a graph (Cleveland and McGill 1984). In practice, over-interpretation of a single plot is common. For instance, Roy Chowdhury et al. (2015) described a published example where authors over-interpreted separation between gene groups from a two-dimensional projection of a linear discriminant analysis even when there were no differences in the expression levels between the gene groups.

One solution to over-interpretation is to examine the plot in the context of natural sampling variability assumed by the model, called the lineup protocol, as proposed in Buja et al. (2009b). Majumder et al. (2013b) showed that the lineup protocol is analogous to the null hypothesis significance testing framework. The protocol consists of m randomly placed plots, where one plot is the data plot, and the remaining $m-1$ plots, referred to as the *null plots*, are constructed using the same graphical procedure as the data plot but the data is replaced with null data that is generated in a manner consistent with the null hypothesis, H_0. Then, an observer who has not seen the data plot is asked to point out the most different plot from the lineup. Under H_0, it is expected that the data plot would

have no distinguishable difference from the null plots, and the probability that the observer correctly picks the data plot is $1/m$. If one rejects H_0 as the observer correctly picks the data plot, then the Type I error of this test is $1/m$. This protocol requires a priori specification of H_0 (or at least a null data generating mechanism), much like the requirement of knowing the sampling distribution of the test statistic in null hypothesis significance testing framework.

Figure Figure 2.1 is an example of a lineup protocol. If the data plot at position 6 is identifiable, then it is evidence for the rejection of H_0. In fact, the actual residual plot is obtained from a misspecified regression model with missing non-linear terms.

Data used in the $m-1$ null plots needs to be simulated. In regression diagnostics, sampling data consistent with H_0 is equivalent to sampling data from the assumed model. As Buja et al. (2009b) suggested, H_0 is usually a composite hypothesis controlled by nuisance parameters. Since regression models can have various forms, there is no general solution to this problem, but it sometimes can be reduced to a so called "reference distribution" by applying one of the three methods: (i) sampling from a conditional distribution given a minimal sufficient statistic under H_0, (ii) parametric bootstrap sampling with nuisance parameters estimated under H_0, and (iii) Bayesian posterior predictive sampling. The conditional distribution given a minimal sufficient statistic is the best justified reference distribution among the three (Buja et al. 2009b). Under this method, the null residuals can essentially be simulated by independent drawing from a standard normal random distribution, then regressing the draws on the predictors, and then re-scaling it by the ratio of the residual sum of square in two regressions.

The effectiveness of lineup protocol for regression analysis has been validated by Majumder et al. (2013b) under relatively simple settings with up to two predictors. Their results suggest that visual tests are capable of testing the significance of a single predictor with a similar power to a t-test, though they express that in general it is unnecessary to use visual inference if there exists a corresponding conventional test, and they do not expect the visual test to perform equally well as the conventional test. In their third experiment, where the contamination of the data violate the assumptions of the conventional test, visual test outperforms the conventional test by a large margin. This supports the use of visual inference in situations where there are no existing numerical testing procedures. Visual inference has also been integrated into diagnostics for hierarchical linear models where the lineup protocol is used to judge the assumptions of linearity, normality and constant error variance for both the level-1 and level-2 residuals (Loy and Hofmann 2013, 2014, 2015).

2.3 Calculation of Statistical Significance and Test Power

2.3.1 What is Being Tested?

In diagnosing a model fit using the residuals, we are generally interested in testing whether "*the regression model is correctly specified*" (H_0) against the broad alternative "*the regression model is misspecified*" (H_a). However, it is practically impossible to test this broad H_0 with conventional tests, because they need specific structure causing the departure to be quantifiable in order to be computable. For example, the RESET test for detecting non-linear departures is formulated by fitting $y = \tau_0 + \sum_{i=1}^{p} \tau_p x_p + \gamma_1 \hat{y}^2 + \gamma_2 \hat{y}^3 + \gamma_3 \hat{y}^4 + u,\ \ u \sim N(0, \sigma_u^2)$ in order to test $H_0 : \gamma_1 = \gamma_2 = \gamma_3 = 0$ against $H_a : \gamma_1 \neq 0$ or $\gamma_2 \neq 0$ or $\gamma_3 \neq 0$. Similarly, the BP test is designed to specifically test H_0 : *error variances are all equal* ($\zeta_i = 0$ for $i = 1, .., p$) versus the alternative H_a : *that the error variances are a multiplicative function of one or more variables* (at least one $\zeta_i \neq 0$) from $e^2 = \zeta_0 + \sum_{i=1}^{p} \zeta_i x_i + u,\ u \sim N(0, \sigma_u^2)$.

While a battery of conventional tests for different types of departures could be applied, this is intrinsic to the lineup protocol. The lineup protocol operates as an omnibus test, able to detect a range of departures from good residuals in a single application.

2.3.2 Statistical Significance

In hypothesis testing, a p-value is defined as the probability of observing test results at least as extreme as the observed result assuming H_0 is true. Conventional hypothesis tests usually have an existing method to derive or compute the p-value based on the null distribution. The method to estimate a p-value for a visual test essentially follows the process detailed by VanderPlas et al. (2021). Details are given in Appendix A.1.

2.3.3 Power of the Tests

The power of a model misspecification test is the probability that H_0 is rejected given the regression model is misspecified in a specific way. It is an important indicator when one is concerned about whether model assumptions have been violated. In practice, one might be more interested in knowing how much the residuals deviate from the model assumptions, and whether this deviation is of practical significance.

The power of a conventional hypothesis test is affected by both the true parameters $\boldsymbol{\theta}$ and the sample size n. These two can be quantified in terms of effect size E to measure the strength of the residual departures from the model assumptions. Details about the calculation of effect size are provided in Section 2.4.2 after the introduction of the simulation model used in our experiment. The theoretical power of a test is sometimes not a trivial solution, but it can be estimated if the data generating

process is known. We use a predefined model to generate a large set of simulated data under different effect sizes, and record if the conventional test rejects H_0. The probability of the conventional test rejects H_0 is then fitted by a logistic regression formulated as

$$Pr(\text{reject } H_0|H_1, E) = \Lambda\left(\log\left(\frac{0.05}{0.95}\right) + \beta_1 E\right), \tag{2.1}$$

where $\Lambda(.)$ is the standard logistic function given as $\Lambda(z) = \exp(z)(1+\exp(z))^{-1}$. The effect size E is the only predictor and the intercept is fixed to $\log(0.05/0.95)$ so that $\hat{Pr}(\text{reject } H_0|H_1, E = 0) = 0.05$, the desired significance level.

The power of a visual test on the other hand, may additionally depend on the ability of the particular participant, as the skill of each individual may affect the number of observers who identify the data plot from the lineup (Majumder et al. 2013b). To address this issue, Majumder et al. (2013b) models the probability of participant j correctly picking the data plot from lineup l using a mixed-effect logistic regression, with participants treated as random effects. Then, the estimated power of a visual test evaluated by a single participant is the predicted value obtained from the mixed effects model. However, this mixed effects model does not work with scenario where participants are asked to select one or more most different plots. In this scenario, having the probability of a participant j correctly picking the data plot from a lineup l is insufficient to determine the power of a visual test because it does not provide information about the number of selections made by the participant for the calculation of the p-value. Therefore, we directly estimate the probability of a lineup being rejected by assuming that individual skill has negligible effect on the variation of the power. This assumption essentially averages out the subject ability and helps to simplify the model structure, thereby obviating a costly large-scale experiment to estimate complex covariance matrices. The same model given in Equation Equation 2.1 is applied to model the power of a visual test.

To study various factors contributing to the power of both tests, the same logistic regression model is fit on different subsets of the collated data grouped by levels of factors. These include the distribution of the fitted values, type of the simulation model and the shape of the residual departures.

2.4 Experimental Design

Our experiment was conducted over three data collection periods to investigate the difference between conventional hypothesis testing and visual inference in the application of linear regression diagnostics. Two types of departures, non-linearity and heteroskedasticity, were collected during data collection periods I and II. The data collection period III was designed primarily to measure human responses to null lineups so that the visual p-values can be estimated. Additional lineups for both non-linearity

Table 2.2: *Levels of the factors used in data collection periods I, II, and III.*

Non-linearity		Heteroskedasticity		Common	
Poly Order (j)	SD (σ)	Shape (a)	Ratio (b)	Size (n)	Distribution of the fitted values
2	0.25	-1	0.25	50	Uniform
3	1.00	0	1.00	100	Normal
6	2.00	1	4.00	300	Skewed
18	4.00		16.00		Discrete
			64.00		

and heteroskedasticity, using uniform fitted value distributions, were included for additional data, and to avoid participant frustration of too many difficult tasks.

During the experiment, every participant recruited from the Prolific crowd-sourcing platform (Palan and Schitter 2018) was presented with a block of 20 lineups. A lineup consisted of a randomly placed data plot and 19 null plots, which were all residual plots drawn with raw residuals on the y-axis and fitted values on the x-axis. An additional horizontal red line was added at $y = 0$ as a visual reference. The data in the data plot was simulated from one of two models described in Section 2.4.1, while the data of the remaining 19 null plots were generated by the residual rotation technique discussed in Section Section 2.2.3.

In each lineup evaluation, the participant was asked to select one or more plots that are most different from others, provide a reason for their selections, and evaluate how different they think the selected plots are from others. If there is no noticeable difference between plots in a lineup, participants had the option to select zero plots without the need to provide a reason. During the process of recording the responses, a zero selection was considered to be equivalent to selecting all 20 plots. No participant was shown the same lineup twice. Information about preferred pronouns, age group, education, and previous experience in visual experiments were also collected. A participant's submission was only included in the analysis if the data plot is identified for at least one attention check.

Overall, we collected 7974 evaluations on 1152 unique lineups performed by 443 participants. A summary of the factors used in the experiment can be found in Table 2.2. There were four levels of the non-linear structure, and three levels of heteroskedastic structure. The signal strength was controlled by error variance (σ) for the non-linear pattern, and by a ratio (b) parameter for the heteroskedasticity. Additionally, three levels of sample size (n) and four different fitted value distributions were incorporated.

2.4.1 Simulating Departures from Good Residuals

2.4.1.1 Non-linearity and Heteroskedasticity

Data collection period I was designed to study the ability of participants to detect non-linearity departures from residual plots. The non-linearity departure was constructed by omitting a jth order Hermite polynomial (Hermite 1864; originally by Laplace 1820) term of the predictor from the simple linear regression equation. Four different values of $j = 2, 3, 6, 18$ were chosen so that distinct shapes of non-linearity were included in the residual plots. These include "U", "S", "M" and "triple-U" shape as shown in Figure 2.3. A greater value of j will result in a curve with more turning points. It is expected that the "U" shape will be the easiest to detect, and as the shape gets more complex it will be harder to perceive in a scatterplot, particularly when there is noise. Figure 2.4 shows the "U" shape for different amounts of noise (σ).

Data collection period II was similar to period I but focuses on heteroskedasticity departures. We generated the heteroskedasticity departures by setting the variance-covariance matrix of the error term as a function of the predictor, but fitted the data with the simple linear regression model, intentionally violated the constant variance assumption. Visual patterns of heteroskedasticity are simulated using three different shapes (a = -1, 0, 1) including "left-triangle", "butterfly" and "right-triangle" shapes as displayed in Figure 2.5. Figure 2.6 shows the butterfly shape as the ratio parameter (b) is changed. More details about the simulation process are provided in Appendix A.2.

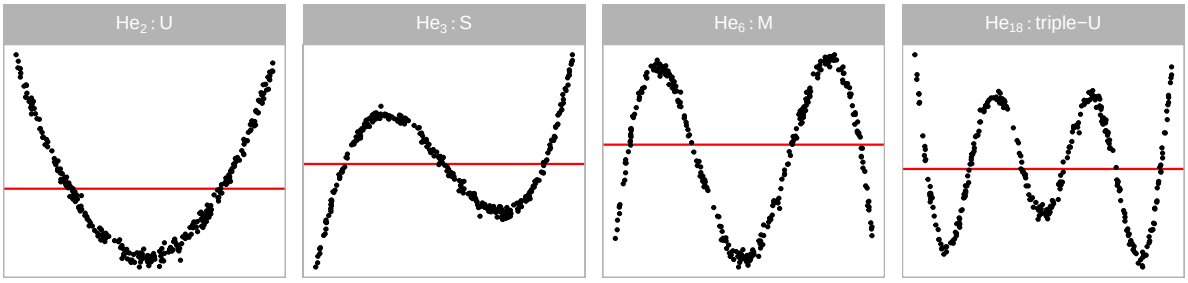


Figure 2.3: *Polynomial forms generated for the residual plots used to assess detecting non-linearity. The four shapes are generated by varying the order of polynomial given by j in $He_j(.)$.*

2.4.1.2 Factors Common to both Data Collection Periods

Fitted values are a function of the independent variables (or predictors), and the distribution of the observed values affects the distribution of the fitted values. Ideally, we would want the fitted values to have a uniform coverage across the range of observed values or have a uniform distribution across all of the predictors. This is not always present in the collected data. Sometimes the fitted values are discrete because one or more predictors were measured discretely. It is also common to see a skewed distribution of fitted values if one or more of the predictors has a skewed distribution. This latter problem is usually corrected before modelling, using a variable transformation. Our simulation

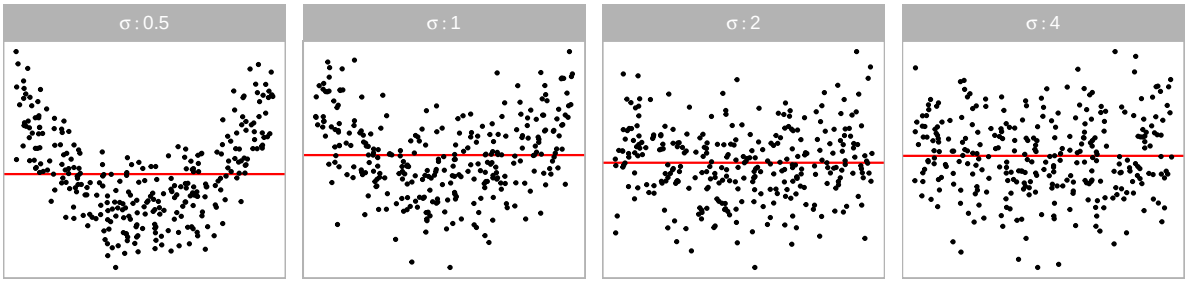


Figure 2.4: *Examining the effect of σ on the signal strength in the non-linearity detection, for $n = 300$, uniform fitted value distribution and the "U" shape. As σ increases the signal strength decreases, to the point that the "U" is almost unrecognisable when $\sigma = 4$.*

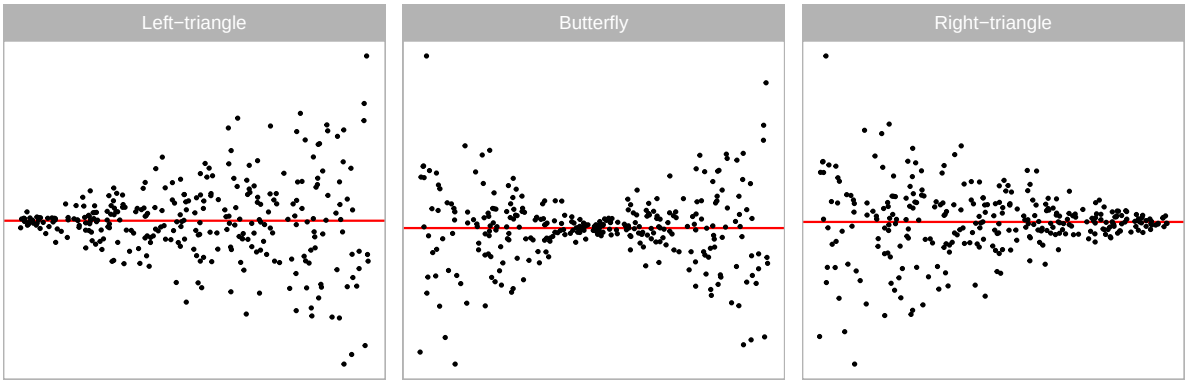


Figure 2.5: *Heteroskedasticity forms used in the experiment. Three different shapes ($a = -1, 0, 1$) are used in the experiment to create "left-triangle", "butterfly" and "right-triangle" shapes, respectively.*

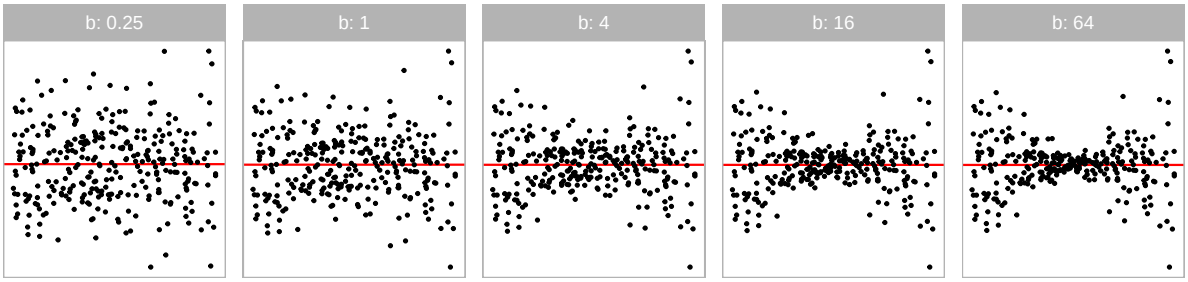


Figure 2.6: *Five different values of b are used in heteroskedasticity simulation to control the strength of the signal. Larger values of b yield a bigger difference in variation, and thus stronger heteroskedasticity signal.*

assess this by using four different distributions to represent fitted values, constructed by different sampling of the predictor, including $U(-1, 1)$ (uniform), $N(0, 0.3^2)$ (normal), lognormal$(0, 0.6^2)/3$ (skewed) and $U\{-1, 1\}$ (discrete).

Figure 2.7 shows the non-linear pattern, a "U" shape, with the different fitted value distributions. We would expect that structure in residual plots would be easier to perceive when the fitted values are uniformly distributed.

Three different sample sizes were used in our experiment: $n = 50, 100, 300$. Figure 2.8 shows the non-linear "S" shape for different sample sizes. We expect signal strength to decline in the simulated data plots with smaller n. We chose 300 as the upper limit, because it is typically enough for structure to be visible in a scatterplot reliably. Beyond 300, the scatterplot should probably be used with transparency or replaced with a density or binned plot as scatterplots suffer from over-plotting.

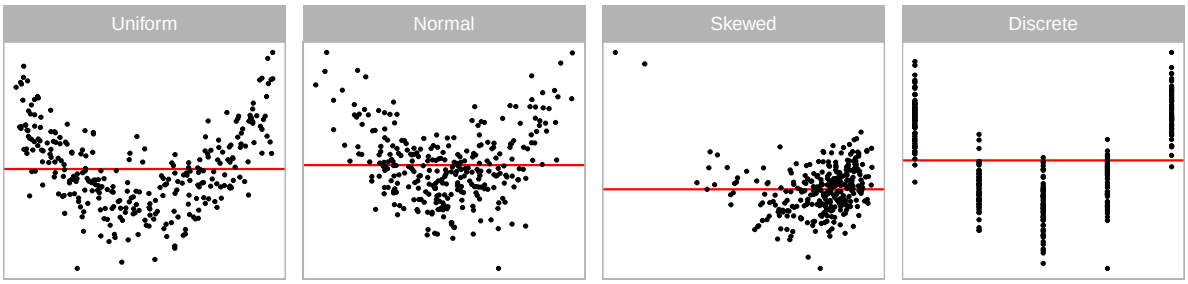


Figure 2.7: *Variations in fitted values, that might affect perception of residual plots. Four different distributions are used.*

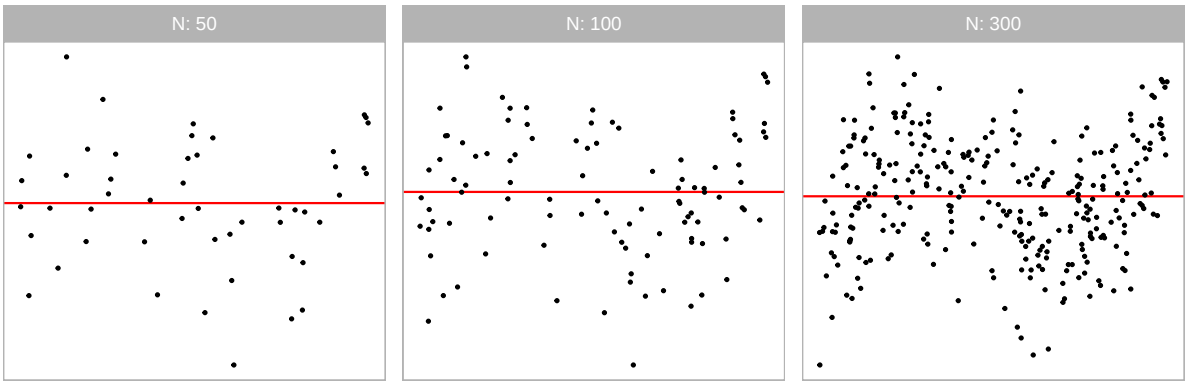


Figure 2.8: *Examining the effect of signal strength for the three different values of n used in the experiment, for non-linear structure with fixed $\sigma = 1.5$, uniform fitted value distribution, and "S" shape. For these factor levels, only when $n = 300$ is the "S" shape clearly visible.*

2.4.2 Effect Size

The lineups are allocated to participants in a manner that uniformly covers the combination of experimental factors in Table 2.2. In addition, we use effect size to measure the signal strength, which helps in assigning a set of lineups with a range of difficulties to each participant.

Effect size in statistics measures the strength of the signal relative to the noise. It is surprisingly difficult to quantify, even for simulated data as used in this experiment.

For the non-linearity model, the key items defining effect size are sample size (n) and the noise level (σ^2), and so effect size would be roughly calculated as $\sqrt{n}/\sigma$. Increasing sample size tends to boost the effect size, while heightened noise diminishes it. However, it is not clear how the additional parameter for the model polynomial order, j, should be incorporated. Intuitively, the

large j means more complex pattern, which likely means effect size would decrease. Similarly, in the heteroskedasticity model, effect size relies on sample size (n) and the ratio of the largest to smallest variance, b. Larger values of both would produce higher effect size, but the role of the additional shape parameter, a, in this context is unclear.

For the purposes of our calculations we have chosen to use an approach based on Kullback-Leibler divergence (Kullback and Leibler 1951). This formulation defines effect size to be

$$E = \frac{1}{2}\left(\log\frac{|\text{diag}(\boldsymbol{RVR}')|}{|\text{diag}(\boldsymbol{R}\sigma^2)|} - n + \text{tr}(\text{diag}(\boldsymbol{RVR}')^{-1}\text{diag}(\boldsymbol{R}\sigma^2)) + \boldsymbol{\mu}_z'(\boldsymbol{RVR}')^{-1}\boldsymbol{\mu}_z\right),$$

where diag(.) is the diagonal matrix constructed from the diagonal elements of a matrix, $\boldsymbol{X}$ is the design matrix, $\boldsymbol{V}$ is the actual covariance matrix of the error term, $\boldsymbol{R} = \boldsymbol{I}_n - \boldsymbol{X}(\boldsymbol{X}'\boldsymbol{X})^{-1}\boldsymbol{X}'$ is the residual operator, $\boldsymbol{\mu}_z = \boldsymbol{RZ}\boldsymbol{\beta}_z$ is the expected values of residuals where $\boldsymbol{Z}$ contains any higher order terms of $\boldsymbol{X}$ left out of the regression equation, $\boldsymbol{\beta}_z$ contains the corresponding coefficients, and $\sigma^2\boldsymbol{I}_n$ is the assumed covariance matrix of the error term when H_0 is true. More details about the effect size derivation are provided in Appendix A.1.

2.5 Results

Data collection used a total of 1152 lineups, and resulted in a total of 7974 evaluations from 443 participants. Roughly half corresponded to the two models, non-linearity and heteroskedasticiy, and the three collection periods had similar numbers of evaluations. Each participant received two of the 24 attention check lineups which were used to filter results of participants who were clearly not making an honest effort (only 11 of 454). To estimate α for calculating statistical significance (see Appendix A.1) there were 720 evaluations of 36 null lineups. Neither the attention checks nor null lineups were used in the subsequent analysis. The de-identified data, `vi_survey`, is made available in the R package, `visage`.

The data was collected on lineups constructed from four different fitted value distributions that stem from the corresponding predictor distribution: uniform, normal, skewed and discrete. Henceforth, we refer to these four different fitted value distributions with respect to their predictor distribution. More data was collected on the uniform distribution (each evaluated by 11 participants) than the others (each evaluated by 5 participants). The analysis in Section 2.5.1–Section 2.5.4 uses only results from lineups with uniform distribution, for a total 3069 lineup evaluations. This allows us to compare the conventional and visual test performance in an optimal scenario. Section 2.5.5 examines how the results may be affected if the fitted value distribution was different.

2.5.1 Power Comparison of the Tests

Figure 2.9 present the power curves of various tests plotted against the effect size in the residuals for non-linearity and heteroskedasticity. In each case the power of visual test is calculated for multiple bootstrap samples leading to the many (solid orange) curves. The effect size was computed at a 5% significance level and plotted on a natural logarithmic scale. To facilitate visual calibration of effect size values with the corresponding diagnostic plots, a sequence of example residual plots with increasing effect sizes is provided at the bottom of these figures. These plots serve as a visual aid to help readers understand how different effect size values translate to changes in the diagnostic plots. The horizontal lines of dots at 0 and 1 represent the non-rejection or rejection decisions made by visual tests for each lineup.

Figure 2.9A compares the power for the different tests for non-linear structure in the residuals. The test with the uniformly higher power is the RESET test, one that specifically tests for non-linearity. Note that the BP and SW tests have much lower power, which is expected because they are not designed to detect non-linearity. The bootstrapped power curves for the visual test are effectively a right shift from that of the RESET test. This means that the RESET test will reject at a lower effect size (less structure) than the visual test, but otherwise the performance will be similar. In other words, the RESET test is more sensitive than the visual test. This is not necessarily a good feature for the purposes of diagnosing model defects: if we scan the residual plot examples at the bottom, we might argue that the non-linearity is not sufficiently problematic until an effect size of around 3 or 3.5. The RESET test would reject closer to an effect size of 2, but the visual test would reject closer to 3.25, for a significance level of 0.05. The visual test matches the robustness of the model to (minor) violations of assumptions much better.

For the heteroskedasticity pattern, the power of BP test, designed for detecting heteroskedasticity, is uniformly higher than the other tests. The visual test power curve shifts to the right. This shows a similar story to the power curves for non-linearity pattern: the conventional test is more sensitive than the visual test. From the example residual plots at the bottom we might argue that the heteroskedasticity becomes noticeably visible around an effect size of 3 or 3.5. However the BP test would reject at around effect size 2.5. Interestingly, the power curve for the SW test (for non-normality) is only slightly different to that of the visual test, suggesting that it performs reasonably well for detecting heteroskedasticity, too. The power curve for the BP test suggests it is not useful for detecting heteroskedasticity, as expected.

Overall, the results show that the conventional tests are more sensitive than the visual test. The conventional tests do have higher power for the patterns they are designed to detect, but they typically

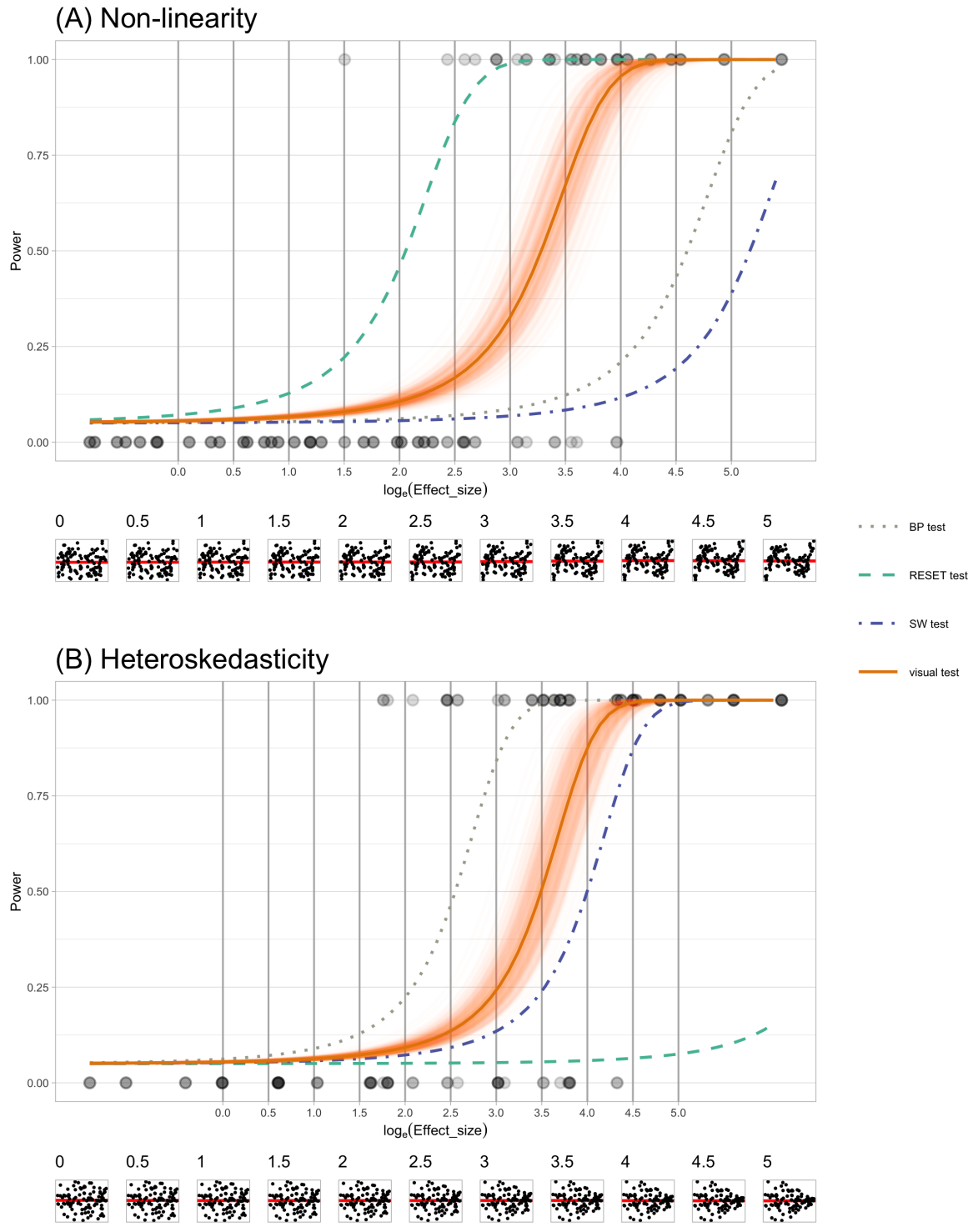


Figure 2.9: *Comparison of power between different tests for (A) non-linear and (B) heteroskedasticity patterns (uniform fitted values only). Main plot shows the power curves, with dots indicating non-reject and reject in visual testing of lineups. The multiple lines for the visual test arise from estimating the power on many bootstrap samples. The row of scatterplots at the bottom are examples of residual plots corresponding to the specific effect sizes marked by vertical lines in the main plot.*

fail to detect other patterns unless those patterns are particularly strong. The visual test does not require specifying the pattern ahead of time, relying purely on whether the observed residual plot is detectably different from "good" residual plots. They will perform equally well regardless of the type of model defect. This aligns with the advice of experts on residual analysis, who consider residual plot analysis to be an indispensable tool for diagnosing model problems. What we gain from using a visual test for this purpose is the removal of any subjective arguments about whether a pattern is visible or not. The lineup protocol provides the calibration for detecting patterns: if the pattern in the data plot cannot be distinguished from the patterns in good residual plots, then no discernible problem with the model exists.

2.5.2 Comparison of Test Decisions Based on p-values

The power comparison demonstrates that the appropriate conventional tests will reject more aggressively than visual tests, but we do not know how the decisions for each lineup would agree or disagree. Here we compare the reject or fail to reject decisions of these tests, across all the lineups. Figure 2.10 shows the agreement of the conventional and visual tests using a mosaic plot for both non-linearity patterns and heteroskedasticity patterns. For both patterns the lineups resulting in a rejection by the visual test are *all* also rejected by the conventional test, except for one from the heteroskedasticity model. This reflects exactly the story from the previous section, that the conventional tests reject more aggressively than the visual test.

For non-linearity lineups, conventional tests and visual tests reject 69% and 32% of the time, respectively. Of the lineups rejected by the conventional test, 46% are rejected by the visual test, that is, approximately half as many as the conventional test. There are no lineups that are rejected by the visual test but not by the conventional test.

In heteroskedasticity lineups, 76% are rejected by conventional tests, while 56% are rejected by visual tests. Of the lineups rejected by the conventional test, the visual test rejects more than two-thirds of them, too.

Surprisingly, the visual test rejects 1 of the 33 (3%) of lineups where the conventional test does not reject. Figure 2.11 shows this lineup. The data plot in position seventeen displays a relatively strong heteroskedasticity pattern, and has a strong effect size ($\log_e(E) = 4.02$), which is reflected by the visual test p-value $= 0.026$. But the BP test p-value $= 0.056$, is slightly above the significance cutoff of 0.05. This lineup was evaluated by 11 participants, it has experimental factors $a = 0$ ("butterfly" shape), $b = 64$ (large variance ratio), $n = 50$ (small sample size), and a uniform distribution for the predictor. It may have been the small sample size and the presence of a few outliers that may have resulted in the lack of detection by the conventional test.

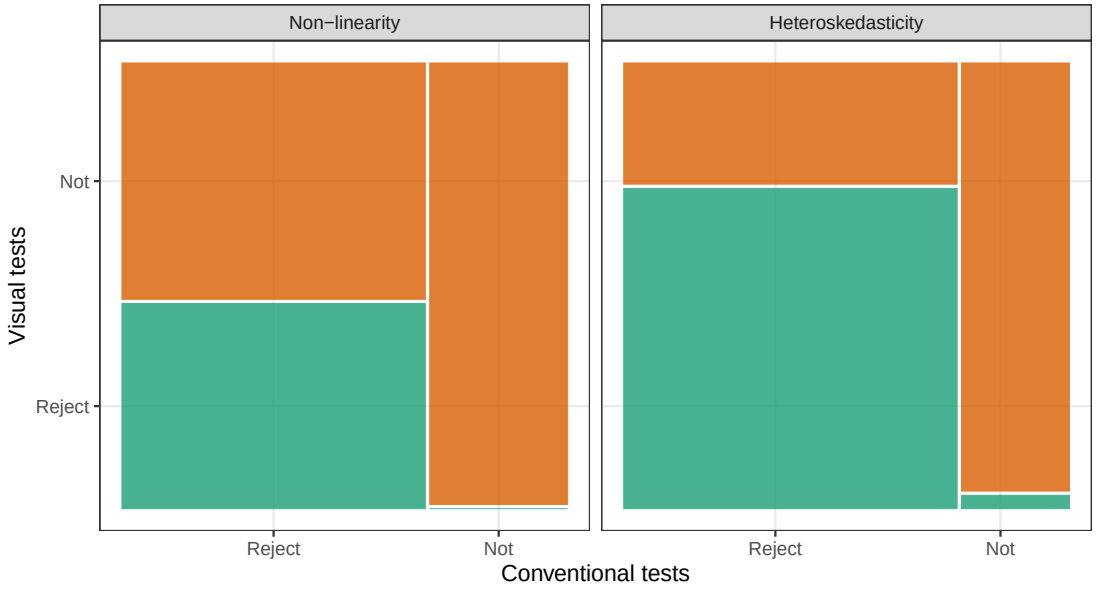


Figure 2.10: *Rejection rate (p-value ≤ 0.05) of visual test conditional on the conventional test decision on non-linearity (left) and heteroskedasticity (right) lineups (uniform fitted values only) displayed using a mosaic plot. The visual test rejects less frequently than the conventional test, and (almost) only rejects when the conventional test does. Surprisingly, one lineup in the heteroskedasticity group is rejected by the visual test but NOT the conventional test.*

Because the power curve of the visual tests are a shift to the right of the conventional test (Figure 2.9) we examined whether adjusting the significance level (to .001, .0001, .00001, ...) of the conventional test would generate similar decisions to that of the visual test. Interestingly, it does not: despite resulting in less rejections, neither the RESET or BP tests come to complete agreement with the visual test (see Appendix A.1).

2.5.3 Effect of Amount of Non-linearity

The order of the polynomial is a primary factor contributing to the pattern produced by the non-linearity model. Figure 2.12 explores the relationship between polynomial order and power of the tests. The conventional tests have higher power for lower orders of Hermite polynomials, and the power drops substantially for the "triple-U" shape. To understand why this is, we return to the application of the RESET test, which requires a parameter indicating degree of fitted values to test for, and the recommendation is to generically use four (Ramsey 1969). However, the "triple-U" shape is constructed from the Hermite polynomials using power up to 18. If the RESET test had been applied using a higher power of no less than six, the power curve of "triple-U" shape will be closer to other power curves. This illustrates the sensitivity of the conventional test to the parameter choice, and highlights a limitation: it helps to know the data generating process to set the parameters for the test, which is unrealistic in practice. However, we examined this in more detail (see Appendix A.1) and found that there is no harm in setting the parameter higher than four on the tests' operation for

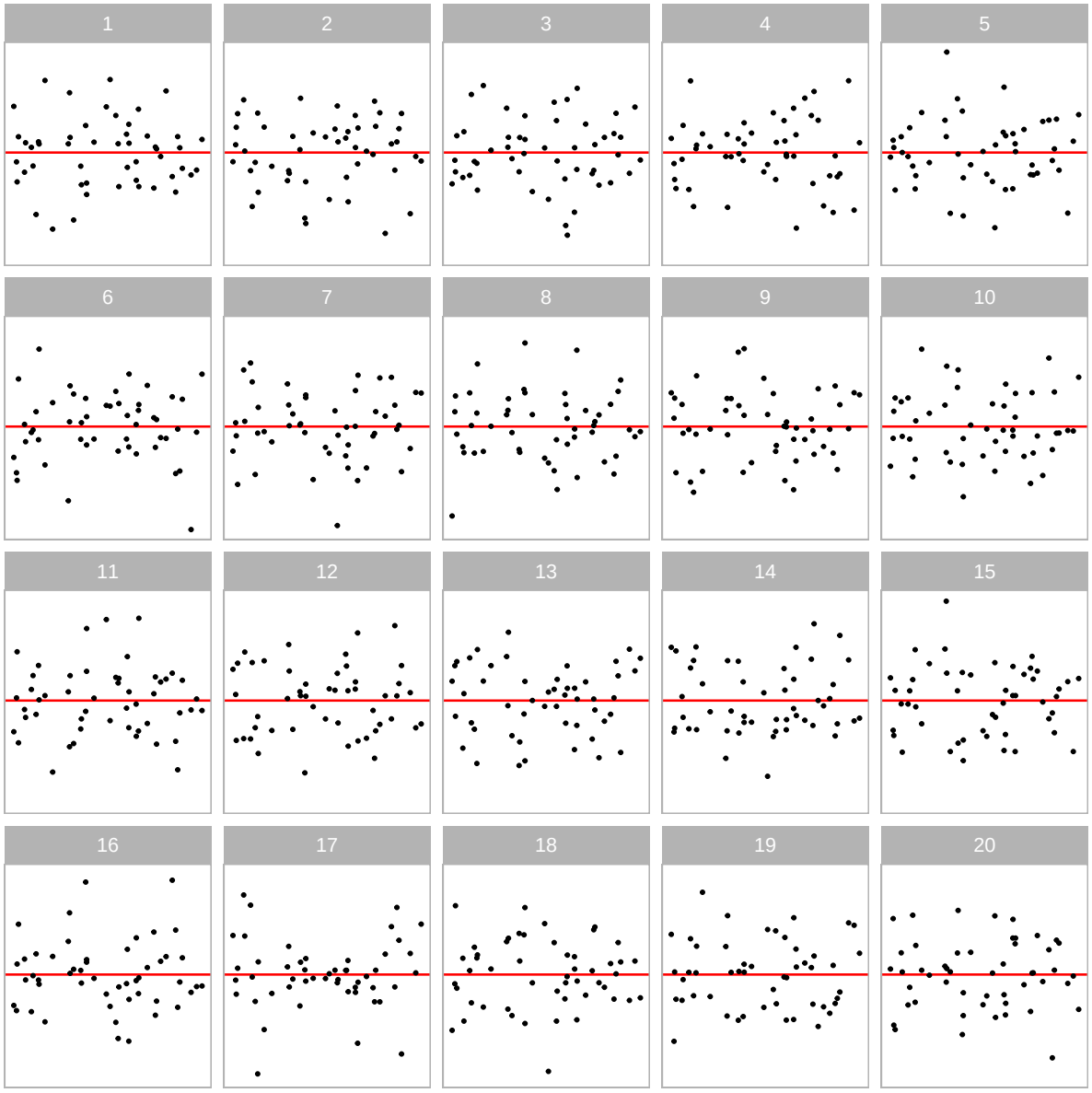


Figure 2.11: *The single heteroskedasticity lineup that is rejected by the visual test but not by the BP test. The data plot (position 17) contains a "butterfly" shape. It visibly displays heteroskedasticity, making it somewhat surprising that it is not detected by the BP test.*

lower order polynomial shapes. Using a parameter value of six, instead of four, yields higher power regardless of generating process, and is recommended.

For visual tests, we expect the "U" shape to be detected more readily, followed by the "S", "M" and "triple-U" shape. From Figure 2.12, it can be observed that the power curves mostly align with these expectations, except for the "M" shape, which is as easily detected as the "S" shape. This suggests a benefit of the visual test: knowing the shape ahead of time is *not* needed for its application.

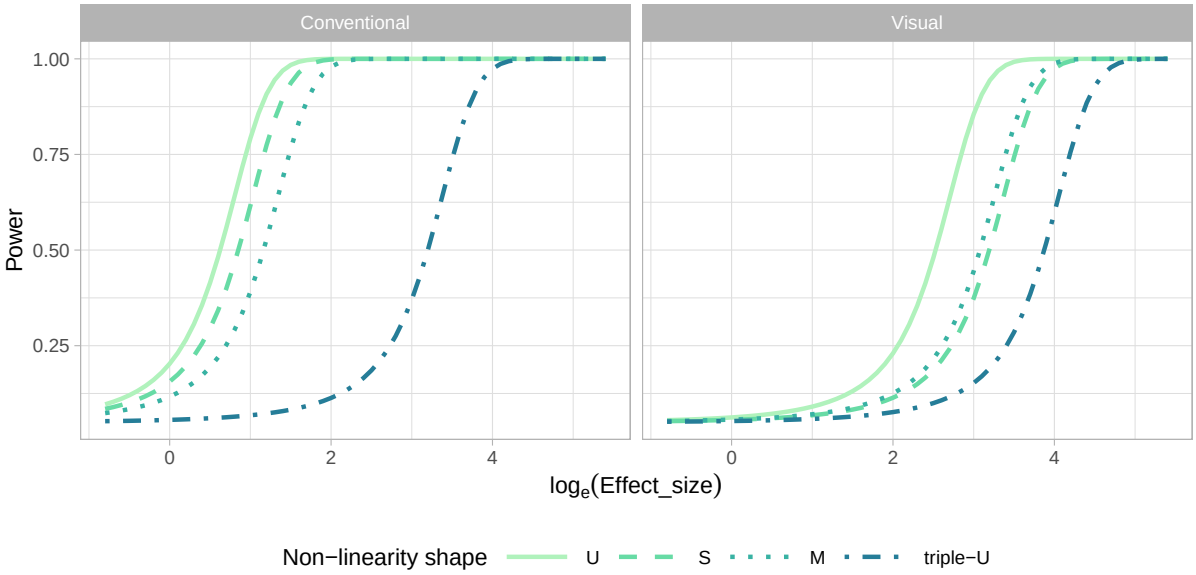


Figure 2.12: *The effect of the order of the polynomial on the power of conventional and visual tests. Deeper colour indicates higher order. The default RESET tests under-performs significantly in detecting the "triple-U" shape. To achieve a similar power as other shapes, a higher order polynomial parameter needs to be used for the RESET test, but this higher than the recommended value.*

2.5.4 Effect of Shape of Heteroskedasticity

Figure 2.13 examines the impact of the shape of the heteroskedasticity on the power of of both tests. The butterfly shape has higher power on both types of tests. The "left-triangle" and the "right-triangle" shapes are functionally identical, and this is observed for the conventional test, where the power curves are identical. Interestingly there is a difference for the visual test: the power curve of the "left-triangle" shape is slightly higher than that of the "right-triangle" shape. This indicates a bias in perceiving heteroskedasticity depending on the direction, and may be worth investigating further.

2.5.5 Effect of Fitted Value Distributions

In regression analysis, predictions are conditional on the observed values of the predictors, that is, the conditional mean of the dependent variable Y given the value of the independent variable X, $\mathrm{E}(Y|X)$. This is an often forgotten element of regression analysis but it is important. Where X is observed, the distribution of the X values in the sample, or consequently $\hat{Y}$, may affect the ability to read any patterns in the residual plots. The effect of fitted value distribution on test performance is assess using four different distributions of fitted values stemming from the predictor distributions: uniform, normal, discrete and lognormal (skewed). We expect that if all predictors have a uniform distribution, it is easier to read the relationship with the residuals.

Figure 2.14 examines the impact of the fitted value distribution on the power of conventional (left) and visual (right) tests for both the non-linearity (top) and heteroskedasticity (bottom) patterns. For

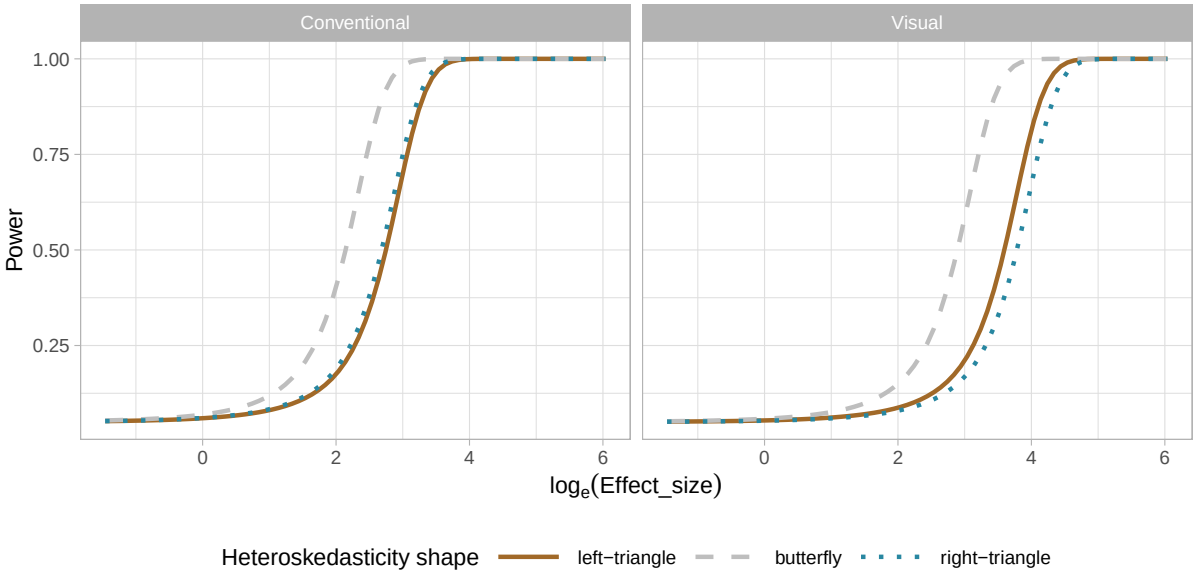


Figure 2.13: *The effect of heteroskedasticity shape (parameter a) on the power of conventional and visual tests. The butterfly has higher power in both tests. Curiously, the visual test has a slightly higher power for the "left-triangle" than the "right-triangle" shape, when it would be expected that they should be similar, which is observed in conventional testing.*

conventional tests, only the power curves of appropriate tests are shown: RESET tests for non-linearity and BP tests for heteroskedasticity. For visual tests, more evaluations on lineups with uniform fitted value distribution were collected, so to have a fair comparison, we randomly sample five from the 11 total evaluations to estimate the power curves, producing the multiple curves for the uniform condition, and providing an indication of the variability in the power estimates.

Perhaps surprisingly, the visual tests have more consistent power across the different fitted value distributions: for the non-linear pattern, there is almost no power difference, and for the heteroskedastic pattern, uniform and discrete have higher power than normal and skewed. The likely reason is that these latter two have fewer observations in the tails where the heteroskedastic pattern needs to be detected.

The variation in power in the conventional tests is at first sight, shocking. However, it is discussed, albeit rarely, in the testing literature. See, for example, Jamshidian et al. (2007), Olvera Astivia et al. (2019) and Zhang and Yuan (2018) which show derivations and use simulation to assess the effect of the observed distribution of the predictors on test power. The big differences in the power curves seen in Figure 2.14 is echoed in the results reported in these articles.

2.6 Limitations and Practicality

One of the primary limitations of the lineup protocol lies in its reliance on human judgements. In this context, the effectiveness of a single lineup evaluation can be dependent on the perceptual ability

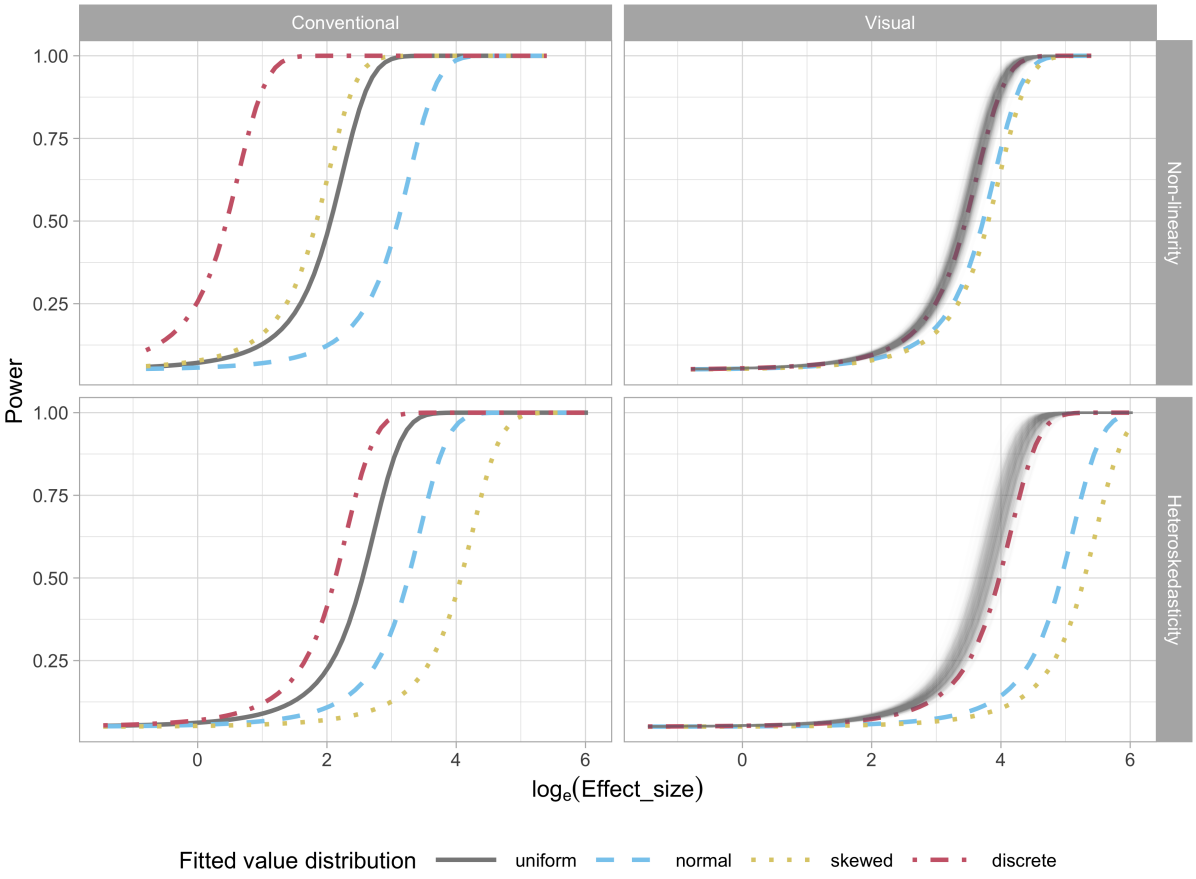


Figure 2.14: *Comparison of power on lineups with different fitted value distributions for conventional and visual tests (columns) for non-linearity and heteroskedasticity patterns (rows). The power curves of conventional tests for non-linearity and heteroskedasticity patterns are produced by RESET tests and BP tests, respectively. Power curves of visual tests are estimated using five evaluations on each lineup. For lineups with a uniform fitted value distribution, the five evaluations are repeatedly sampled from the total eleven evaluations to give multiple power curves (solid grey). Surprisingly, the fitted value distribution has produces more variability in the power of conventional tests than visual tests. Uneven distributions, normal and skewed distributions, tend to yield lower power.*

and visual skills of the individual. However, when results from multiple individuals are combined the outcome is encouragingly high-quality and robust. For simple plots and strong patterns just a few individuals are needed to arrive at a clear answer, but more individuals will be needed when the plot design is complex, or the signal strength is weak.

Using a lineup protocol removes subjectiveness in interpreting patterns in plots. A plot is compared with draws from a null model, in much the same way as a test statistic is compared to its sampling distribution. It is important to remove plot elements that might introduce bias, such as axis labels, text and legends, or to make them generic.

The lineup protocol can be used cheaply and informally with the R package `nullabor`. There is evidence that it is being used fairly broadly, based on software download rates and citations of the

original papers. For residual plot analysis we recommend that the lineup be the default first plot so that the data plot is only seen in the context of null plots. When a rigorous test is needed, we recommend using a crowd-sourcing service, as done in gene expression experiment described in Yin et al. (2013). While it takes extra effort it is not difficult today, and costs are tiny compared to the overall costs of conducting a scientific experiment. We do also expect that at some point a computer vision model can be developed to take over the task of employing people to evaluate residual plots.

For this study, simulated data was used to provide a precisely controlled environment within which to compare results from conventional testing to those from visual testing. We also explored only the most commonly used, the residual vs fitted value plots. However, we expect the behaviour of the conventional test and the visual test to be similar when observed residuals are diagnosed with this type of plot or other residual plots. The conventional tests will be more sensitive to small departures from the null. They will also fail to detect departures when residuals have some contamination, like outliers or anomalies, as is often encountered when working with data. The lineup approach is well-suited for generally interpreting data plots, and also detecting unexpected patterns not related to the model. This is supported by earlier research (e.g. Loy et al. 2016; Loy and Hofmann 2015; Roy Chowdhury et al. 2015; VanderPlas and Hofmann 2016; Wickham et al. 2010).

2.7 Conclusions

This paper has described experimental evidence providing support for the advice of regression analysis experts *that residual plots are indispensable methods for assessing model fit*, using the formal framework of the lineup protocol. We conducted a perceptual experiment on scatterplots of residuals vs fitted values, with two primary departures from good residuals: non-linearity and heteroskedasticity. We found that conventional residual-based statistical tests are more sensitive to weak departures from model assumptions than visual tests. That is, a conventional test concludes there are problems with the model fit almost twice as often as a human. Conventional tests often reject the null hypothesis when departures in the form of non-linearity and heteroskedasticity are not visibly different from null residual plots.

While it might be argued that the conventional tests are correctly detecting small but real effects, this can also be seen as the conventional tests are rejecting unnecessarily. Many of these rejections happen even when downstream analysis and results would not be significantly affected by the small departures from a good fit. The results from human evaluations provide a more practical solution, which reinforces the statements from regression experts that residual plots are an indispensable method for model diagnostics. Further work would be needed to *quantify* how much departure from good residuals is too much.

It is important to emphasize that this work also supports a change in common practice, which is to deliver residual plots as a lineup, embedded in a field of null plots, rather than be viewed out of context. A residual plot may contain many visual features, but some are caused by the characteristics of the predictors and the randomness of the error, not by the violation of the model assumptions. These irrelevant visual features have a chance to be filtered out by participants with a comparison to null plots, resulting in more accurate reading. The lineup enables a careful calibration for reading structure in residual plots, and also provides the potential to discover interesting and important features in the data not directly connected to linear model assumptions.

Human evaluation of residuals is expensive, time-consuming and laborious. This is possibly why residual plot analysis is often not done in practice. However, with the emergence of effective computer vision, it is hoped this work helps to lay the foundation for automated residual plot assessment.

The experiment also revealed some interesting results. For the most part, the visual test performed similarly to the appropriate conventional test with a shift in the power curve. Unlike conventional tests, where one needs to specifically test for non-linearity or heteroskedasticity the visual test operated effectively across the range of departures from good residuals. If the fitted value distribution is not uniform, there is a small loss of power in the visual test. Surprisingly, there is a big difference in power of the conventional test across fitted value distributions. Another unexpected finding was that the direction of heteroskedasticity appears to affect the ability to visually detect it: both triangles being more difficult to detect than the butterfly, and a small difference in detection between left- and right-triangle.

Chapter 3

Automated Assessment of Residual Plots with Computer Vision Models

Plotting residuals is a recommended practice to diagnose deviations from linear model assumptions such as non-linearity, heteroscedasticity, and non-normality. The presence or absence of structure in residual plots can be tested using the lineup protocol to do visual inference. As a statistical test, the lineup protocol is less sensitive and applies more broadly than available conventional tests. However, the lineup protocol relies on human judgment which limits its scalability. This work presents a solution by providing a computer vision model to automate the assessment of residual plots. It is trained to predict the distance measure that quantifies the disparity between the residual distribution of a fitted classical normal linear regression model and the reference distribution, based on Kullback-Leibler divergence. From extensive simulation studies, the computer vision model exhibits lower sensitivity than conventional tests but higher sensitivity than human visual tests. It is slightly less effective on non-linearity patterns. Several examples from classical papers and contemporary data illustrate the new procedures, highlighting its usefulness in automating the diagnostic process and supplementing existing methods.

3.1 Introduction

Plotting residuals is commonly regarded as a standard practice in linear regression diagnostics (Belsley et al. 1980; Cook and Weisberg 1982). This visual assessment plays a crucial role in identifying whether model assumptions, such as linearity, homoscedasticity, and normality, are reasonable. It also helps in understanding the goodness of fit and various unexpected characteristics of the model.

Generating a residual plot in most statistical software is often as straightforward as executing a line

of code or clicking a button. However, accurately interpreting a residual plot can be challenging. A residual plot can exhibit various visual features, but it is crucial to recognize that some may arise from the characteristics of predictors and the natural stochastic variation of the observational unit, rather than indicating a violation of model assumptions (Li et al. 2024). Consider Figure 3.1 as an example, the residual plot displays a triangular left-pointing shape. The distinct difference in the spread of the residuals across the fitted values may result in the analyst suggesting that there may be heteroskedasticity, however, it is important to avoid over-interpreting this visual pattern. In this case, the fitted regression model is correctly specified, and the triangular shape is actually a result of the skewed distribution of the predictors, rather than indicating a flaw in the model.

The concept of visual inference, as proposed by Buja et al. (2009b), provides an inferential framework to assess whether residual plots indeed contain visual patterns inconsistent with the model assumptions. The fundamental idea involves testing whether the true residual plot visually differs significantly from null plots, where null plots are plotted with residuals generated from the residual rotation distribution (Langsrud 2005), which is a distribution consistent with the null hypothesis H_0 that the linear regression model is correctly specified. Typically, the visual test is accomplished through the lineup protocol, where the true residual plot is embedded within a lineup alongside several null plots. If the true residual plot can be distinguished from the lineup, it provides evidence for rejecting H_0.

The practice of delivering a residual plot as a lineup is generally regarded as a valuable approach. Beyond its application in residual diagnostics, the lineup protocol has been integrated into the analysis of diverse subjects. For instance, Loy and Hofmann (2013, 2014, 2015) illustrated its applicability in diagnosing hierarchical linear models. Additionally, Widen et al. (2016) and Fieberg et al. (2024) demonstrated its utility in geographical and ecology research respectively, while Krishnan and Hofmann (2021) explored its effectiveness in forensic examinations.

A practical limitation of the lineup protocol lies in its reliance on human judgements (see Li et al. 2024 about the practical limitations). Unlike conventional statistical tests that can be performed computationally in statistical software, the lineup protocol requires human evaluation of images. This characteristic makes it less suitable for large-scale applications, given the associated high labour costs and time requirements. There is a substantial need to develop an approach to substitute these human judgement with an automated reading of data plots using machines.

The utilization of computers to interpret data plots has a rich history, with early efforts such as "Scagnostics" by Tukey and Tukey (1985), a set of numerical statistics that summarise features of scatter plots. Wilkinson et al. (2005) expanded on this work, introducing scagnostics based on

computable measures applied to planar proximity graphs. These measures, including, but not limited to, “Outlying”, “Skinny”, “Stringy”, “Straight”, “Monotonic”, “Skewed”, “Clumpy”, and “Striated”, aimed to characterize outliers, shape, density, trend, coherence and other characteristics of the data. While this approach has been inspiring, there is a recognition (Buja et al. 2009b) that it may not capture all the necessary visual features that differentiate true residual plots from null plots. A more promising alternative entails enabling machines to learn the function for extracting visual features from residual plots. Essentially, this means empowering computers to discern the crucial visual features for residual diagnostics and determining the method to extract them.

Modern computer vision models are well-suited for addressing this challenge. They rely on deep neural networks with convolutional layers (Fukushima and Miyake 1982). These layers leverage hierarchical patterns in data, downsizing and transforming images by summarizing information in a small space. Numerous studies have demonstrated the efficacy of convolutional layers in addressing various vision tasks, including image recognition (Rawat and Wang 2017). Despite the widespread use of computer vision models in fields like computer-aided diagnosis (Lee and Chen 2015), pedestrian detection (Brunetti et al. 2018), and facial recognition (Emami and Suciu 2012), their application in reading data plots remains limited. While some studies have explored the use of computer vision models for tasks such as reading recurrence plots for time series regression (Ojeda et al. 2020), time series classification (Chu et al. 2019; Hailesilassie 2019; Hatami et al. 2018; Zhang et al. 2020), anomaly detection (Chen et al. 2020), and pairwise causality analysis (Singh et al. 2017), the application of reading residual plots with computer vision models is a new field of study.

In this chapter, we develop computer vision models and integrate them into the residual plots diagnostics workflow, addressing the need for an automated visual inference. The chapter is structured as follows. Section 3.2 discusses various specifications of the computer vision models. Section 3.3 defines the distance measure used to detect model violations, while Section 3.4 explains how the computer vision models estimate this distance measure. Section 3.5 covers the statistical tests based on the estimated distance, and Section 3.6 introduces a Model Violations Index, which offers a quicker and more convenient assessment. Sections 3.7, 3.8, and 3.9 detail the data preparation, model architecture, and training process, respectively. The results are presented in Section 3.10. Example dataset applications are discussed in Section 3.11. Finally, we conclude with a discussion of our findings and propose ideas for future research directions.

3.2 Model Specifications

There are various specifications of the computer vision model that can be used to assess residual plots. We discuss these specifications below focusing on two key components of the model formula: the

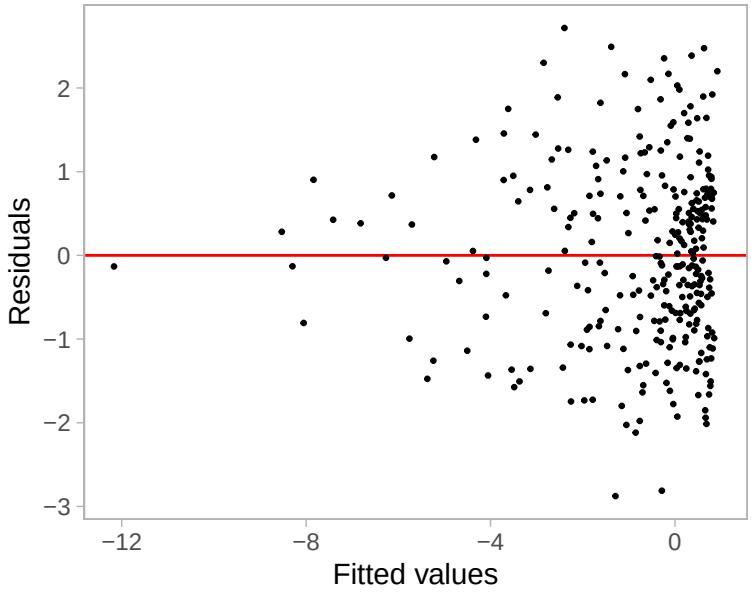


Figure 3.1: *An example residual vs fitted values plot (red line indicates 0 corresponds to the x-intercept, i.e. $y = 0$). The vertical spread of the data points varies with the fitted values. This often indicates the existence of heteroskedasticity, however, here the result is due to skewed distribution of the predictors rather than heteroskedasticity. The Breusch-Pagan test rejects this residual plot at 95% significance level (p-value $= 0.046$).*

input and the output format.

3.2.1 Input Formats

Deep learning models are in general very sensitive to the input data. The quality and relevance of the input data greatly influence the model's capacity to generate insightful and meaningful results. There are several designs of the input format that can be considered.

A straightforward design involves feeding a vector of residuals along with a vector of fitted values, essentially providing all the necessary information for creating a residuals vs fitted values plot. However, a drawback of this method is the dynamic input size, which changes based on the number of observations. For modern computer vision models implemented in mainstream software like TensorFlow (Abadi et al. 2016), the input shape is typically fixed. One solution is to pad the input vectors with leading or trailing zeros when the input tensor expects longer vectors, but it may fail if the input vector surpasses the designed length.

Another strategy is to summarize the residuals and fitted values separately using histograms and utilize the counts as the input. By controlling the number of bins in the histograms, it becomes possible to provide fixed-length input vectors. Still, since histograms only capture the marginal distribution of residuals and fitted values respectively, they can not be used to differentiate visual patterns with same marginal distributions but different joint distributions.

Another design involves using an image as input. The primary advantage of this design, as opposed to the vector format, is the availability of the existing and sophisticated image processing architectures developed over the years, such as the VGG16 architecture proposed in Simonyan and Zisserman (2014). These architectures can effectively capture and summarize spatial information from nearby pixels, which is less straightforward with vector input. The main considerations are the image resolution and the aesthetics of the residual plot. In general, a higher resolution provides more information to the model but comes with a trade-off of increased complexity and greater difficulty in training. As for the aesthetics of the residual plot, a practical solution is to consistently present residual plots in the same style to the model. This implies that the model can not accept arbitrary images as input but requires the use of the same pre-processing pipeline to convert residuals and fitted values into a standardized-style residual plot.

Providing multiple residual plots to the model, such as a pair of plots, a triplet or a lineup is also a possible option. Chopra et al. (2005) have shown that computer vision models designed for image comparison can assess whether a pair of images are similar or dissimilar. Applied to our specific problem, we can define null plots of a fitted regression model to be similar to each other, while considering true residual plots to be distinct from null plots of any fitted regression model. A triplet constitutes a set of three images, denoted as $image_1$, $image_2$ and $image_3$. It is often used to predict whether $image_2$ or $image_3$ is more similar to $image_1$, proving particularly useful for establishing rankings between samples. For this setup, we can apply the same criteria to define similarity between images. However, it is important to note that these two approaches usually require additional considerations regarding the loss function and, at times, non-standard training processes due to shared weights between different convolutional blocks.

Presenting a lineup to a model aligns closely with the lineup protocol. However, as the number of residual plots in a lineup increases, the resolution of the input image grows rapidly, posing challenges in training the model. We experimented with this approach in a pilot study, but the performance of the trained model was sub-optimal.

Taking into account the implementation cost and the need for model interpretability, we used the single residual plot input format in this chapter.

3.2.2 Output Formats

Given that the input is a single residual plot represented as a fixed-resolution image, we can choose the output from the computer vision model to be either binary (classification) or numeric (regression).

The binary outcome can represent whether the input image is consistent with a null plot as determined by either (1) the data generating process or (2) the result of a visual test based on human judgement.

Training a model following the latter option requires data from prior human subject experiments, presenting difficulties in controlling the quality of data due to variations in experimental settings across different studies. Additionally, some visual inference experiments are unrelated to linear regression models or residual plot diagnostics, resulting in a limited amount of available training data.

Alternatively, the output could be a meaningful and interpretable numerical measure useful for assessing residual plots, such as the strength of suspicious visual patterns reflecting the extent of model violations, or the difficulty index for identifying whether a residual plot has no issues. However, these numeric measures are often informally used in daily communication but are not typically formalized or rigorously defined. For the purpose of training a model, this numeric measure has to be quantifiable.

In this study, we chose to define and use a distance between a true residual plot and a theoretically "good" residual plot. This is further explained in Section 3.3. Vo and Hays (2016) have also demonstrated that defining a proper distance between images can enhance the matching accuracy in image search compared to a binary outcome model.

3.2.3 Auxiliary Information with Scagnostics

In Section 3.1, we mentioned that scagnostics consist of a set of manually designed visual feature extraction functions. While our computer vision model will learn its own feature extraction function during training, leveraging additional information from scagnostics can enhance the model's predictive accuracy.

For each residual plot used as an input image, we computed four scagnostics – "Monotonic", "Sparse", "Splines", and "Striped" – using the `cassowaryr` R package (Mason et al. 2022). These computed measures, along with the number of observations from the fitted model, were provided as the second input for the computer vision model. While other scagnostics provide valuable insights, they come with high computational costs and are not suitable for quick inference.

3.3 Distance from a Theoretically "Good" Residual Plot

To develop a computer vision model for assessing residual plots within the visual inference framework, it is important to precisely define a numerical measure of "difference" or "distance" between plots. This distance can take the form of a basic statistical operation on pixels, such as the sum of square differences, however, a pixel-to-pixel comparison makes little sense in comparing residual plots where the main interest would be structural patterns. Alternatively, it could involve established image similarity metrics like the Structural Similarity Index Measure (Wang et al. 2004) which compares

images by integrating three perception features of an image: contrast, luminance, and structure (related to average, standard deviation and correlation of pixel values over a window, respectively). These image similarity metrics are tailored for image comparison in vastly different tasks to evaluating data plots, where only essential plot elements require assessment (Chowdhury et al. 2018). We can alternatively define a notion of distance by integrating key plot elements (instead of key perception features like luminance, contrast, and structure), such as those captured by scagnostics mentioned in Section 3.1, but the functional form still needs to be carefully refined to accurately reflect the extent of the violations.

In this section, we introduce a distance measure between a true residual plot and a theoretically 'good' residual plot. This measure quantifies the divergence between the residual distribution of a given fitted regression model and that of a correctly specified model. The computation assumes knowledge of the data generating processes for predictors and response variables. Since these processes are often unknown in practice, we will discuss a method to estimate this distance using a computer vision model in Section 3.4.

3.3.1 Residual Distribution

For a classical normal linear regression model, $\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{e}$, the residual $\hat{\boldsymbol{e}}$ are derived as the difference of the fitted values and observed values $\boldsymbol{y}$. Suppose the data generating process is known and the regression model is correctly specified, by the Frisch-Waugh-Lowell theorem (Frisch and Waugh 1933), residuals $\hat{\boldsymbol{e}}$ can also be treated as random variables and written as a linear transformation of the error $\boldsymbol{e}$ formulated as $\hat{\boldsymbol{e}} = \boldsymbol{R}\boldsymbol{e}$, where $\boldsymbol{R} = \boldsymbol{I}_n - \boldsymbol{X}(\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top$ is the residual operator, $\boldsymbol{I}_n$ is a n by n identity matrix, and n is the number of observations.

One of the assumptions of the classical normal linear regression model is that the error $\boldsymbol{e}$ follows a multivariate normal distribution with zero mean and constant variance, i.e., $\boldsymbol{e} \sim N(\boldsymbol{0}_n, \sigma^2\boldsymbol{I}_n)$. It can be known that residuals $\hat{\boldsymbol{e}}$ also follow a certain probability distribution transformed from the multivariate normal distribution, which will be denoted as Q. This reference distribution Q summarizes what "good" residuals should follow given the design matrix $\boldsymbol{X}$ is known and fixed.

Suppose the design matrix $\boldsymbol{X}$ has linearly independent columns, the trace of the hat matrix $\boldsymbol{H} = \boldsymbol{X}(\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top$ will equal to the number of columns in $\boldsymbol{X}$ denoted as k. As a result, the rank of $\boldsymbol{R}$ is $n-k$, and Q is a degenerate multivariate distribution. To capture the characteristics of Q, such as moments, we can simulate a large numbers of $\boldsymbol{\varepsilon}$ and transform it to $\boldsymbol{e}$ to get the empirical estimates. For simplicity, in this study, we replaced the variance-covariance matrix of residuals $\text{cov}(\boldsymbol{e}, \boldsymbol{e}) = \boldsymbol{R}\sigma^2\boldsymbol{R}^\top = \boldsymbol{R}\sigma^2$ with a full-rank diagonal matrix $\text{diag}(\boldsymbol{R}\sigma^2)$, where diag(.) sets the non-diagonal entries of a matrix to zeros. The resulting distribution for Q is $N(\boldsymbol{0}_n, \text{diag}(\boldsymbol{R}\sigma^2))$.

Distribution Q is derived from the correctly specified model. However, if the model is misspecified, then the actual distribution of residuals denoted as P, will be different from Q. For example, if the data generating process contains variables correlated with any column of $\boldsymbol{X}$ but missing from $\boldsymbol{X}$, causing an omitted variable problem, P will be different from Q because the residual operator obtained from the fitted regression model will not be the same as $\boldsymbol{R}$. Besides, if the $\boldsymbol{\varepsilon}$ follows a non-normal distribution such as a multivariate lognormal distribution, P will usually be skewed and has a long tail.

3.3.2 Distance of P from Q

Defining a proper distance between distributions is usually easier than defining a proper distance between data plots. Given the true residual distribution Q and the reference residual distribution P, we used a distance measure based on Kullback-Leibler divergence (Kullback and Leibler 1951) to quantify the difference between two distributions as

$$D = \log(1 + D_{KL}), \tag{3.1}$$

where D_{KL} is defined as

$$D_{KL} = \int_{\mathbb{R}^n} \log \frac{p(\boldsymbol{e})}{q(\boldsymbol{e})} p(\boldsymbol{e}) d\boldsymbol{e}, \tag{3.2}$$

and $p(.)$ and $q(.)$ are the probability density functions for distribution P and distribution Q, respectively.

This distance measure was first proposed in Li et al. (2024). It was mainly designed for measuring the effect size of non-linearity and heteroskedasticity in a residual plot. Li et al. (2024) have derived that, for a classical normal linear regression model that omits necessary higher-order predictors $\boldsymbol{Z}$ and the corresponding parameter $\boldsymbol{\beta}_z$, and incorrectly assumes $\boldsymbol{\varepsilon} \sim N(\boldsymbol{0}_n, \sigma^2 \boldsymbol{I}_n)$ while in fact $\boldsymbol{\varepsilon} \sim N(\boldsymbol{0}_n, \boldsymbol{V})$ where $\boldsymbol{V}$ is an arbitrary symmetric positive semi-definite matrix, Q can be represented as $N(\boldsymbol{RZ\beta}_z, \text{diag}(\boldsymbol{RVR}))$. Note that the variance-covariance matrix is replaced with the diagonal matrix to ensure it is a full-rank matrix.

Since both P and Q are adjusted to be multivariate normal distributions, Equation 3.2 can be further expanded to

$$D_{KL} = \frac{1}{2}\left(\log \frac{|\boldsymbol{W}|}{|\text{diag}(\boldsymbol{R}\sigma^2)|} - n + \text{tr}(\boldsymbol{W}^{-1}\text{diag}(\boldsymbol{R}\sigma^2)) + \boldsymbol{\mu}_z^\top \boldsymbol{W}^{-1}\boldsymbol{\mu}_z\right), \tag{3.3}$$

where $\boldsymbol{\mu}_z = \boldsymbol{RZ\beta}_z$, and $\boldsymbol{W} = \text{diag}(\boldsymbol{RVR})$. The assumed error variance σ^2 is set to be $\text{tr}(\boldsymbol{V})/n$, which is the expectation of the estimated variance.

3.3.3 Non-normal P

For non-normal error $\boldsymbol{\varepsilon}$, the true residual distribution P is unlikely to be a multivariate normal distribution. Thus, Equation 3.3 given in Li et al. (2024) will not be applicable to models violating the normality assumption.

To evaluate the Kullback-Leibler divergence of non-normal P from Q, the fallback is to solve Equation 3.2 numerically. However, since $\boldsymbol{e}$ is a linear transformation of non-normal random variables, it is very common that the general form of P is unknown, meaning that we can not easily compute $p(\boldsymbol{e})$ using a well-known probability density function. Additionally, even if $p(\boldsymbol{e})$ can be calculated for any $\boldsymbol{e} \in \mathbb{R}^n$, it will be very difficult to do numerical integration over the n-dimensional space, because n could be potentially very large.

In order to approximate D_{KL} in a practically computable manner, the elements of $\boldsymbol{e}$ are assumed to be independent of each other. This assumption solves both of the issues mentioned above. First, we no longer need to integrate over n random variables. The result of Equation 3.2 is now the sum of the Kullback-Leibler divergence evaluated for each individual residual due to the assumption of independence between observations. Second, it is not required to know the joint probability density $p(\boldsymbol{e})$ any more. Instead, the evaluation of Kullback-Leibler divergence for an individual residual relies on the knowledge of the marginal density $p_i(e_i)$, where e_i is the i-th residual for $i = 1, ..., n$. This is much easier to approximate through simulation. It is also worth mentioning that this independence assumption generally will not hold if $\text{cov}(e_i, e_j) \neq 0$ for any $1 \leq i < j \leq n$, but its existence is essential for reducing the computational cost.

Given $\boldsymbol{X}$ and $\boldsymbol{\beta}$, the algorithm for approximating Equation 3.2 starts from simulating m sets of observed values $\boldsymbol{y}$ according to the data generating process. The observed values are stored in a matrix $\boldsymbol{A}$ with n rows and m columns, where each column of $\boldsymbol{A}$ is a set of observed values. Then, we can get m sets of realized values of $\boldsymbol{e}$ stored in the matrix $\boldsymbol{B}$ by applying the residual operator $\boldsymbol{B} = \boldsymbol{RA}$. Furthermore, kernel density estimation (KDE) with Gaussian kernel and optimal bandwidth selected by the Silverman's rule of thumb (Silverman 2018) is applied on each row of $\boldsymbol{B}$ to estimate $p_i(e_i)$ for $i = 1, ..., n$. The KDE computation can be done by the `density` function in R.

Since the Kullback-Leibler divergence can be viewed as the expectation of the log-likelihood ratio between distribution P and distribution Q evaluated on distribution P, we can reuse the simulated residuals in matrix $\boldsymbol{B}$ to estimate the expectation by the sample mean. With the independence assumption, for non-normal P, D_{KL} can be approximated by

$$
\begin{aligned}
D_{KL} &\approx \sum_{i=1}^{n} \hat{D}_{KL}^{(i)}, \\
\hat{D}_{KL}^{(i)} &= \frac{1}{m} \sum_{j=1}^{m} \log \frac{\hat{p}_i(B_{ij})}{q(B_{ij})},
\end{aligned}
\tag{3.4}
$$

where $\hat{D}_{KL}^{(i)}$ is the estimator of the Kullback-Leibler divergence for an individual residual e_i, $\boldsymbol{B}_{ij}$ is the i-th row and j-th column entry of the matrix $\boldsymbol{B}$, $\hat{p}_i(.)$ is the kernel density estimator of $p_i(.)$, $q(.)$ is the normal density function with mean zero and an assumed variance estimated as $\hat{\sigma}^2 = \sum_{b \in vec(\boldsymbol{B})} (b - \sum_{b \in vec(\boldsymbol{B})} b/nm)^2/(nm-1)$, and $vec(.)$ is the vectorization operator which turns a $n \times m$ matrix into a $nm \times 1$ column vector by stacking the columns of the matrix on top of each other.

3.4 Distance Estimation

In the previous sections, we have defined a distance measure given in Equation 3.1 for quantifying the difference between the true residual distribution P and an ideal reference distribution Q. However, this distance measure can only be computed when the data generating process is known. In reality, we often have no knowledge about the data generating process, otherwise we do not need to do a residual diagnostic in the first place.

We use a computer vision model to estimate this distance measure for a residual plot. Let D be the result of Equation 3.1, and our estimator $\hat{D}$ is formulated as

$$
\hat{D} = f_{CV}(V_{h \times w}(\boldsymbol{e}, \hat{\boldsymbol{y}})),
\tag{3.5}
$$

where $V_{h \times w}(.)$ is a plotting function that saves a residuals vs fitted values plot with fixed aesthetic as an image with $h \times w$ pixels in three channels (RGB), $f_{CV}(.)$ is a computer vision model which takes an $h \times w$ image as input and predicts the distance in the domain $[0, +\infty)$.

With the estimated distance $\hat{D}$, we can compare the underlying distribution of the residuals to a theoretically "good" residual distribution. $\hat{D}$ can also be used as an index of the model violations indicating the strength of the visual signal embedded in the residual plot.

It is not expected that $\hat{D}$ will be equal to original distance D. This is largely because information contained in a single residual plot is limited and it may not be able to summarise all the important characteristics of the residual distribution. For a given residual distribution P, many different residual plots can be simulated, where many will share similar visual patterns, but some of them could be visually very different from the rest, especially for regression models with small n. This suggests the

error of the estimation will vary depends on whether the input residual plot is representative or not.

3.5 Statistical testing

3.5.1 Lineup Evaluation

Theoretically, the distance D for a correctly specified model is 0, because P will be the same as Q. However, the computer vision model may not necessary predict 0 for a null plot. Using Figure 3.1 as an example, it contains a visual pattern which is an indication of heteroskedasticity. We would not expect the model to be able to magically tell if the suspicious pattern is caused by the skewed distribution of the fitted values or the existence of heteroskedasticity. Additionally, some null plots could have outliers or strong visual patterns due to randomness, and a reasonable model will try to summarise those information into the prediction, resulting in $\hat{D} > 0$.

This property is not an issue if $\hat{D} \gg 0$ for which the visual signal of the residual plot is very strong, and we usually do not need any further examination of the significance of the result. However, if the visual pattern is weak or moderate, having $\hat{D}$ will not be sufficient to determine if H_0 should be rejected.

To address this issue we can adhere to the paradigm of visual inference, by comparing the estimated distance $\hat{D}$ to the estimated distances for the null plots in a lineup. Specifically, if a lineup comprises 20 plots, the null hypothesis H_0 will be rejected if $\hat{D}$ exceeds the maximum estimated distance among the $m-1$ null plots, denoted as $\max_{1 \leq i \leq m-1} \hat{D}^{(i)}_{null}$, where $\hat{D}^{(i)}_{null}$ represents the estimated distance for the i-th null plot. This approach is equivalent to the typical lineup protocol requiring a 95% significance level, where H_0 is rejected if the data plot is identified as the most distinct plot by the sole observer. The estimated distance serves as a metric to quantify the difference between the data plot and the null plots, as intended.

Moreover, if the number of plots in a lineup, denoted by m, is sufficiently large, the empirical distribution of $\hat{D}^{(i)}_{null}$ can be viewed as an approximation of the null distribution of the estimated distance. Consequently, quantiles of the null distribution can be estimated using the sample quantiles, and these quantiles can be utilized for decision-making purposes. The details of the sample quantile computation can be found in Hyndman and Fan (1996). For instance, if $\hat{D}$ is greater than or equal to the 95% sample quantile, denoted as $Q_{null}(0.95)$, we can conclude that the estimated distance for the true residual plot is significantly different from the estimated distance for null plots with a 95% significance level. Based on our experience, to obtain a stable estimate of the 95% quantile, the number of null plots, n_{null}, typically needs to be at least 100. However, if the null distribution exhibits a long tail, a larger number of null plots may be required. Alternatively, a p-value is the

probability of observing a distance equally or greater than $\hat{D}$ under the null hypothesis H_0, and it can be estimated by $\frac{1}{m} + \frac{1}{m}\sum_{i=1}^{m-1} I\left(\hat{D}_{null}^{(i)} \geq \hat{D}\right)$.

To alleviate computation burden, a lattice of quantiles for $\hat{D}$ under H_0 with specified sample sizes can be precomputed. We can then map the $\hat{D}$ and sample size to the closet quantile and sample size in lattice to calculate the corresponding p-value. This approach lose precision in p-value calculation, however, significantly improves computational efficiency.

3.5.2 Bootstrapping

Bootstrap is often employed in linear regression when conducting inference for estimated parameters Efron and Tibshirani (1994). It is typically done by sampling individual cases with replacement and refitting the regression model. If the observed data accurately reflects the true distribution of the population, the bootstrapped estimates can be used to measure the variability of the parameter estimate without making strong distributional assumptions about the data generating process.

Similarly, bootstrap can be applied on the estimated distance $\hat{D}$. For each refitted model $M_{boot}^{(i)}$, there will be an associated residual plot $V_{boot}^{(i)}$ which can be fed into the computer vision model to obtain $\hat{D}_{boot}^{(i)}$, where $i = 1, ..., n_{boot}$, and n_{boot} is the number of bootstrapped samples. If we are interested in the variation of $\hat{D}$, we can use $\hat{D}_{boot}^{(i)}$ to estimate a confidence interval.

Alternatively, since each $M_{boot}^{(i)}$ has a set of estimated coefficients $\hat{\boldsymbol{\beta}}_{boot}^{(i)}$ and an estimated variance $\hat{\sigma^2}_{boot}^{(i)}$, a new approximated null distribution can be construed and the corresponding 95% sample quantile $Q_{boot}^{(i)}(0.95)$ can be computed. Then, if $\hat{D}_{boot}^{(i)} \geq Q_{boot}^{(i)}(0.95)$, H_0 will be rejected for $M_{boot}^{(i)}$. The ratio of rejected $M_{boot}^{(i)}$ among all the refitted models provides an indication of how often the assumed regression model are considered to be incorrect if the data can be obtained repetitively from the same data generating process. But this approach is computationally very expensive because it requires $n_{boot} \times n_{null}$ times of residual plot assessment. In practice, $Q_{null}(0.95)$ can be used to replace $Q_{boot}^{(i)}(0.95)$ in the computation.

3.6 Model Violations Index

While statistical testing is a powerful tool for detecting model violations, it can become cumbersome and time-consuming when quick decisions are needed, particularly due to the need to evaluate numerous null plots. In practice, a more convenient and immediate method for assessing model performance is often required. This is where an index, such as the Model Violations Index (MVI), becomes valuable. It offers a straightforward way to quantify deviations from model assumptions, enabling rapid assessment and easier comparison across models.

The estimator $\hat{D}$ measures the difference between the true residual distribution and the reference

Table 3.1: *Degree of model violations or the strength of the visual signals according to the Model Violations Index (MVI). The constant C is set to be 10.*

Degree of model violations	Range ($C = 10$)
Strong	MVI > 8
Moderate	$6 <$ MVI < 8
Weak	MVI < 6

residual distribution, a difference primarily arises from deviations in model assumptions. The magnitude of D directly reflects the degree of these deviations, thus making $\hat{D}$ instrumental in forming a model violations index (MVI).

Note that if more observations are used for estimating the linear regression, the result of Equation 3.2 will increase, as the integration will be performed with larger n. For a given data generating process, D typically increases logarithmically with the number of observations. This behaviour comes from the relationship $D = \log(1 + D_{KL})$, where $D_{KL} = \sum_{i=1}^{n} D_{KL}^{(i)}$ under the assumption of independence.

Since $\hat{D}$ is an estimate of D, it is expected that a larger number of observations will also lead to a higher $\hat{D}$. However, this does not imply that $\hat{D}$ fails to accurately represent the extent of model violations. In fact, when examining residual plots with more observations, we often observe a stronger visual signal strength, as the underlying patterns are more likely to be revealed, except in cases of significant overlapping.

Therefore, the Model Violations Index (MVI) can be proposed as

$$\text{MVI} = C + \hat{D} - \log(n), \tag{3.6}$$

where C is a large enough constant keeping the result positive and the term $-\log(n)$ is used to offset the increase in D due to sample size.

Figure 3.2 displays the residual plots for fitted models exhibiting varying degrees of non-linearity and heteroskedasticity. Each residual plot's MVI is computed using Equation 3.6 with $C = 10$. When MVI > 8, the visual patterns are notably strong and easily discernible by humans. In the range $6 <$ MVI < 8, the visibility of the visual pattern diminishes as MVI decreases. Conversely, when MVI < 6, the visual pattern tends to become relatively faint and challenging to observe. Table 3.1 provides a summary of the MVI usage and it is applicable to other linear regression models.

3.7 Data Generation

3.7.1 Simulation Scheme

While observational data is frequently employed in training models for real-world applications, the data generating process of observational data often remains unknown, making computation for our target variable D unattainable. Consequently, the computer vision models developed in this study were trained using synthetic data, including 80,000 training images and 8,000 test images. This approach provided us with precise label annotations. Additionally, it ensured a large and diverse training dataset, as we had control over the data generating process, and the simulation of the training data was relatively cost-effective.

We have incorporated three types of residual departures of linear regression model in the training data, including non-linearity, heteroskedasticity and non-normality. All three departures can be summarised by the data generating process formulated as

$$\begin{aligned} \boldsymbol{y} &= \mathbf{1}_n + \boldsymbol{x}_1 + \beta_1 \boldsymbol{x}_2 + \beta_2(\boldsymbol{z} + \beta_1 \boldsymbol{w}) + \boldsymbol{k} \odot \boldsymbol{\varepsilon}, \\ \boldsymbol{z} &= \mathrm{He}_j(g(\boldsymbol{x}_1, 2)), \\ \boldsymbol{w} &= \mathrm{He}_j(g(\boldsymbol{x}_2, 2)), \\ \boldsymbol{k} &= \left[\mathbf{1}_n + b(2-|a|)(\boldsymbol{x}_1 + \beta_1 \boldsymbol{x}_2 - a\mathbf{1}_n)^{\circ 2}\right]^{\circ 1/2}, \end{aligned} \tag{3.7}$$

where $\boldsymbol{y}$, $\boldsymbol{x}_1$, $\boldsymbol{x}_2$, $\boldsymbol{z}$, $\boldsymbol{w}$, $\boldsymbol{k}$ and $\boldsymbol{\varepsilon}$ are vectors of size n, $\mathbf{1}_n$ is a vector of ones of size n, $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are two independent predictors, $\mathrm{He}_j(.)$ is the jth-order probabilist's Hermite polynomials (Hermite 1864), $(.)^{\circ 2}$ and $(.)^{\circ 1/2}$ are Hadamard square and square root, $\odot$ is the Hadamard product, and $g(\boldsymbol{x}, k)$ is a scaling function to enforce the support of the random vector to be $[-k, k]^n$ defined as

$$g(\boldsymbol{x}, k) = 2k \cdot \frac{\boldsymbol{x} - x_{\min}\mathbf{1}_n}{x_{\max} - x_{\min}} - k\mathbf{1}_n, \ for\ k > 0,$$

where $x_{\min} = \min_{i \in \{1,\ldots,n\}} x_i$, $x_{\max} = \max_{i \in \{1,\ldots,n\}} x_i$ and x_i is the i-th entry of $\boldsymbol{x}$.

The residuals and fitted values of the fitted model were obtained by regressing $\boldsymbol{y}$ on $\boldsymbol{x}_1$. If $\beta_1 \neq 0$, $\boldsymbol{x}_2$ was also included in the design matrix. This data generation process was adapted from Li et al. (2024), where it was utilized to simulate residual plots exhibiting non-linearity and heteroskedasticity visual patterns for human subject experiments. A summary of the factors utilized in Equation 3.7 is provided in Table 3.2.

In Equation 3.7, $\boldsymbol{z}$ and $\boldsymbol{w}$ represent higher-order terms of $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, respectively. If $\beta_2 \neq 0$, the regression model will encounter non-linearity issues. Parameter j serves as a shape parameter that

Table 3.2: *Factors used in the data generating process for synthetic data simulation. Factor j and a controls the non-linearity shape and the heteroskedasticity shape respectively. Factor b, σ_ε and n control the signal strength. Factor $dist_\varepsilon$, $dist_{x1}$ and $dist_{x2}$ specifies the distribution of ε, X_1 and X_2 respectively.*

Factor	Domain
j	{2, 3, ..., 18}
a	[-1, 1]
b	[0, 100]
β_1	{0, 1}
β_2	{0, 1}
$dist_\varepsilon$	{discrete, uniform, normal, lognormal}
$dist_{x1}$	{discrete, uniform, normal, lognormal}
$dist_{x2}$	{discrete, uniform, normal, lognormal}
σ_ε	[0.0625, 9]
σ_{X1}	[0.3, 0.6]
σ_{X2}	[0.3, 0.6]
n	[50, 500]

controls the number of tuning points in the non-linear pattern. Typically, higher values of j lead to an increase in the number of tuning points, as illustrated in Figure 3.3.

Additionally, scaling factor $\boldsymbol{k}$ directly affects the error distribution and it is correlated with $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$. If $b \neq 0$ and $\boldsymbol{\varepsilon} \sim N(\mathbf{0}_n, \sigma^2 \boldsymbol{I}_n)$, the constant variance assumption will be violated. Parameter a is a shape parameter controlling the location of the smallest variance in a residual plot as shown in Figure 3.4.

Non-normality violations arise from specifying a non-normal distribution for $\boldsymbol{\varepsilon}$. In the synthetic data simulation, four distinct error distributions are considered, including discrete, uniform, normal, and lognormal distributions, as presented in Figure 3.5. Each distribution imparts unique characteristics in the residual plot. The discrete error distribution introduces discrete clusters in residuals, while the lognormal distribution typically yields outliers. Uniform error distribution may result in residuals filling the entire space of the residual plot. All of these distributions exhibit visual distinctions from the normal error distribution.

Equation 3.7 accommodates the incorporation of the second predictor $\boldsymbol{x}_2$. Introducing it into the data generation process by setting $\beta_1 = 1$ significantly enhances the complexity of the shapes, as illustrated in Figure 3.6. In comparison to Figure 3.3, Figure 3.6 demonstrates that the non-linear shape resembles a surface rather than a single curve. This augmentation can facilitate the computer vision model in learning visual patterns from residual plots of the multiple linear regression model.

In real-world analysis, it is not uncommon to encounter instances where multiple model violations

coexist. In such cases, the residual plots often exhibit a mixed pattern of visual anomalies corresponding to different types of model violations. Figure 3.7 and Figure 3.8 show the visual patterns of models with multiple model violations.

3.7.2 Balanced Dataset

To train a robust computer vision model, we deliberately controlled the distribution of the target variable D in the training data. We ensured that it followed a uniform distribution between 0 and 7. This was achieved by organizing 50 buckets, each exclusively accepting training samples with D falling within the range $[7(i-1)/49, 7i/49)$ for $i < 50$, where i represents the index of the i-th bucket. For the 50-th bucket, any training samples with $D \geq 7$ were accepted.

With 80,000 training images prepared, each bucket accommodated a maximum of $80000/50 = 1600$ training samples. The simulator iteratively sampled parameter values from the parameter space, generated residuals and fitted values using the data generation process, computed the distance, and checked if the sample fitted within the corresponding bucket. This process continued until all buckets were filled.

Similarly, we adopted the same methodology to prepare 8,000 test images for performance evaluation and model diagnostics.

3.8 Model Architecture

The architecture of the computer vision model is adapted from a well-established architecture known as VGG16, which has demonstrated high performance in image classification (Simonyan and Zisserman 2014). Figure 3.9 provides a diagram of the architecture. More details about the neural network layers used in this study are provided in Appendix B.

The model begins with an input layer of shape $n \times h \times w \times 3$, capable of handling n RGB images. This is followed by a grayscale conversion layer utilizing the luma formula under the Rec. 601 standard, which converts the colour image to grayscale. Grayscale suffices for our task since data points are plotted in black. We experiment with three combinations of h and w: 32×32, 64×64, and 128×128, aiming to achieve sufficiently high image resolution for the problem at hand.

The processed image is used as the input for the first convolutional block. The model comprises at most five consecutive convolutional blocks, mirroring the original VGG16 architecture. Within each block, there are two 2D convolutional layers followed by two activation layers, respectively. Subsequently, a 2D max-pooling layer follows the second activation layer. The 2D convolutional layer convolves the input with a fixed number of 3×3 convolution filters, while the 2D max-pooling layer downsamples the input along its spatial dimensions by taking the maximum value over a 2×2 window

for each channel of the input. The activation layer employs the rectified linear unit (ReLU) activation function, a standard practice in deep learning, which introduces a non-linear transformation of the output of the 2D convolutional layer. Additionally, to regularize training, a batch normalization layer is added after each 2D convolutional layer and before the activation layer. Finally, a dropout layer is appended at the end of each convolutional block to randomly set some inputs to zero during training, further aiding in regularization.

The output of the last convolutional block is summarized by either a global max pooling layer or a global average pooling layer, resulting in a two-dimensional tensor. To leverage the information contained in scagnostics, this tensor is concatenated with an additional $n \times 5$ tensor, which contains the "Monotonic", "Sparse", "Splines", and "Striped" measures, along with the number of observations for n residual plots.

The concatenated tensor is then fed into the final prediction block. This block consists of two fully-connected layers. The first layer contains at least 128 units, followed by a dropout layer. Occasionally, a batch normalization layer is inserted between the fully-connected layer and the dropout layer for regularization purposes. The second fully-connected layer consists of only one unit, serving as the output of the model.

The model weights $\boldsymbol{\theta}$ were randomly initialized and they were optimized by the Adam optimizer (Kingma and Ba 2014) with the mean square error loss function

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \frac{1}{n_{\text{train}}} \sum_{i=1}^{n_{\text{train}}} (D_i - f_{\boldsymbol{\theta}}(V_i, S_i))^2,$$

where n_{train} is the number of training samples, V_i is the i-th residual plot and S_i is the additional information about the i-th residual plot including four scagnostics and the number of observations.

3.9 Model Training

To achieve a near-optimal deep learning model, hyperparameters like the learning rate often need to be fine-tuned using a tuner. In our study, we utilized the Bayesian optimization tuner from the `KerasTuner` Python library (O'Malley et al. 2019) for this purpose. A comprehensive list of hyperparameters is provided in Table 3.3.

The number of base filters determines the number of filters for the first 2D convolutional layer. In the VGG16 architecture, the number of filters for the 2D convolutional layer in a block is typically twice the number in the previous block, except for the last block, which maintains the same number of convolution filters as the previous one. This hyperparameter aids in controlling the complexity

of the computer vision model. A higher number of base filters results in more trainable parameters. Likewise, the number of units for the fully-connected layer determines the complexity of the final prediction block. Increasing the number of units enhances model complexity, resulting in more trainable parameters.

The dropout rate and batch normalization are flexible hyperparameters that work in conjunction with other parameters to facilitate smooth training. A higher dropout rate is necessary when the model tends to overfit the training dataset by learning too much noise (Srivastava et al. 2014). Conversely, a lower dropout rate is preferred when the model is complex and challenging to converge. Batch normalization, on the other hand, addresses the internal covariate shift problem arising from the randomness in weight initialization and input data (Goodfellow et al. 2016). It helps stabilize and accelerate the training process by normalizing the activations of each layer.

Additionally, incorporating additional inputs such as scagnostics and the number of observations can potentially enhance prediction accuracy. Therefore, we allow the tuner to determine whether these inputs were necessary for optimal model performance.

The learning rate is a crucial hyperparameter, as it dictates the step size of the optimization algorithm. A high learning rate can help the model avoid local minima but risks overshooting and missing the global minimum. Conversely, a low learning rate smoothens the training process but makes the convergence time longer and increases the likelihood of getting trapped in local minima.

Our model was trained on the MASSIVE M3 high-performance computing platform (Goscinski et al. 2014), using TensorFlow (Abadi et al. 2016) and Keras (Chollet et al. 2015). During training, 80% of the training data was utilized for actual training, while the remaining 20% was used as validation data. The Bayesian optimization tuner conducted 100 trials to identify the best hyperparameter values based on validation root mean square error. The tuner then restored the best epoch of the best model from the trials. Additionally, we applied early stopping, terminating the training process if the validation root mean square error fails to improve for 50 epochs. The maximum allowed epochs was set at 2,000, although no models reached this threshold.

Based on the tuning process described above, the optimized hyperparameter values are presented in Table 3.4. It was observable that a minimum of 32 base filters was necessary, with the preferable choice being 64 base filters for both the 64×64 and 128×128 models, mirroring the original VGG16 architecture. The optimized dropout rate for convolutional blocks hovered around 0.4, and incorporating batch normalization for convolutional blocks proved beneficial for performance.

All optimized models chose to retain the additional inputs, contributing to the reduction of validation

Table 3.3: *Name of hyperparameters and their correspoding domain for the computer vision model.*

Hyperparameter	Domain
Number of base filters	4, 8, 16, 32, 64
Dropout rate for convolutional blocks	[0.1, 0.6]
Batch normalization for convolutional blocks	false, true
Type of global pooling	max, average
Ignore additional inputs	false, true
Number of units for the fully-connected layer	128, 256, 512, 1024, 2048
Batch normalization for the fully-connected layer	false, true
Dropout rate for the fully-connected layer	[0.1, 0.6]
Learning rate	$[10^{-8}, 10^{-1}]$

Table 3.4: *Hyperparameters values for the optimized computer vision models with different input sizes.*

Hyperparameter	32 × 32	64 × 64	128 × 128
Number of base filters	32	64	64
Dropout rate for convolutional blocks	0.4	0.3	0.4
Batch normalization for convolutional blocks	true	true	true
Type of global pooling	max	average	average
Ignore additional inputs	false	false	false
Number of units for the fully-connected layer	256	256	256
Batch normalization for the fully-connected layer	false	true	true
Dropout rate for the fully-connected layer	0.2	0.4	0.1
Learning rate	0.0003	0.0006	0.0052

error. The number of units required for the fully-connected layer was 256, a relatively modest number compared to the VGG16 classifier, suggesting that the problem at hand was less complex. The optimized learning rates were higher for models with higher resolution input, likely because models with more parameters are more prone to getting stuck in local minima, requiring a higher learning rate.

3.10 Results

3.10.1 Model Performance

The test performance for the optimized models with three different input sizes are summarized in Table 3.5. Among these models, the 32 × 32 model consistently exhibited the best test performance. The mean absolute error of the 32 × 32 model indicated that the difference between $\hat{D}$ and D was approximately 0.43 on the test set, a negligible deviation considering the normal range of D typically falls between 0 and 7. The high R^2 values also suggested that the predictions were largely linearly correlated with the target.

Table 3.5: *The test performance of three optimized models with different input sizes.*

	RMSE	R^2	MAE	Huber loss
32×32	0.660	0.901	0.434	0.18
64×64	0.674	0.897	0.438	0.19
128×128	0.692	0.892	0.460	0.20

Figure 3.10 presents a hexagonal heatmap for $D - \hat{D}$ versus D. The brown smoothing curves, fitted by generalized additive models (Hastie 2017), demonstrate that all the optimized models perform admirably on the test sets when $1.5 < D < 6$, where no structural issues are noticeable. However, over-predictions occurred when $D < 1.5$, while under-predictions occurred predominantly when $\hat{D} > 6$.

For input images representing null plots where $D = 0$, it was expected that the models will over-predict the distance, as explained in Section 3.5.1. However, it can not explain the under-prediction issue. Therefore, we analysed the relationship between residuals and all the factors involved in the data generating process. We found that most issues actually arose from non-linearity problems and the presence of a second predictor in the regression model as illustrated in Figure 3.11. When the variance for the error distribution was small, the optimized model tended to under-predict the distance. Conversely, when the error distribution had a large variance, the model tended to over-predict the distance.

Since most of the deviation stemmed from the presence of non-linearity violations, to further investigate this, we split the test set based on violation types and re-evaluated the performance, as detailed in Table 3.6. It was evident that metrics for null plots were notably worse compared to other categories. Furthermore, residual plots solely exhibiting non-normality issues were the easiest to predict, with very low test root mean square error (RMSE) at around 0.3. Residual plots with non-linearity issues were more challenging to assess than those with heteroskedasticity or non-normality issues. When multiple violations were introduced to a residual plot, the performance metrics typically lay between the metrics for each individual violation.

Based on the model performance metrics, we chose to use the best-performing model evaluated on the test set, namely the 32×32 model, for the subsequent analysis.

3.10.2 Comparison with Human Visual Inference and Conventional Tests

3.10.2.1 Overview of the Human Subject experiment

In order to check the validity of the proposed computer vision model, residual plots presented in the human subject experiment conducted by Li et al. (2024) will be assessed.

Table 3.6: *The training and test performance of the* 32×32 *model presented with different model violations.*

Violations	#samples	RMSE
no violations	155	1.267
non-linearity	2218	0.787
heteroskedasticity	1067	0.602
non-linearity + heteroskedasticity	985	0.751
non-normality	1111	0.320
non-linearity + non-normality	928	0.600
heteroskedasticity + non-normality	819	0.489
non-linearity + heteroskedasticity + non-normality	717	0.620

Table 3.7: *The performance of the* 32×32 *model on the data used in the human subject experiment.*

Violation	RMSE	R^2	MAE	Huber loss
heteroskedasticity	0.721	0.852	0.553	0.235
non-linearity	0.738	0.770	0.566	0.246

This study has collected 7,974 human responses to 1,152 lineups. Each lineup contains one randomly placed true residual plot and 19 null plots. Among the 1,152 lineups, 24 are attention check lineups in which the visual patterns are designed to be extremely obvious and very different from the corresponding to null plots, 36 are null lineups where all the lineups consist of only null plots, 279 are lineups with uniform predictor distribution evaluated by 11 participants, and the remaining 813 are lineups with discrete, skewed or normal predictor distribution evaluated by 5 participants. Attention check lineups and null lineups will not be assessed in the following analysis.

In Li et al. (2024), the residual plots are simulated from a data generating process which is a special case of Equation 3.7. The main characteristic is the model violations are introduced separately, meaning non-linearity and heteroskedasticity will not co-exist in one lineup but assigned uniformly to all lineups. Additionally, non-normality and multiple predictors are not considered in the experimental design.

3.10.2.2 Model Performance on the Human-evaluated Data

For each lineup used in Li et al. (2024), there is one true residual plot and 19 null plots. While the distance D for the true residual plot depends on the underlying data generating process, the distance D for the null plots is zero. We have used our optimized computer vision model to estimate distance for both the true residual plots and the null plots. To have a fair comparison, H_0 will be rejected if the true residual plot has the greatest estimated distance among all plots in a lineup. Additionally,

the appropriate conventional tests including the Ramsey Regression Equation Specification Error Test (RESET) (Ramsey 1969) for non-linearity and the Breusch-Pagan test (Breusch and Pagan 1979) for heteroskedasticity were applied on the same data for comparison.

The performance metrics of $\hat{D}$ for true residual plots are outlined in Table 3.7. It's notable that all performance metrics are slightly worse than those evaluated on the test data. Nevertheless, the mean absolute error remains at a low level, and the linear correlation between the prediction and the true value remains very high. Consistent with results in Table 3.6, lineups with non-linearity issues are more challenging to predict than those with heteroskedasticity issues.

Table 3.8 provides a summary of the agreement between decisions made by the computer vision model and conventional tests. The agreement rates between conventional tests and the computer vision model are 85.95% and 79.69% for residual plots containing heteroskedasticity and non-linearity patterns, respectively. These figures are higher than those calculated for visual tests conducted by human, indicating that the computer vision model exhibits behaviour more akin to the best available conventional tests. However, Figure 3.12 shows that the computer vision model does not always reject when the conventional tests reject. And a small number of plots will be rejected by computer vision model but not by conventional tests. This suggests that conventional tests are more sensitive than the computer vision model.

Figure 3.14 further illustrates the decisions made by visual tests conducted by human, computer vision models, and conventional tests, using a parallel coordinate plots. It can be observed that all three tests will agree with each other for around 50% of the cases. When visual tests conducted by human do not reject, there are substantial amount of cases where computer vision model also do not reject but conventional tests reject. There are much fewer cases that do not reject by visual tests and conventional tests, but is rejected by computer vision models. This indicates computer vision model can behave like visual tests conducted by human better than conventional tests. Moreover, there are great proportion of cases where visual tests conducted by human is the only test who does not reject.

When plotting the decision against the distance, as illustrated in Figure 3.13, several notable observations emerge. Firstly, compared to conventional tests, the computer vision model tends to have fewer rejected cases when $D < 2$ and fewer non-rejected cases when $2 < D < 4$. This suggests tests based on the computer vision model are less sensitive to small deviations from model assumptions than conventional tests but more sensitive to moderate deviations. Additionally, visual tests demonstrate the lowest sensitivity to residual plots with small distances where not many residual plots are rejected when $D < 2$. Similarly, for large distances where $D > 4$, almost all residual plots are rejected by the computer vision model and conventional tests, but for visual tests conducted by humans, the

threshold is higher with $D > 5$.

In Figure 3.13, rejection decisions are fitted by logistic regression models with no intercept terms and an offset equals to $\log(0.05/0.95)$. The fitted curves for the computer vision model fall between those of conventional tests and visual tests for both non-linearity and heteroskedasticity, which means there is still potential to refine the computer vision model to better align its behaviour with visual tests conducted by humans.

In the experiment conducted in Li et al. (2024), participants were allowed to make multiple selections for a lineup. The weighted detection rate was computed by assigning weights to each detection. If the participant selected zero plots, a weight of 0.05 was assigned; otherwise, if the true residual plot was detected, the weight was 1 divided by the number of selections. This weighted detection rate allow us to assess the quality of the distance measure purposed in this chapter, by using the δ-difference statistic. The δ-difference is originally defined by Chowdhury et al. (2018), is given by

$$\delta = \bar{d}_{\text{true}} - \max_j \left(\bar{d}_{\text{null}}^{(j)} \right) \quad \text{for } j = 1, ..., m-1,$$

where $\bar{d}_{\text{null}}^{(j)}$ is the mean distance between the j-th null plot and the other null plots, $\bar{d}_{\text{true}}$ is the mean distance between the true residual plot and null plots, and m is the number of plots in a lineup. These mean distances are used because, as noted by Chowdhury et al. (2018), the distances can vary depending on which data plot is used for comparison. For instance, with three null plots, A, B and C, the distance between A and B may differ from the distance between A and C. To obtain a consistent distance for null plot A, averaging is necessary. However, this approach is not applicable to the distance proposed in this chapter, as we only compare the residual plot against a theoretically good residual plot. Consequently, the statistic must be adjusted to evaluate our distance measure effectively.

One important aspect that the δ-difference was designed to capture is the empirical distribution of distances for null plot. If we were to replace the mean distances $\bar{d}_{\text{null}}^{(j)}$ directly with $D_{\text{null}}^{(j)}$, the distance of the j-th null plot, the resulting distribution would be degenerate, since D_{null} equals zero by definition. Additionally, D can not be derived from an image, meaning it falls outside the scope of the distances considered by Chowdhury et al. (2018). Instead, the focus should be on the empirical distribution of $\hat{D}$, as it influences decision-making. Therefore, the adjusted δ-different is defined as

$$\delta_{\text{adj}} = \hat{D} - \max_j \left(\hat{D}_{\text{null}}^{(j)} \right) \quad \text{for } j = 1, ..., m-1,$$

Table 3.8: *Summary of the comparison of decisions made by computer vision model with decisions made by conventional tests and visual tests conducted by human.*

Violations	#Samples	#Agreements	Agreement rate
Compared with conventional tests			
heteroskedasticity	540	464	0.8593
non-linearity	576	459	0.7969
Compared with visual tests conducted by human			
heteroskedasticity	540	367	0.6796
non-linearity	576	385	0.6684

where $\hat{D}^{(j)}_{\text{null}}$ is the estimated distance for the j-th null plot, and m is the number of plots in a lineup.

Figure 3.15 displays the scatter plot of the weighted detection rate vs the adjusted δ-difference. It indicates that the weighted detection rate increases as the adjusted δ-difference increases, particularly when the adjusted δ-difference is greater than zero. A negative adjusted δ-difference suggests that there is at least one null plot in the lineup with a stronger visual signal than the true residual plot. In some instances, the weighted detection rate is close to one, yet the adjusted δ-difference is negative. This discrepancy implies that the distance measure, or the estimated distance, may not perfectly reflect actual human behaviour.

3.11 Examples

In this section, we present the performance of trained computer vision model on three example datasets. These include the dataset associated with the residual plot displaying a "left-triangle" shape, as displayed in Figure 3.1, along with the Boston housing dataset (Harrison Jr and Rubinfeld 1978), and the "dino" datasets from the `datasauRus` R package (Davies et al. 2022).

The first example illustrates a scenario where both the computer vision model and human visual inspection successfully avoid rejecting H_0 when H_0 is true, contrary to conventional tests. This underscores the necessity of visually examining the residual plot.

In the second example, we encounter a more pronounced violation of the model, resulting in rejection of H_0 by all three tests. This highlights the practicality of the computer vision model, particularly for less intricate tasks.

The third example presents a situation where the model deviation is non-typical. Here, the computer vision model and human visual inspection reject H_0, whereas some commonly used conventional tests do not. This emphasizes the benefits of visual inspection and the unique advantage of the computer vision model, which, like humans, makes decisions based on visual discoveries.

3.11.1 Left-triangle

In Section 3.1, we presented an example residual plot showcased in Figure 3.1, illustrating how humans might misinterpret the "left-triangle" shape as indicative of heteroskedasticity. Additionally, the Breusch-Pagan test yielded a rejection with a p-value of 0.046, despite the residuals originating from a correctly specified model. Figure 3.17 offers a lineup for this fitted model, showcasing various degrees of "left-triangle" shape across all residual plots. This phenomenon is evidently caused by the skewed distribution of the fitted values. Notably, if the residual plot were evaluated through a visual test, it would not be rejected since the true residual plot positioned at 10 can not be distinguished from the others.

Figure 3.16 presents the results of the assessment by the computer vision model. Notably, the observed visual signal strength is considerably lower than the 95% sample quantile of the null distribution. Moreover, the bootstrapped distribution suggests that it is highly improbable for the fitted model to be misspecified as the majority of bootstrapped fitted models will not be rejected. Thus, for this particular fitted model, both the visual test and the computer vision model will not reject H_0. However, the Breusch-Pagan test will reject H_0 because it can not effectively utilize information from null plots.

The attention map at Figure 3.16B suggests that the estimation is highly influenced by the top-right and bottom-right part of the residual plot, as it forms two vertices of the triangular shape. A principal component analysis (PCA) is also performed on the output of the global pooling layer of the computer vision model. As mentioned in Simonyan and Zisserman (2014), a computer vision model built upon the convolutional blocks can be viewed as a feature extractor. For the 32×32 model, there are 256 features outputted from the global pooling layer, which can be further used for different visual tasks not limited to distance prediction. To see if these features can be effectively used for distinguishing null plots and true residual plot, we linearly project them into the first and second principal components space as shown in Figure 3.16D. It can be observed that because the bootstrapped plots are mostly similar to the null plots, the points drawn in different colours are mixed together. The true residual plot is also covered by both the cluster of null plots and cluster of bootstrapped plots. This accurately reflects our understanding of Figure 3.17.

3.11.2 Boston Housing

The Boston housing dataset, originally published by Harrison Jr and Rubinfeld (1978), offers insights into housing in the Boston, Massachusetts area. For illustration purposes, we utilize a reduced version from Kaggle, comprising 489 rows and 4 columns: average number of rooms per dwelling (RM), percentage of lower status of the population (LSTAT), pupil-teacher ratio by town (PTRATIO), and median value of owner-occupied homes in $1000's (MEDV). In our analysis, MEDV will serve as the

response variable, while the other columns will function as predictors in a linear regression model. Our primary focus is to detect non-linearity, because the relationships between RM and MEDV or LSTAT and MEDV are non-linear.

Figure 3.18 displays the residual plot and the assessment conducted by the computer vision model. A clear non-linearity pattern resembling a "U" shape is shown in the plot A. Furthermore, the RESET test yields a very small p-value. The estimated distance $\hat{D}$ significantly exceeds $Q_{null}(0.95)$, leading to rejection of H_0. The bootstrapped distribution also suggests that almost all the bootstrapped fitted models will be rejected, indicating that the fitted model is unlikely to be correctly specified. The attention map in plot B suggests the center of the image has higher leverage than other areas, and it is the turning point of the "U" shape. The CPA provided in plot D shows two distinct clusters of data points, further underling the visual differences between bootstrapped plots and null plots. This coincides the findings from Figure 3.19, where the true plot exhibiting a "U" shape is visually distinctive from null plots. If a visual test is conducted by human, H_0 will also be rejected.

3.11.3 DatasauRus

The computer vision model possesses the capability to detect not only typical issues like non-linearity, heteroskedasticity, and non-normality but also artifact visual patterns resembling real-world objects, as long as they do not appear in null plots. These visual patterns can be challenging to categorize in terms of model violations. Therefore, we will employ the RESET test, the Breusch-Pagan test, and the Shapiro-Wilk test (Shapiro and Wilk 1965) for comparison.

The "dino" dataset within the `datasauRus` R package exemplifies this scenario. With only two columns, x and y, fitting a regression model to this data yields a residual plot resembling a "dinosaur", as displayed in Figure 3.20A. Unsurprisingly, this distinct residual plot stands out in a lineup, as shown in Figure 3.21. A visual test conducted by humans would undoubtedly reject H_0.

According to the residual plot assessment by the computer vision model, $\hat{D}$ exceeds $Q_{null}(0.95)$, warranting a rejection of H_0. Additionally, most of the bootstrapped fitted models will be rejected, indicating an misspecified model. However, both the RESET test and the Breusch-Pagan test yield p-values greater than 0.3, leading to a non-rejection of H_0. Only the Shapiro-Wilk test rejects the normality assumption with a small p-value.

More importantly, the attention map in Figure 3.20B clearly exhibits a "dinosaur" shape, strongly suggesting the prediction of the distance is based on human perceptible visual patterns. The computer vision model is also capable of capturing the contour or the outline of the embedded shape, just like human being reading residual plots. The PCA in Figure 3.20D also shows that the cluster of bootstrapped plots is at the corner of the cluster of null plots.

More importantly, the attention map in Figure 3.20B clearly exhibits a "dinosaur" shape, strongly suggesting that the distance prediction is based on human-perceptible visual patterns. The computer vision model effectively captures the contour or outline of the embedded shape, similar to how humans interpret residual plots. Additionally, the PCA in Figure 3.20D demonstrates that the cluster of bootstrapped plots is positioned at the corner of the cluster of null plots.

In practice, without accessing the residual plot, it would be challenging to identify the artificial pattern of the residuals. Moreover, conducting a normality test for a fitted regression model is not always standard practice among analysts. Even when performed, violating the normality assumption is sometimes deemed acceptable, especially considering the application of quasi-maximum likelihood estimation in linear regression. This example underscores the importance of evaluating residual plots and highlights how the proposed computer vision model can facilitate this process.

3.12 Limitations and Future Work

Despite the computer vision model performing well with general cases under the synthetic data generation scheme and the three examples used in this chapter, this study has several limitations that could guide future work.

The proposed distance measure assumes that the true model is a classical normal linear regression model, which can be restrictive. Although this chapter does not address the relaxation of this assumption, there are potential methods to evaluate other types of regression models. The most comprehensive approach would be to define a distance measure for each different class of regression model and then train the computer vision model following the methodology described in this chapter. To accelerate training, one could use the convolutional blocks of our trained model as a feature extractor and perform transfer learning on top of it, as these blocks effectively capture shapes in residual plots. Another approach would be to transform the residuals so they are roughly normally distributed and have constant variance. If only raw residuals are used, the distance-based statistical testing compares the difference in distance to a classical normal linear regression model for the true plot and null plots. This comparison is meaningful only if the difference can be identified by the distance measure proposed in this chapter.

There are other types of residual plots commonly used in diagnostics, such as residuals vs. predictor and quantile-quantile plots. In this study, we focused on the most commonly used residual plot as a starting point for exploring the new field of automated visual inference. Similarly, we did not explore other, more sophisticated computer vision model architectures and specifications for the same reason. While the performance of the computer vision model is acceptable, there is still room

for improvement to achieve behavior more closely resembling that of humans interpreting residual plots. This may require external survey data or human subject experiment data to understand the fundamental differences between our implementation and human evaluation.

3.13 Conclusion

In this chapter, we have introduced a distance measure based on Kullback-Leibler divergence to quantify the disparity between the residual distribution of a fitted classical normal linear regression model and the reference residual distribution assumed under correct model specification. This distance measure effectively captures the magnitude of model violations in misspecified models. We propose a computer vision model to estimate this distance, utilizing the residual plot of the fitted model as input. The resulting estimated distance serves as the foundation for constructing a single Model Violations Index (MVI), facilitating the quantification of various model violations.

Moreover, the estimated distance enables the development of a formal statistical testing procedure by evaluating a large number of null plots generated from the fitted model. Additionally, employing bootstrapping techniques and refitting the regression model allows us to ascertain how frequently the fitted model is considered misspecified if data were repeatedly obtained from the same data generating process.

The trained computer vision model demonstrates strong performance on both the training and test sets, although it exhibits slightly lower performance on residual plots with non-linearity visual patterns compared to other types of violations. The statistical tests relying on the estimated distance predicted by the computer vision model exhibit lower sensitivity compared to conventional tests but higher sensitivity compared to visual tests conducted by humans. While the estimated distance generally mirrors the strength of the visual signal perceived by humans, there remains scope for further improvement in its performance.

Several examples are provided to showcase the effectiveness of the proposed method across different scenarios, emphasizing the similarity between visual tests and distance-based tests. Overall, both visual tests and distance-based tests can be viewed as ensemble of tests, aiming to assess any violations of model assumptions collectively. In contrast, individual residual diagnostic tests such as the RESET test and the Breusch-Pagan test only evaluate specific violations of model assumptions. In practice, selecting an appropriate set of statistical tests for regression diagnostics can be challenging, particularly given the necessity of adjusting the significance level for each test.

Our method holds significant value as it helps alleviate a portion of analysts' workload associated with assessing residual plots. While we recommend analysts to continue reading residual plots whenever

feasible, as they offer invaluable insights, our approach serves as a valuable tool for automating the diagnostic process or for supplementary purposes when needed.

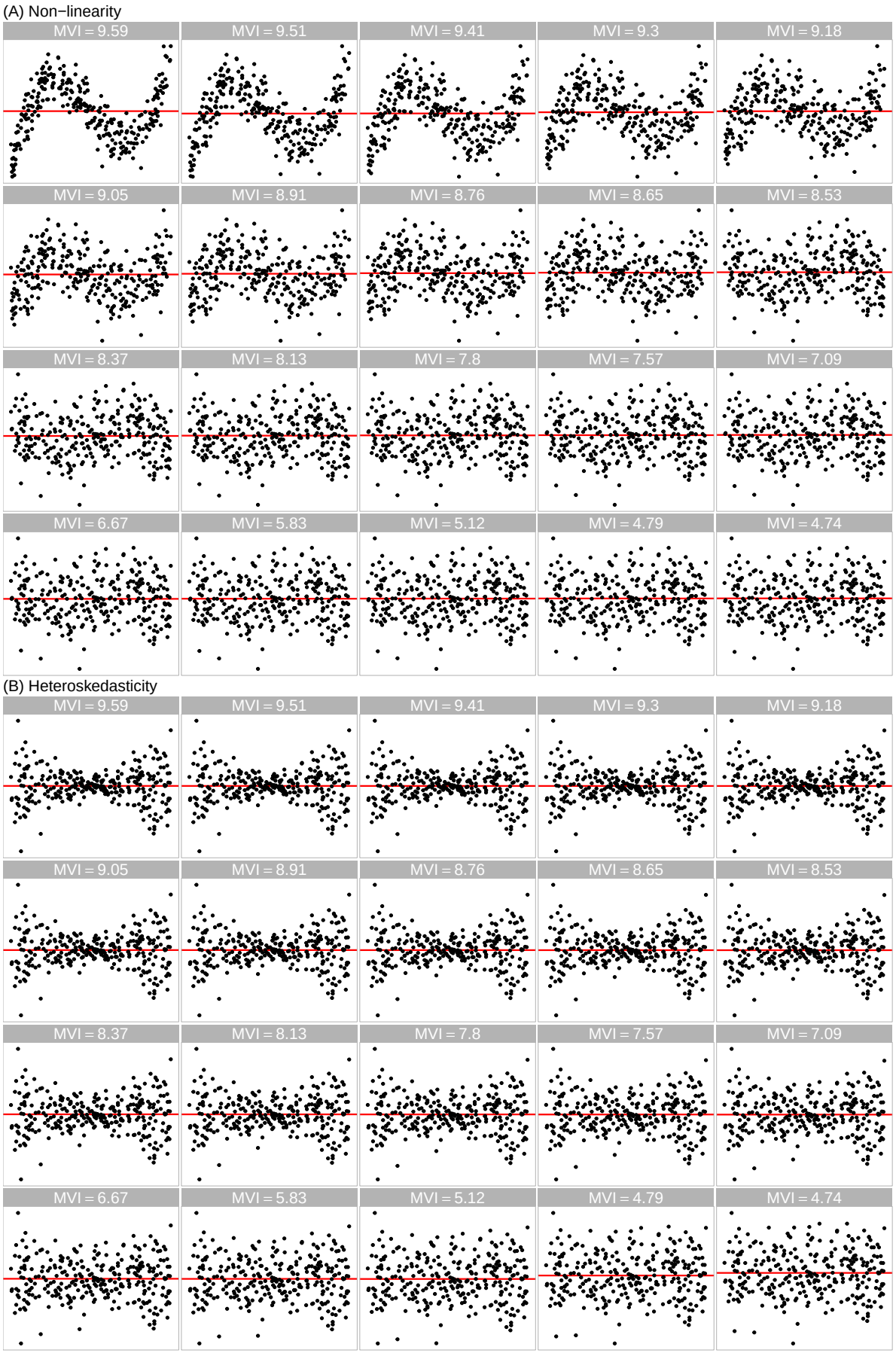


Figure 3.2: *Residual plots generated from fitted models exhibiting varying degrees of (A) non-linearity and (B) heteroskedasticity violations. The model violations index (MVI) is displayed atop each residual plot. The non-linearity patterns are relatively strong for $MVI > 8$, and relatively weak for $MVI < 6$, while the heteroskedasticity patterns are relatively strong for $MVI > 8$, and relatively weak for $MVI < 6$.*

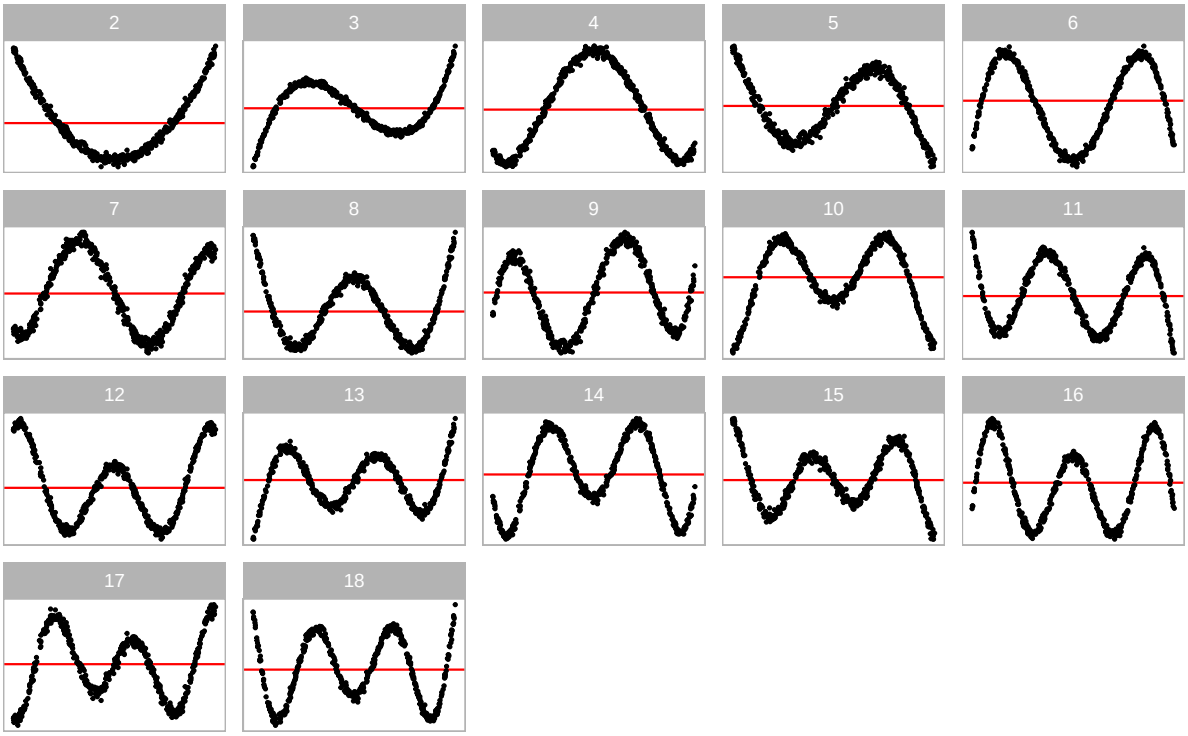


Figure 3.3: *Non-linearity forms generated for the synthetic data simulation. The 17 shapes are generated by varying the order of polynomial given by j in $He_j(.)$.*

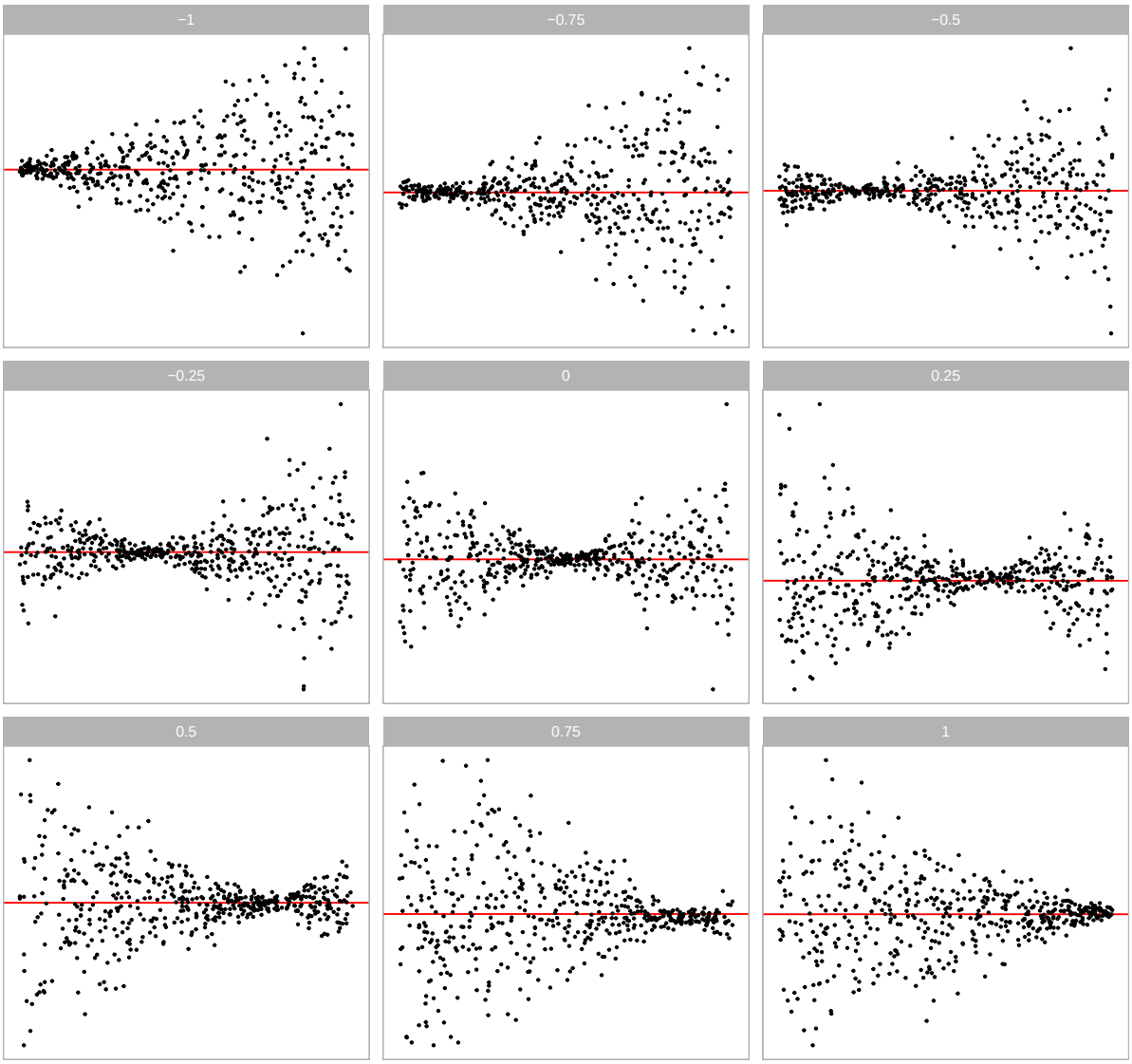


Figure 3.4: *Heteroskedasticity forms generated for the synthetic data simulation. Different shapes are controlled by the continuous factor a between -1 and 1. For $a = -1$, the residual plot exhibits a "left-triangle" shape. And for $a = 1$, the residual plot exhibits a "right-triangle" shape.*

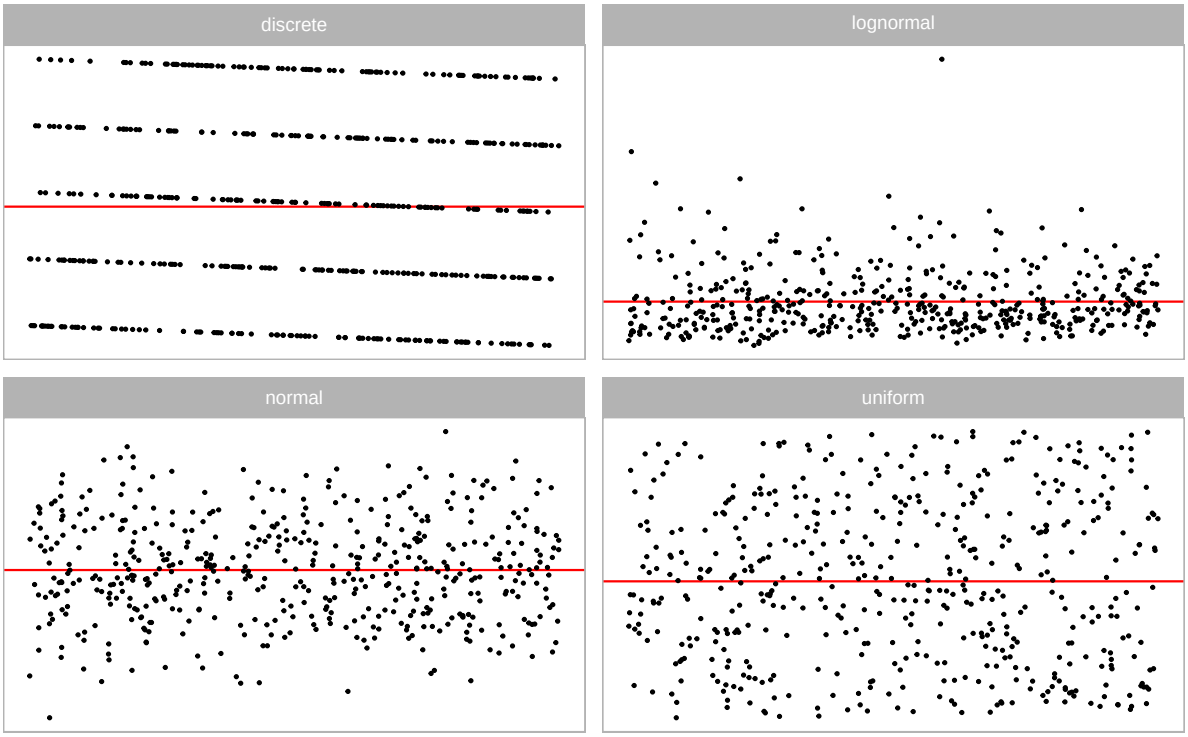


Figure 3.5: *Non-normality forms generated for the synthetic data simulation. Four different error distributions including discrete, lognormal, normal and uniform are considered.*

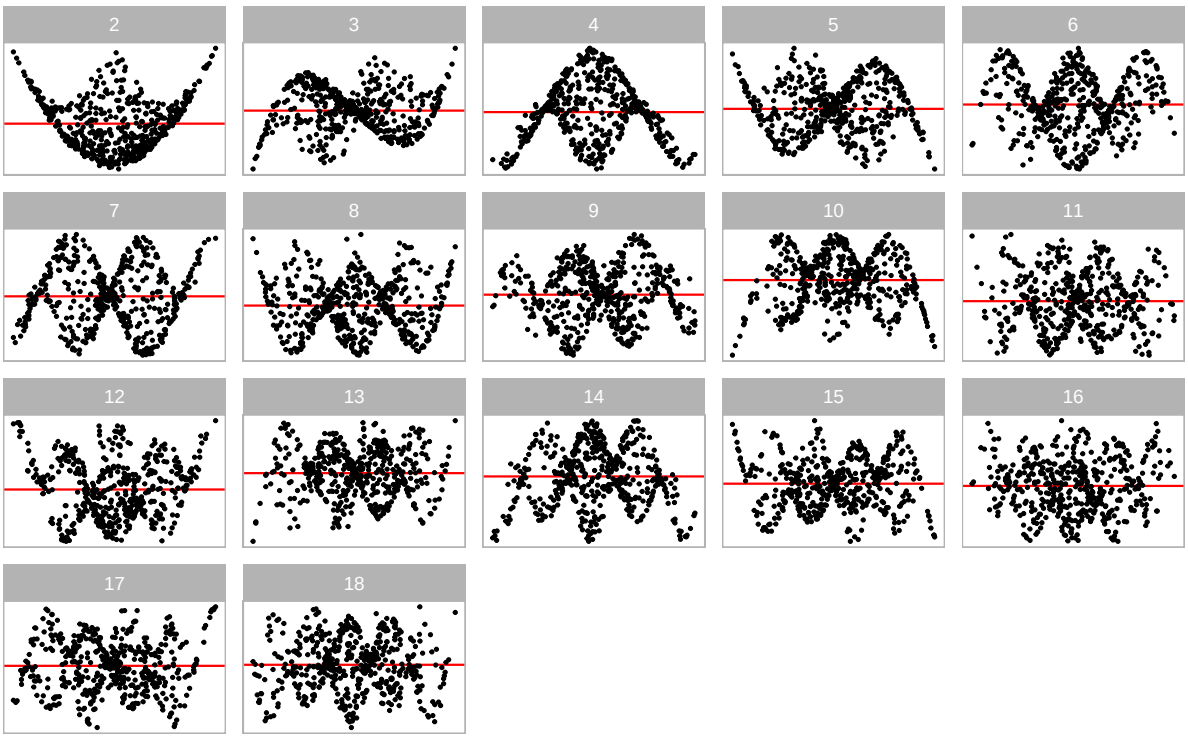


Figure 3.6: *Residual plots of multiple linear regression models with non-linearity issues. The 17 shapes are generated by varying the order of polynomial given by j in $He_j(.)$. A second predictor x_2 is introduced to the regression model to create complex shapes.*

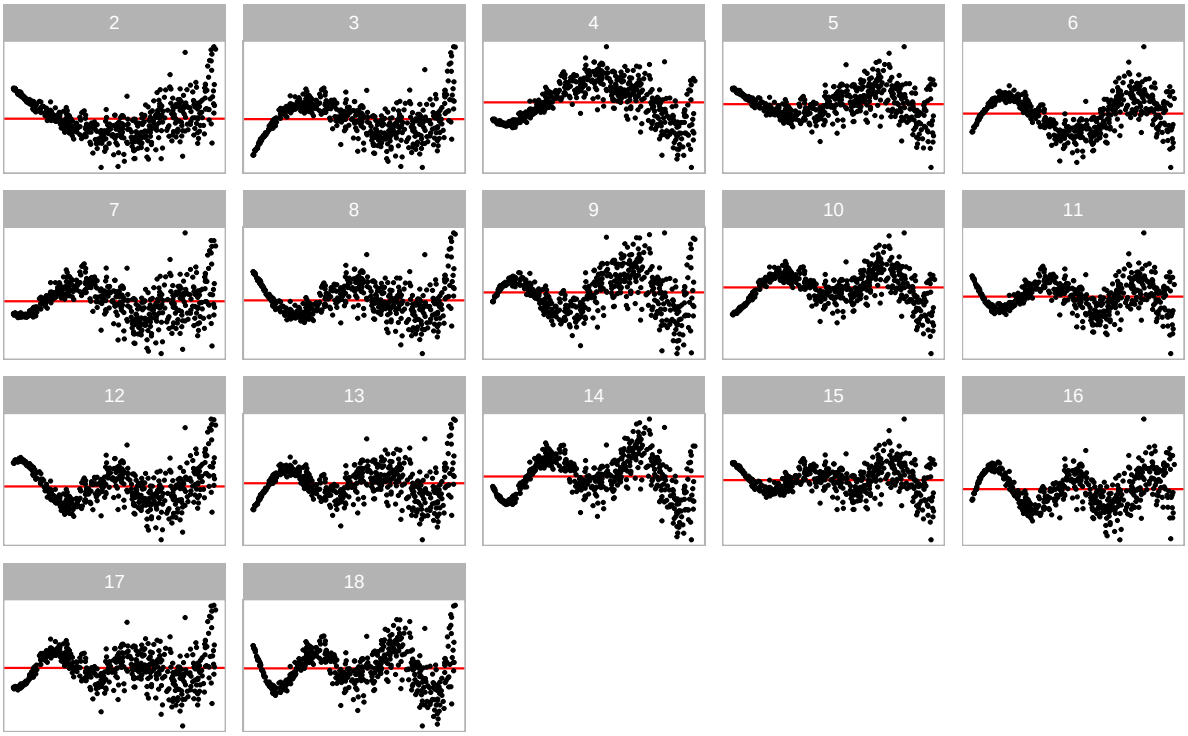


Figure 3.7: *Residual plots of models violating both the non-linearity and the heteroskedasticity assumptions. The 17 shapes are generated by varying the order of polynomial given by j in $He_j(.)$, and the "left-triangle" shape is introduced by setting $a = -1$.*

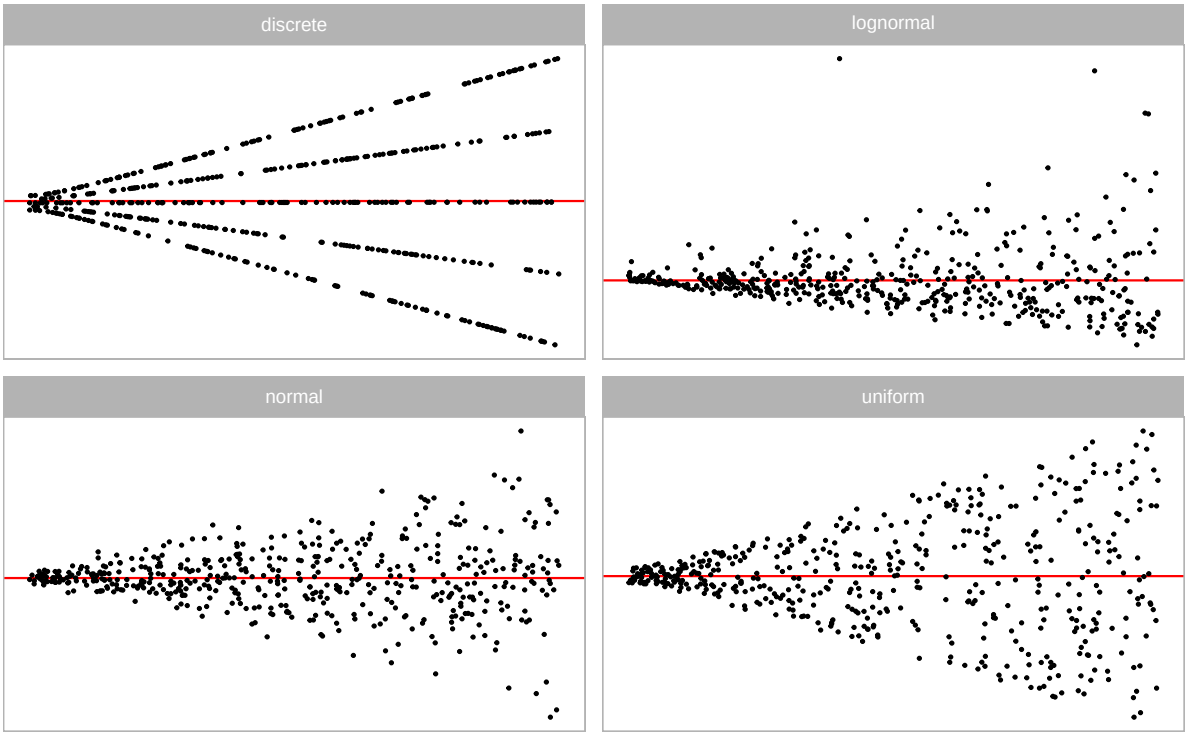


Figure 3.8: *Residual plots of models violating both the non-normality and the heteroskedasticity assumptions. The four shapes are generated by using four different error distributions including discrete, lognormal, normal and uniform, and the "left-triangle" shape is introduced by setting $a = -1$.*

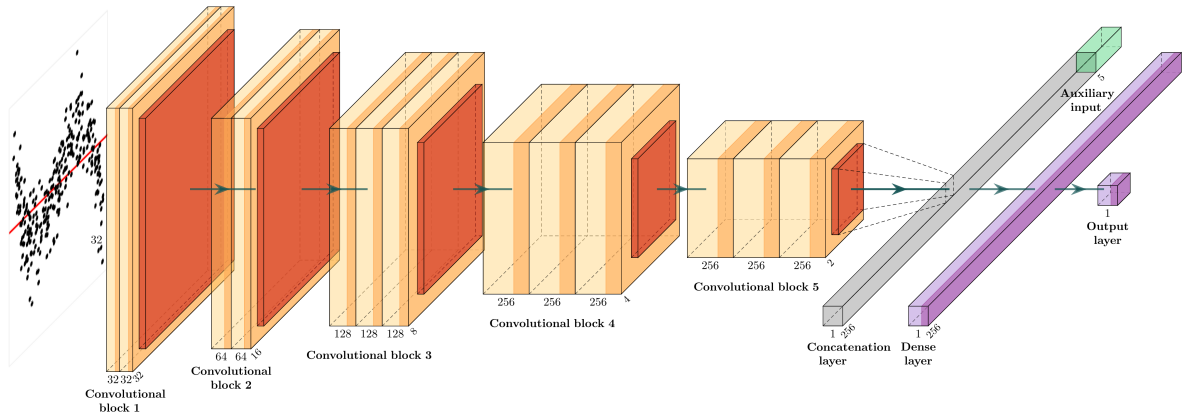


Figure 3.9: *Diagram of the architecture of the optimized computer vision model. Numbers at the bottom of each box show the shape of the output of each layer. The band of each box drawn in a darker colour indicates the use of the rectified linear unit activation function. Yellow boxes are 2D convolutional layers, orange boxes are pooling layers, the grey box is the concatenation layer, and the purple boxes are dense layers.*

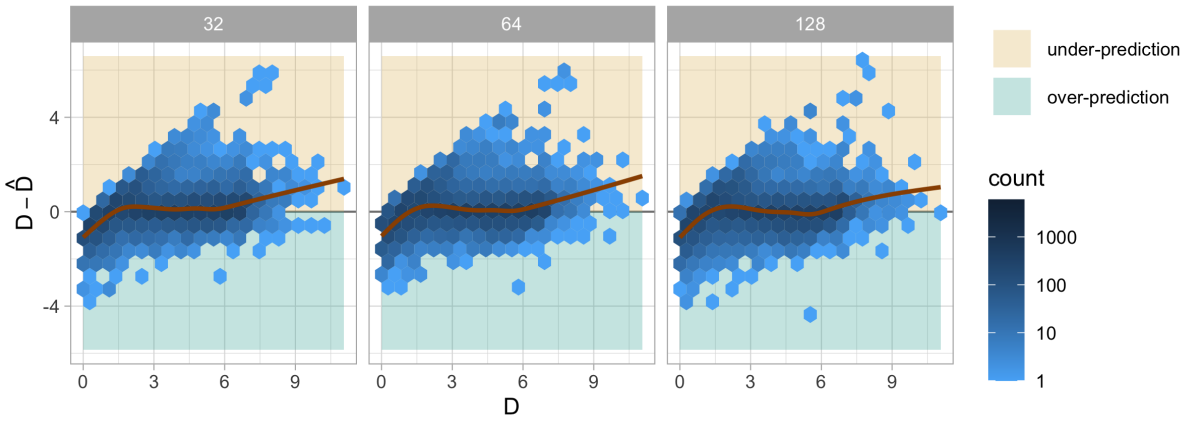


Figure 3.10: *Hexagonal heatmap for difference in D and $\hat{D}$ vs D on test data for three optimized models with different input sizes. The brown lines are smoothing curves produced by fitting gnealized additive models. The area over the zero line in light yellow indicates under-prediction, and the area under the zero line in light green indicates over-prediction.*

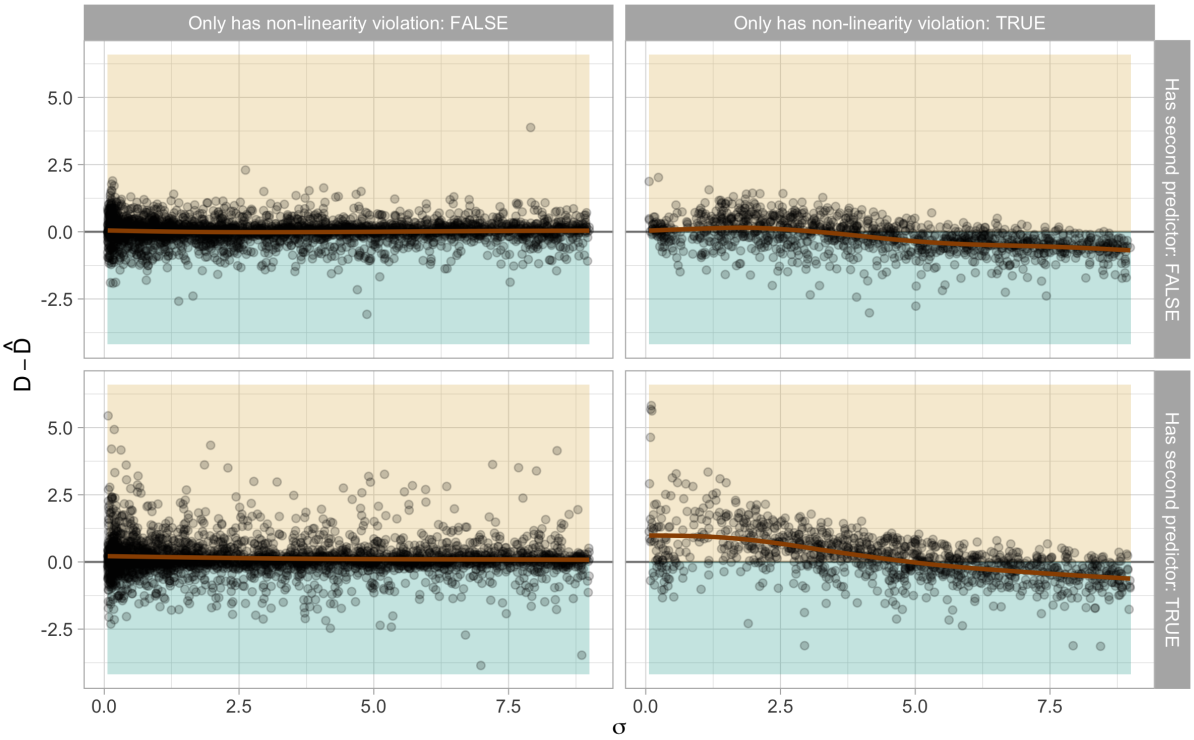


Figure 3.11: *Scatter plots for difference in D and $\hat{D}$ vs σ on test data for the 32×32 optimized model. The data is grouped by whether the regression has only non-linearity violation, and whether it includes a second predictor in the regression formula. The brown lines are smoothing curves produced by fitting gnealized additive models. The area over the zero line in light yellow indicates under-prediction, and the area under the zero line in light green indicates over-prediction.*

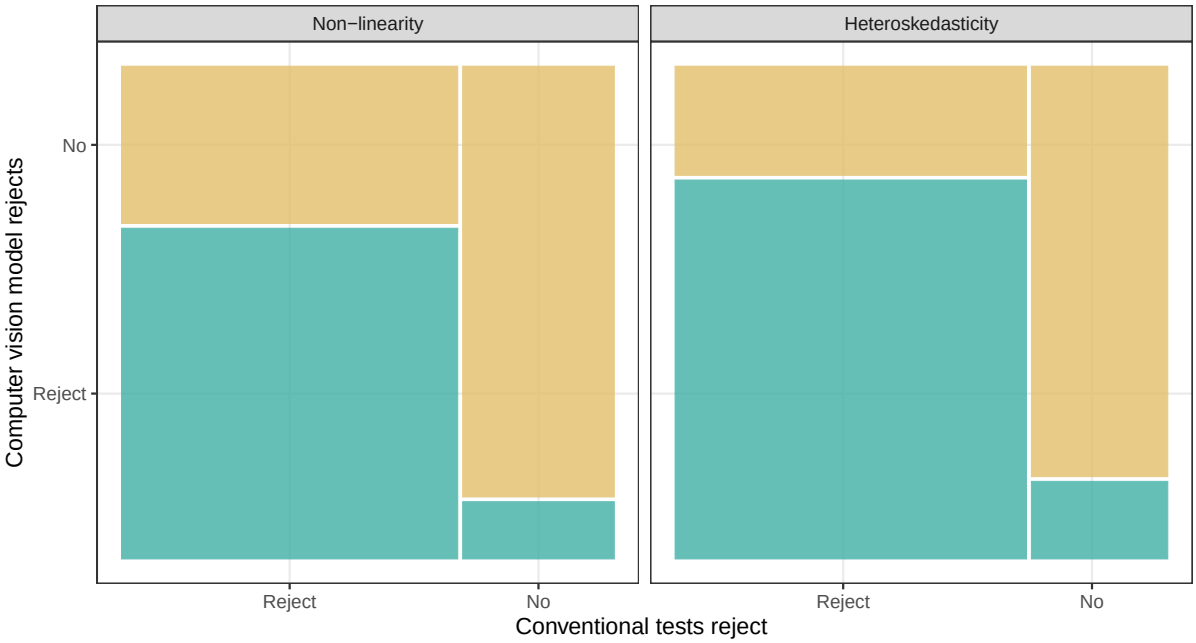


Figure 3.12: *Rejection rate (p-value ≤ 0.05) of computer vision models conditional on conventional tests on non-linearity (left) and heteroskedasticity (right) lineups displayed using a mosaic plot. When the conventional test fails to reject, the computer vision mostly fails to reject the same plot as well as indicated by the height of the top right yellow rectangle, but there are non negliable amount of plots where the conventoinal test rejects but the computer vision model fails to reject as indicated by the width of the top left yellow rectangle.*

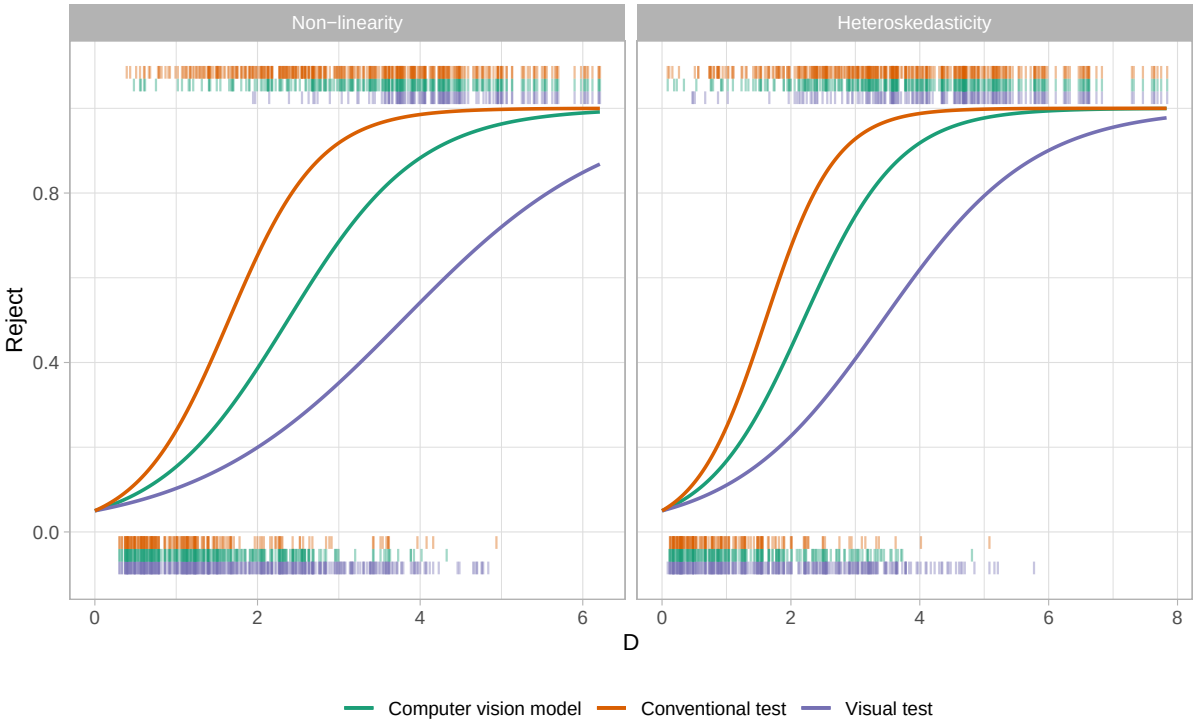


Figure 3.13: *Comparison of power of visual tests, conventional tests and the computer vision model. Marks along the x-axis at the bottom of the plot represent rejections made by each type of test. Marks at the top of the plot represent acceptances. Power curves are fitted by logistic regression models with no intercept but an offset equals to* $\log(0.05/0.95)$.

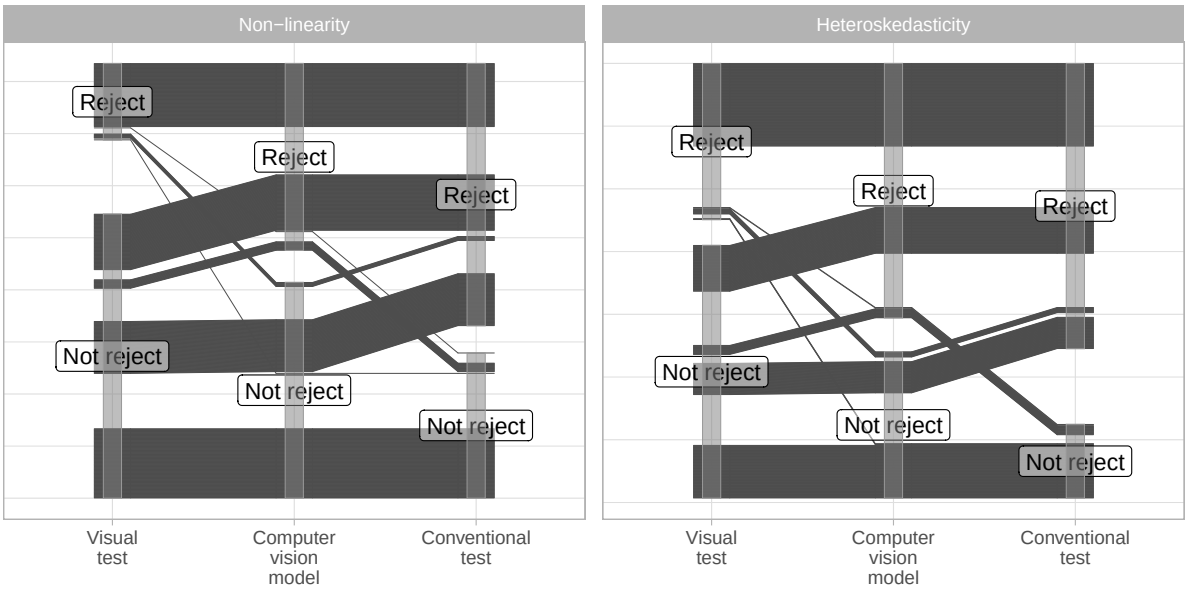


Figure 3.14: *Parallel coordinate plots of decisions made by computer vision model, conventional tests and visual tests made by human.*

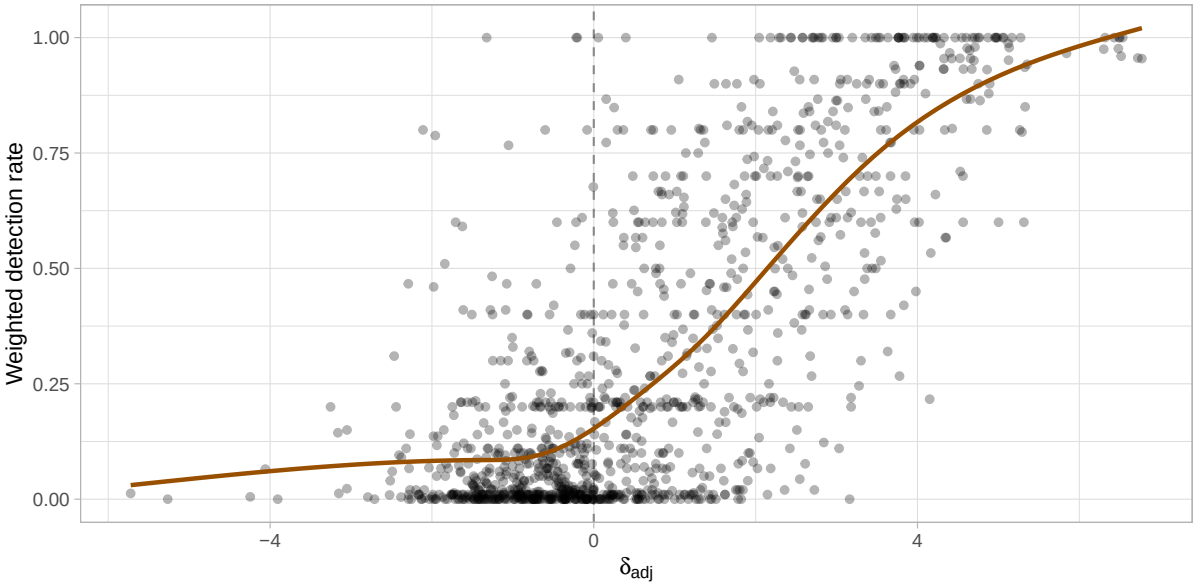


Figure 3.15: *A weighted detection rate vs adjusted δ-difference plot. The brown line is smoothing curve produced by fitting gnealized additive models.*

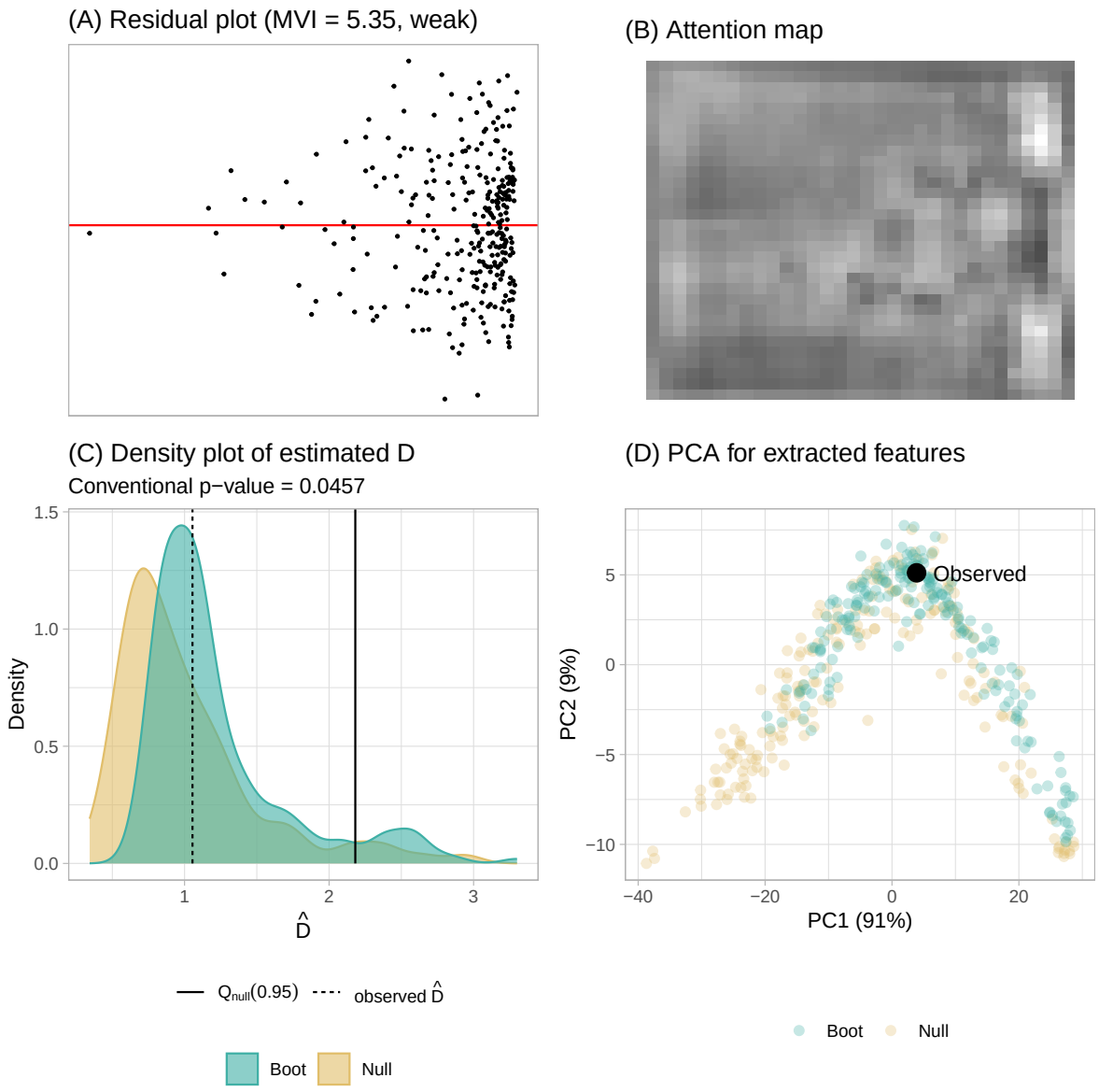


Figure 3.16: *A summary of the residual plot assessment evaluted on 200 null plots and 200 bootstrapped plots. (A) The true residual plot exhibiting a "left-triangle" shape. (B) The attention map produced by computing the gradient of the output with respect to the greyscale input. (C) The density plot of estimated distance for null plots and bootstrapped plots. The green area indicates the distribution of estimated distances for bootstrapped plots, while the yellow area represents the distribution of estimated distances for null plots. The fitted model will not be rejected since $\hat{D} < Q_{null}(0.95)$. (D) plot of first two principal components of features extracted from the global pooling layer of the computer vision model.*

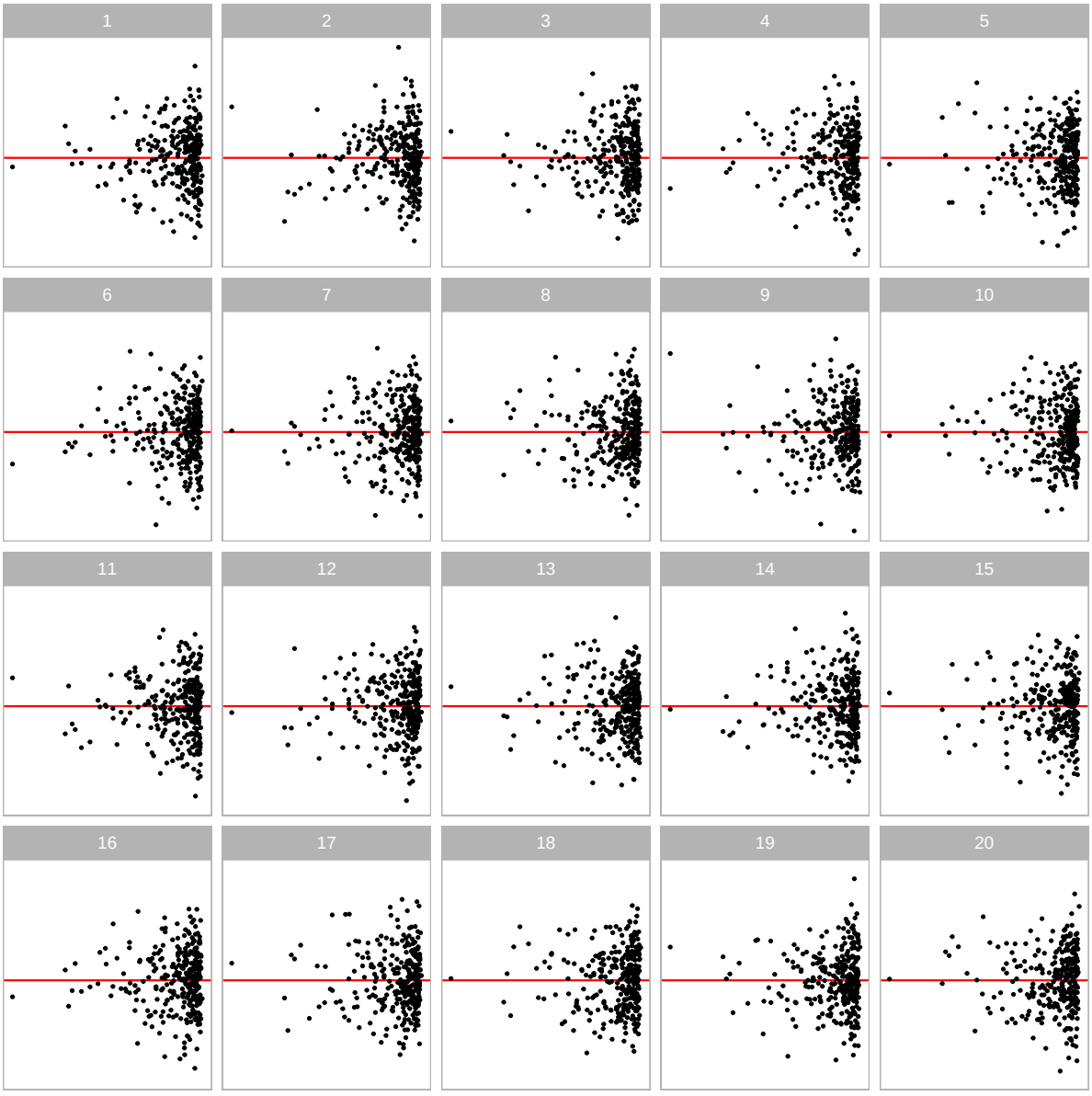


Figure 3.17: *A lineup of residual plots displaying "left-triangle" visual patterns. The true residual plot occupies position 10, yet there are no discernible visual patterns that distinguish it from the other plots.*

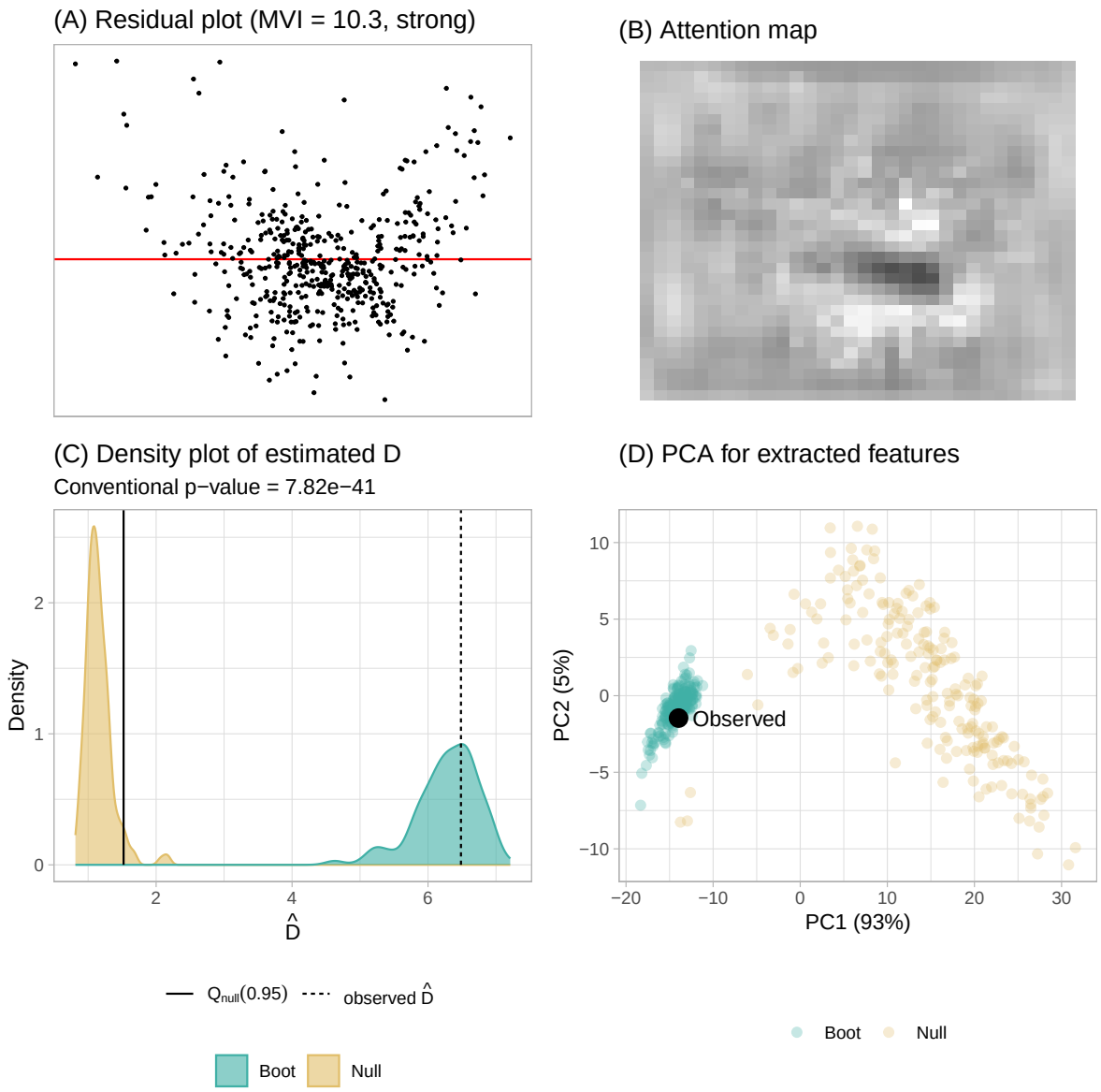


Figure 3.18: *A summary of the residual plot assessment for the Boston housing fitted model evaluted on 200 null plots and 200 bootstrapped plots. (A) The true residual plot exhibiting a "U" shape. (B) The attention map produced by computing the gradient of the output with respect to the greyscale input. (C) The density plot of estimated distance for null plots and bootstrapped plots. The blue area indicates the distribution of estimated distances for bootstrapped plots, while the yellow area represents the distribution of estimated distances for null plots. The fitted model will be rejected since* $\hat{D} \geq Q_{null}(0.95)$. *(D) plot of first two principal components of features extracted from the global pooling layer of the computer vision model.*

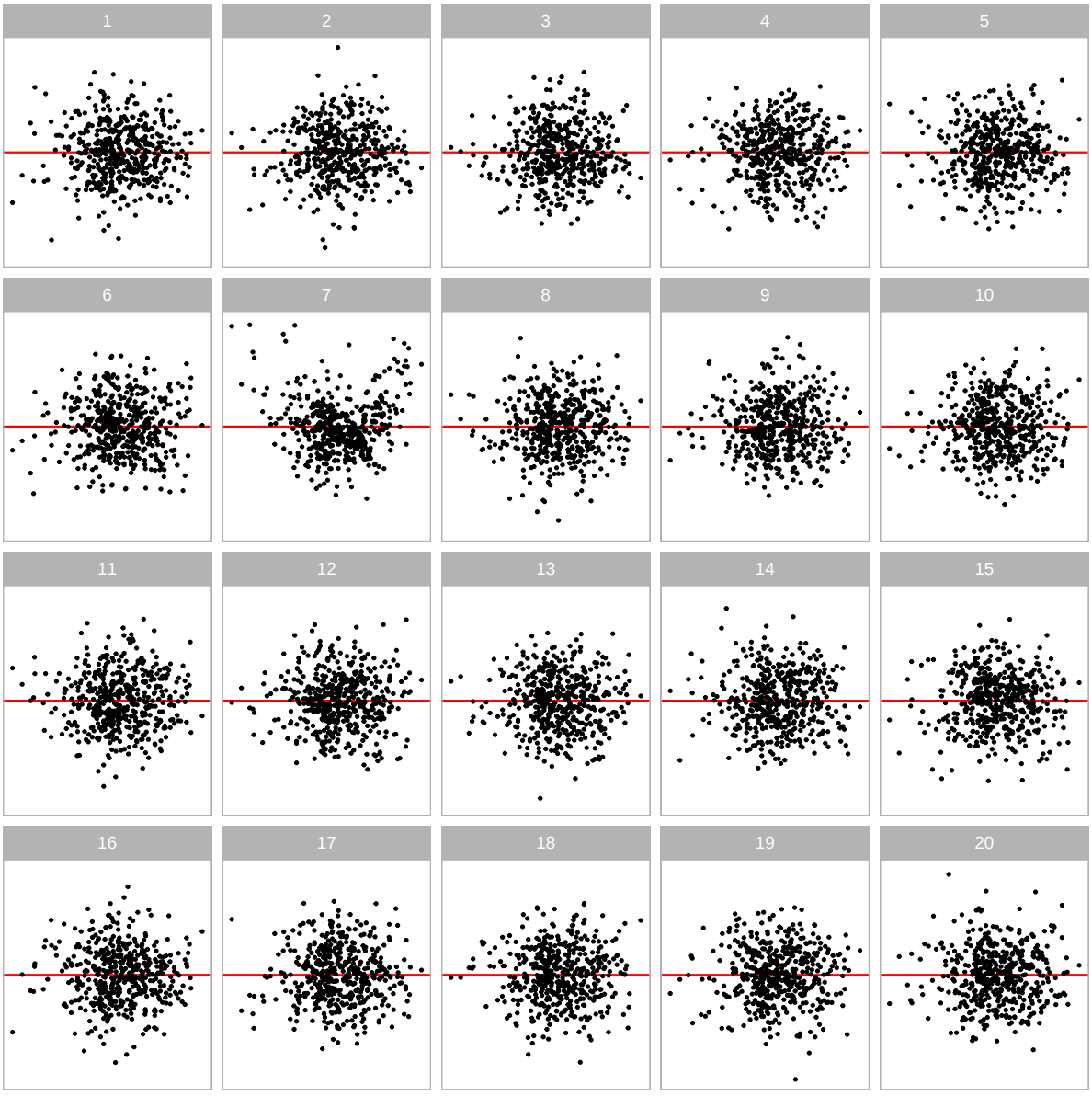


Figure 3.19: *A lineup of residual plots for the Boston housing fitted model. The true residual plot is at position 7. It can be easily identified as the most different plot.*

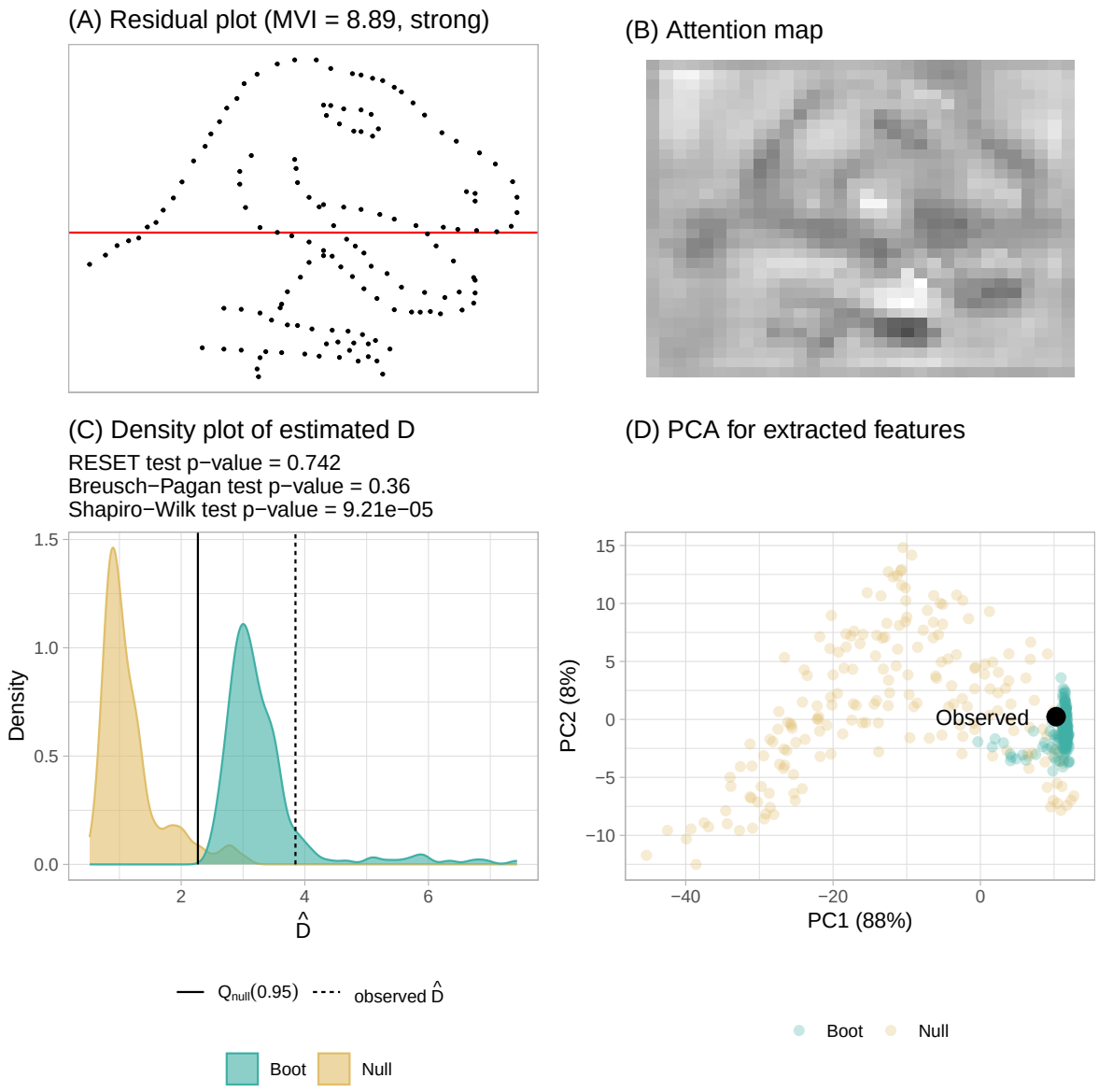


Figure 3.20: *A summary of the residual plot assessment for the datasauRus fitted model evaluted on 200 null plots and 200 bootstrapped plots. (A) The residual plot exhibits a "dinosaur" shape. (B) The attention map produced by computing the gradient of the output with respect to the greyscale input. (C) The density plot of estimated distance for null plots and bootstrapped plots. The blue area indicates the distribution of estimated distances for bootstrapped plots, while the yellow area represents the distribution of estimated distances for null plots. The fitted model will be rejected since* $\hat{D} \geq Q_{null}(0.95)$. *(D) plot of first two principal components of features extracted from the global pooling layer of the computer vision model.*

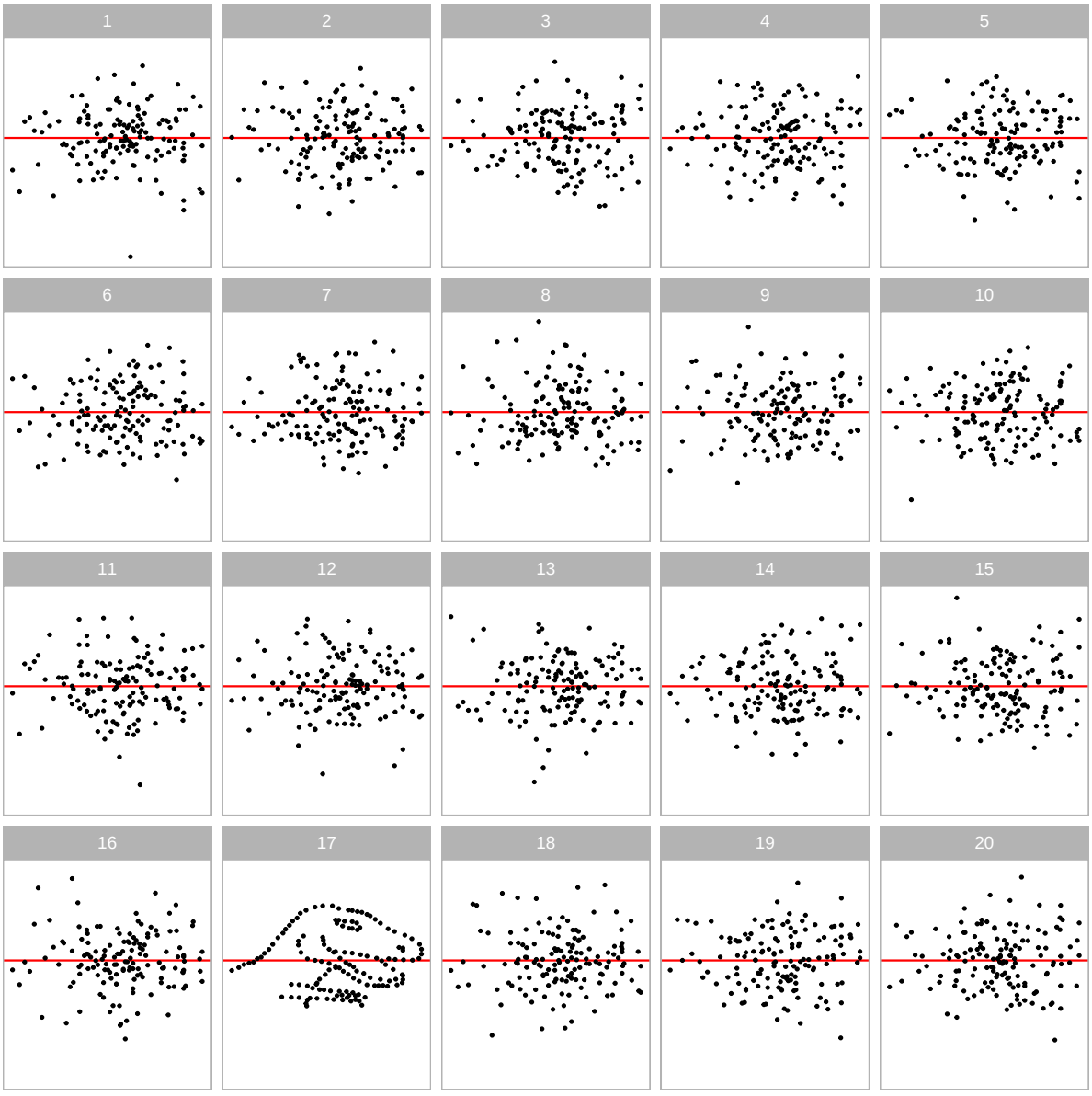


Figure 3.21: *A lineup of residual plots for the fitted model on the "dinosaur" dataset. The true residual plot is at position 17. It can be easily identified as the most different plot as the visual pattern is extremly artificial.*

Chapter 4

Software for Automated Residual Plot Assessment: autovi and autovi.web

Regression software is widely available today, but tools for effective diagnostics are still lagging. Although it is advised to diagnose a linear model by plotting residuals, it required human effort which can be prohibit the efforts. Here we describe a new R package that includes a computer vision model for automated assessment of residual plots, and an accompanying shiny app for ease of use.

4.1 Introduction

Regression analysis is a fundamental statistical technique widely used for modeling data from many fields. In modern practice, software for regression analysis tools is widely. The Comprehensive R Archive Network (CRAN) (Hornik 2012) hosts a vast array of packages, many of which diagnosing models using residual plots. These packages can be broadly categorized into three groups: general purpose, enhanced diagnostics, diagnostics with statistical testing.

General-purpose regression analysis tools are the largest and most commonly used group. These packages aren't specifically designed for graphical diagnostics of residuals in linear regression but offer this functionality as part of a broader set of statistical tools. A prime example is R's built-in `stats` package (R Core Team 2022), which provides a comprehensive collection of statistical modelling tools that includes common diagnostic plots like residuals vs fitted values, quantile-quantile (Q-Q) plots, and residuals vs leverage plots. Other packages such as `jtools` (Long 2022), `olsrr` (Hebbali 2024), `rockchalk` (Johnson 2022), and `ggResidpanel` (Goode and Rey 2019) provide similar graphical tools with alternative aesthetic styles or interactive features. Although these packages may differ in presentation, they all fundamentally deliver diagnostic plots based on well-established principles in

regression analysis, as outlined in classic works like Cook and Weisberg (1982). However, consistently drawing accurate conclusions from these tools can be challenging due to individual differences in interpreting statistical graphics. As noted in Li et al. (2024), relying solely on subjective assessments of data plots can lead to problems, such as over-interpreting random patterns as model violations.

Enhanced visual diagnostics is the second group, which offers advanced visual aids for interpreting diagnostic plots. A notable example is the `DHARMa` package (Hartig 2022), which uses an innovative approach by fitting quantile regression on scaled residual plots. It compares the empirical 0.25, 0.5, and 0.75 quantiles in scaled residuals with their theoretical counterparts, making it particularly useful for highlighting deviations from model assumptions, detecting model violations such as heteroscedasticity and incorrect functional forms, and uncovering issues specific to generalized linear models and mixed-effect models, like over/underdispersion. By offering these enhanced visualizations, `DHARMa` enables users to more easily identify potential issues in their regression models that might not be immediately apparent with standard diagnostic plots. In summary, this group of packages enhances the interpretability of diagnostic plots by drawing attention to critical elements such as trends, clusters, and outliers. They sometimes automatically perform conventional tests on these elements, displaying the results as annotations, labels, or text within the plot, thereby further reducing the likelihood of misinterpretation.

Statistical testing for visual discoveries is the third group, which focuses on providing tools for conducting formal statistical tests for visual discoveries obtained from diagnostic plots (Buja et al. 2009b). Examples in this category include the `nullabor` (Wickham et al. 2020) and `regressinator` (Reinhart 2024) packages, which enables users to quantify the significance of patterns observed in residual diagnostic plots, perform hypothesis tests on specific aspects of model fit, and validate visual interpretations with statistical evidence. This approach addresses the issue of inconsistent interpretation of diagnostic plots by bridging the gap between visual inspection and formal statistical inference, thus offering a more robust framework for regression diagnostics.

Conducting a visual test for a residual plot, which is among the most common diagnostic plots in regression analysis, involves using a lineup protocol. In this protocol, the true residual plot is embedded within a lineup of several null plots and presented to one or more observers. The null plots are created by simulating residuals consistent with the null hypothesis H_0 that the regression model is correctly specified. Observers are then asked to identify the plot that appears most different from the others. If a significant percentage of observers correctly identify the true residual plot, it provides evidence against the H_0, as according to Buja et al. (2009b), the true residual plot would have no distinguishable difference from the null plots if all residuals are generated by the same process.

However, as discussed in Chapter 3, the lineup protocol has significant limitations in large-scale applications due to its high labor costs and time consumption. To address these issues, we developed a computer vision model and an associated statistical testing procedure to automate the assessment of residual plots. This model takes a residual plot and a vector of auxiliary variables (such as the number of observations) as inputs and outputs a visual signal strength. This strength estimates the distance between the residual distribution of the fitted regression model and the reference distribution assumed under correct model specification.

By estimating the visual signal strength for all plots in a lineup, we can compute a p-value based on the ratio of plots with visual signal strength greater than or equal to that of the true residual plot. This p-value has a lower bound of one divided by the number of plots in the lineup. Additionally, we can construct a null distribution of visual signal strength using the strengths of null plots.

We can also apply bootstrapping to obtain a distribution of visual signal strengths. This involves bootstrapping the data used to fit the linear regression model, refitting the model to obtain bootstrapped residuals, and then using the computer vision model to predict visual signal strength for these residuals. The resulting bootstrapped distribution can be compared against the null distribution. If these distributions are largely similar, it suggests that the bootstrapped residual plots are similar to the null plots. The proportion of bootstrapped visual signal strengths exceeding a critical value (such as the 95% sample quantile of the null distribution) indicates how often the assumed regression model would be considered incorrect if the data could be repeatedly obtained from the same data-generating process.

To make the statistical testing procedure and trained computer vision model widely accessible, we developed the R package `autovi`. In addition, we created a web-based tool that offers a user-friendly interface, enabling users to diagnose their residual plots without the need to install any dependencies required by the `autovi` package.

The remainder of this chapter is structured as follows: Section 4.2 provides a detailed documentation of the `autovi` package, including its usage and infrastructure. Section 4.3 focuses on the `autovi.web` interface, describing its design and usage, along with illustrative examples. Finally, Section 4.4 presents the main conclusions of this work.

4.2 R package: autovi

The main purpose of `autovi` is to provide rejection decisions and p-values for testing whether a regression model is correctly specified. The package introduces a novel approach to automating statistical analysis, particularly in the interpretation of residual plots. The name `autovi` stands for

automated visual inference. While initially designed for linear regression residual diagnostics, it has the potential to be extended to broader visual inference applications, as we'll discuss in section Section 4.2.4.

4.2.1 Implementation

`autovi` is built upon the `bandicoot` object-oriented programming (OOP) system (Li 2024), which marks a departure from R's conventional S3 generic system. The adoption of an OOP architecture enhances flexibility and modularity, enabling users to redefine key functions within the infrastructure through method overriding. While similar functionality could be replicated using R's S3 system with generic functions, the OOP system offers a more structured and extensible foundation for the package.

The infrastructure of `autovi` demonstrates the effective integration of multiple programming languages and libraries to create a comprehensive analytical tool. It depends on five core libraries from Python and R, each contributing critically to the analysis pipeline. In Python, `pillow` (Clark et al. 2015) handles image processing tasks, including reading PNG files of residual plots, resizing them, and converting them into input tensors for further analysis. The `TensorFlow` (Abadi et al. 2016) library, a cornerstone of contemporary machine learning, is employed to predict the visual signal strength of these residual plots, utilizing a pre-trained convolutional neural network.

Within the R environment, `autovi` utilizes several powerful libraries. `ggplot2` (Wickham 2016) is employed to generate the initial residual plots, which are then saved as PNG files, using as the primary visual input for the analysis. The `cassowaryr` (Mason et al. 2022) library calculates scagnostics (scatter plot diagnostics) of the residual plots, providing numerical features that capture various statistical properties of the plots. These scagnostics complement the visual analysis by supplying quantitative metrics as secondary input to the computer vision model. The `reticulate` (Ushey et al. 2024) package is used to bridge R and Python, allowing for seamless communication between the two languages and supporting an integrated infrastructure.

The package includes internal functions to check the current Python environment used by the `reticulate` package. If the necessary Python packages are not installed in the Python interpreter, an error will be raised. If you want to select a specific Python environment, you can do so by calling the `reticulate::use_python()` function before using the `autovi` package.

4.2.2 Installation

The `autovi` package is available on CRAN. It is actively developed and maintained, with the latest updates accessible on GitHub at https://github.com/TengMCing/autovi. The code discussed in this chapter is based on `autovi` version 0.4.1.

4.2.3 Usage

To get started quickly, users need only three lines of code to obtain a summary of the automated residual assessment:

```
library(autovi)
checker <- residual_checker(fitted_model = lm(dist ~ speed, data = cars))
checker$check()
```

```
-- <AUTO_VI object>
Status:
 - Fitted model: lm
 - Keras model: UNKNOWN
    - Output node index: 1
 - Result:
    - Observed visual signal strength: 3.162 (p-value = 0.0396)
    - Null visual signal strength: [100 draws]
       - Mean: 1.274
       - Quantiles:
```

25%	50%	75%	80%	90%	95%	99%
0.8021	1.1109	1.5751	1.6656	1.9199	2.6564	3.3491

```
    - Bootstrapped visual signal strength: [100 draws]
       - Mean: 2.786 (p-value = 0.05941)
       - Quantiles:
```

25%	50%	75%	80%	90%	95%	99%
2.452	2.925	3.173	3.285	3.463	3.505	3.652

```
    - Likelihood ratio: 0.7275 (boot) / 0.06298 (null) = 11.55
```

1. Load the package using the `library()` function.
2. Construct a checker with two inputs: a linear regression model and a pre-trained Keras model (Chollet et al. 2015).
3. Use `get_keras_model()`, a function provided by `autovi`, to download a trained computer

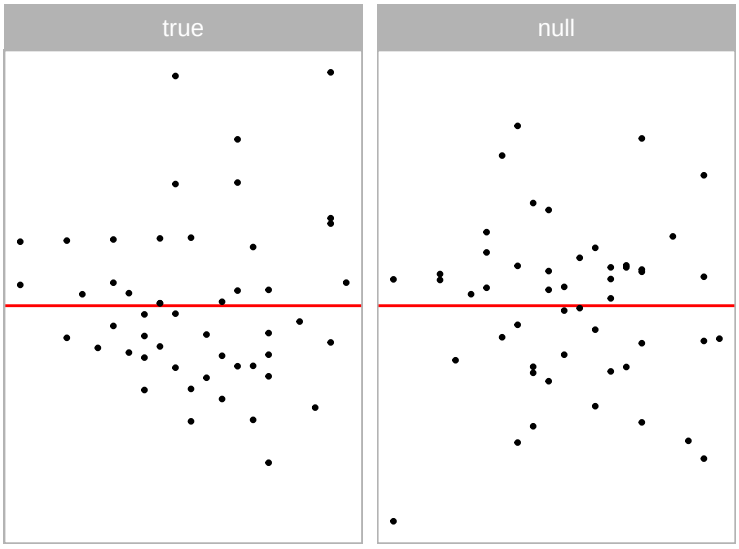
vision model (described in Chapter 3) from GitHub. "vss_phn_32" specifies a model that predicts visual signal strength (vss) and is trained on residuals with polynomial, heteroskedasticity, and non-normality patterns (phn). More details about the hosted models will be provided in section Section 4.2.7.

4. Call the `check()` method of the checker with default arguments. This predicts the visual signal strength for the true residual plot, 100 null plots, and 100 bootstrapped plots, storing the predictions internally.
5. Use the `print()` function to generate a concise report of the check results.

The report highlights key findings such as the visual signal strength of the true residual plot and the p-value of the automated visual test. The p-value is the ratio of null plots having visual signal strength greater than or equal to the true residual plot. We typically reject the null hypothesis when the p-value is smaller than or equal to 5%. The report also provides sample quantiles of visual signal strength for null and bootstrapped plots, helping to explain the severity and likelihood of model violations.

Although the p-value is sufficient for automated decision-making, users are strongly encouraged to visually inspect the original residual plot alongside a sample null plot. This visual comparison can clarify why H_0 is either rejected or not, and help identify potential remedies. The `plot_pair()` method facilitates this comparison.

```
checker$plot_pair()
```

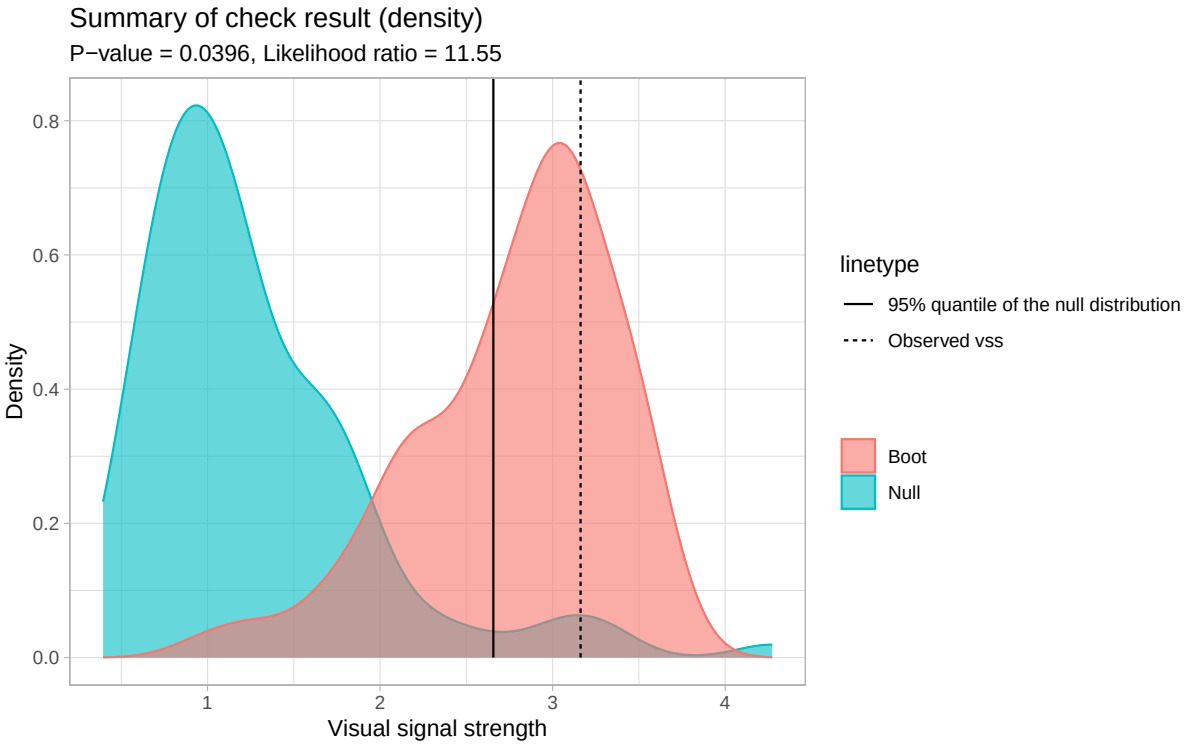


This method displays the true residual plot on the left and a null plot on the right. Users should look for any distinct visual patterns in the true residual plot that are absent in the null plot. It's recommended to run this function multiple times to confirm any visual findings, as each execution

generates a new random null plot for comparison.

The package offers a straightforward visualization of the assessment result through the `summary_plot()` function.

```
checker$summary_plot()
```



In the visualization, the blue area represents the density of visual signal strength for null residual plots, while the red area shows the density for bootstrapped residual plots. The dashed line indicates the visual signal strength of the true residual plot, and the solid line marks the critical value at a 95% significance level. The p-value and the likelihood ratio are displayed in the subtitle. The likelihood ratio represents the ratio of the likelihood of observing the visual signal strength of the true residual plot from the bootstrapped distribution compared to the null distribution.

Interpreting the plot involves several key aspects. If the dashed line falls to the right of the solid line, it suggests rejecting the null hypothesis. The degree of overlap between the red and blue areas indicates similarity between the true residual plot and null plots; greater overlap suggests more similarity. Lastly, the portion of the red area to the right of the solid line represents the percentage of bootstrapped models considered to have model violations.

This visual summary provides an intuitive way to assess the model's fit and potential violations, allowing users to quickly grasp the results of the automated analysis.

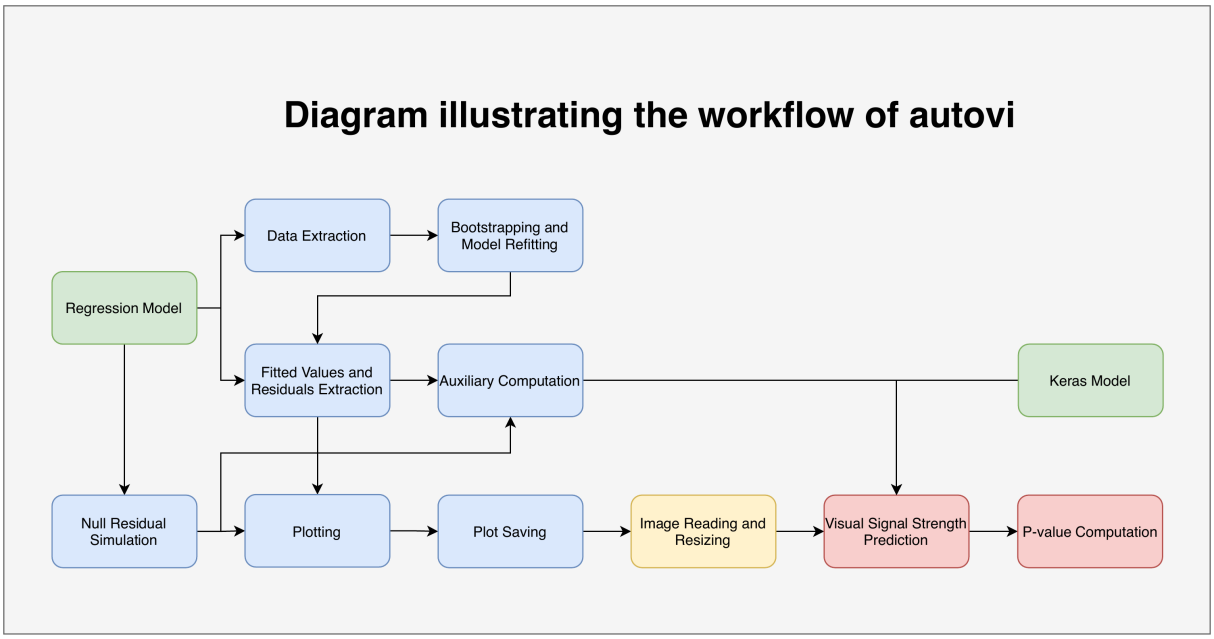


Figure 4.1: *Diagram illustrating the infrastructure of the R package autovi. The modules in green are primary inputs provided by users. Modules in blue are overridable methods that can be modified to accommodate users' specific needs. The module in yellow is a pre-defined non-overridable method. The modules in red are primary outputs of the package.*

4.2.4 Modularized Infrastructure

The initial motivation for developing `autovi` was to create a convenient interface for sharing the models described and trained in Chapter 3. However, recognizing that the classical normal linear regression model represents a restricted class of models, we sought to avoid limiting the potential for future extensions, whether by the original developers or other users. As a result, the package was designed to function seamlessly with linear regression models with minimal modification and few required arguments, while also accommodating other classes of models through partial infrastructure substitution. This modular and customizable design allows `autovi` to handle a wide range of residual diagnostics tasks.

The infrastructure of `autovi` consists of ten core modules: data extraction, bootstrapping and model refitting, fitted values and residuals extraction, auxiliary computation, null residual simulation, plotting, plot saving, image reading and resizing, visual signal strength prediction, and p-value computation. Each module is designed with minimal dependency on the preceding modules, allowing users to customize parts of the infrastructure without affecting its overall integrity. An overview of this infrastructure is illustrated in Figure Figure 4.1.

The modules for visual signal strength prediction and p-value computation are predefined and cannot be overridden, although users can interact with them directly through function arguments. Similarly, the image reading and resizing module is fixed but will adapt to different Keras models by checking

their input shapes. The remaining seven modules are designed to be overridable, enabling users to tailor the infrastructure to their specific needs. These modules will be discussed in detail in the following sections.

4.2.4.1 Initialization

An `autovi` checker can be initialized by supplying two primary inputs, including a regression model object, such as an `lm` object representing the result of a linear regression model, and a trained computer vision model compatible with the `Keras` (Chollet et al. 2015) Application Programming Interface (API), to the `AUTO_VI` class constructor `auto_vi()`. The input will be stored in the checker and can be accessed by the user through the `$` operator.

```
library(autovi)
checker <- auto_vi(fitted_model = lm(dist ~ speed, data = cars),
                   keras_model = get_keras_model("vss_phn_32"))
```

Optionally, the user may specify the node index of the output layer of the trained computer vision model to be monitored by the checker via the `node_index` argument if there are multiple output nodes. This is particularly useful for multiclass classifiers when the user wants to use one of the nodes as a visual signal strength indicator.

After initializing the object, you can print the checker to view its status.

```
checker
```

```
-- <AUTO_VI object>
Status:
 - Fitted model: lm
 - Keras model: UNKNOWN
    - Output node index: 1
 - Result:
    - Observed visual signal strength: 3.162 (p-value = 0.0396)
    - Null visual signal strength: [100 draws]
       - Mean: 1.274
       - Quantiles:
```

25%	50%	75%	80%	90%	95%	99%
0.8021	1.1109	1.5751	1.6656	1.9199	2.6564	3.3491

```
╚═══════════════════════════════════════════════════════╝
```

```
  - Bootstrapped visual signal strength: [100 draws]
    - Mean: 2.786 (p-value = 0.05941)
    - Quantiles:
```

```
╔═══════════════════════════════════════════════════════╗
  25%   50%   75%   80%   90%   95%   99%
2.452 2.925 3.173 3.285 3.463 3.505 3.652
╚═══════════════════════════════════════════════════════╝
```

```
  - Likelihood ratio: 0.7275 (boot) / 0.06298 (null) = 11.55
```

The status includes the list of regression model classes (as provided by the built-in `class()` function), the input and output shapes of the Keras model in the standard `Numpy` format (Harris et al. 2020), the output node index being monitored, and the assessment result. If no check has been run yet, the assessment result will display as "UNKNOWN".

4.2.4.2 Fitted Values and Residuals Extraction

To be able to predict visual signal strength for a residual plot, both fitted values and residuals are needed to be extracted from the regression model object supplied by the user. In R, statistical models like `lm` (linear model) and `glm` (generalized linear model) typically support the use of generic functions such as `fitted()` and `resid()` to retrieve these values. The `get_fitted_and_resid()` method, called by the checker, relies on these generic functions by default. However, generic functions only work with classes that have appropriate method implementations. Some regression modelling packages may not fully adhere to the `stats` package guidelines for implementing these functions. In such cases, overriding the method becomes necessary.

By design, the `get_fitted_and_resid()` method accepts a regression model object as input and returns a `tibble` with two columns: `.fitted` and `.resid`, representing the fitted values and residuals, respectively. If no input is supplied, the method uses the regression model object stored in the checker. Although modules in the `autovi` infrastructure make minimal assumptions about other modules, they do require strictly defined input and output formats to ensure data validation and prevent fatal bugs. Therefore, any overridden method should follow to these conventions.

```
checker$get_fitted_and_resid()
```

```
# A tibble: 50 x 2
   .fitted .resid
     <dbl>  <dbl>
 1   -1.85   3.85
```

```
 2   -1.85  11.8
 3    9.95  -5.95
 4    9.95  12.1
 5   13.9    2.12
 6   17.8   -7.81
 7   21.7   -3.74
 8   21.7    4.26
 9   21.7   12.3
10   25.7   -8.68
# i 40 more rows
```

4.2.4.3 Data Extraction

For linear regression model in R, the model frame contains all the data required by a formula for evaluation. This is essential for bootstrapping and refitting the model when constructing a bootstrapped distribution of visual signal strength. Typically, the model frame can be extracted from the regression model object using the `model.frame()` generic function, which is the default method used by `get_data()`. However, some regression models don't use a formula or are evaluated differently, potentially lacking a model frame. In such cases, users can either provide the data used to fit the regression model through the `data` argument when constructing the checker, or customize the method to better suit their needs. It's worth noting that this module is only necessary if bootstrapping is required, as the model frame is not used in other modules of the infrastructure.

The `get_data()` method accepts a regression model object as input and returns a `data.frame` representing the model frame of the fitted regression model. If no input is supplied, the regression model stored in the checker will be used.

```
checker$get_data() |>
  head()
```

```
  dist speed
1    2     4
2   10     4
3    4     7
4   22     7
5   16     8
6   10     9
```

4.2.4.4 Bootstrapping and Model Refitting

Bootstrapping a regression model typically involves sampling the observations with replacement and refitting the model with the bootstrapped data. The `boot_method()` method follows this bootstrapping scheme by default. It accepts a fitted regression model and a `data.frame` as inputs, and returns a `tibble` of bootstrapped residuals. If no inputs are provided, the method uses the regression model stored in the checker and the result of the `get_data()` method.

Note that instead of calling `get_data()` implicitly within the method, it is used as part of the default argument definition. This approach allows users to bypass the `get_data()` method entirely and directly supply a `data.frame` to initiate the bootstrap process. Many other methods in `autovi` adopt this principle when possible, where dependencies are explicitly listed in the formal arguments. This design choice enhances the reusability and isolation of modules, offers better control for testing, and simplifies the overall process.

```
checker$boot_method(data = checker$get_data())
```

```
# A tibble: 50 x 2
   .fitted .resid
     <dbl>  <dbl>
 1   27.0   -2.96
 2   38.8  -12.8
 3   34.8   -8.82
 4   27.0  -13.0
 5   11.2    4.76
 6   42.7   -2.68
 7   42.7   -2.68
 8   38.8  -18.8
 9   38.8   15.2
10   -4.47   6.47
# i 40 more rows
```

4.2.4.5 Auxiliary Computation

According to Chapter 3, in some cases, a residual plot alone may not provide enough information to accurately determine visual signal strength. For instance, when the residual plot has significant overlap, the trend and shape of the residual pattern can be difficult to discern. Including auxiliary variables, such as the number of observations, as additional inputs to the computer vision model can be beneficial. To address this, `autovi` includes internal functions within the checker that automatically

detect the number of inputs required by the provided Keras model. If multiple inputs are necessary, the checker invokes the `auxiliary()` method to compute these additional inputs.

The `auxiliary()` method takes a `data.frame` containing fitted values and residuals as input and returns a `data.frame` with five numeric columns. These columns represent four scagnostics — "Monotonic", "Sparse", "Striped", and "Splines" — calculated using the `cassowaryr` package, as well as the number of observations. This approach is consistent with the training process of the computer vision models described in Chapter 3. If no `data.frame` is provided, the method will default to retrieving fitted values and residuals by calling `get_fitted_and_resid()`.

Technically, any Keras-implemented computer vision model can be adapted to accept an image as the primary input and additional variables as secondary inputs by adding a data pre-processing layer before the actual input layer. If users wish to override `auxiliary()`, the output should be a `data.frame` with a single row and the number of columns matching the supplied Keras model.

```
checker$auxiliary()
```

```
# A tibble: 1 x 5
  measure_monotonic measure_sparse measure_splines measure_striped     n
              <dbl>          <dbl>            <dbl>            <dbl> <int>
1            0.0621          0.470           0.0901             0.62    50
```

4.2.4.6 Null Residual Simulation

A fundamental element of the automated residual assessment described in Chapter 3 is comparing the visual signal strength of null plots with that of the true residual plot. However, due to the variety of regression models, there is no universal method for simulating null residuals that are consistent with model assumptions. Fortunately, for classical normal linear regression models, null residuals can be effectively simulated using the residual rotation method, as outlined in Buja et al. (2009b). This process involves generating random draws from a standard normal distribution, regressing these draws on the original predictors, and then rescaling the resulting residuals by the ratio of the residual sum of squares to the that of the original linear regression model. Other regression models, such as `glm` (generalized linear model) and `gam` (generalized additive model), generally cannot use this method to efficiently simulate null residuals. Therefore, it is recommended that users override the `null_method()` to suit their specific model. The `null_method()` takes a fitted regression model as input, defaulting to the regression model stored in the checker, and returns a `tibble`.

```
checker$null_method()
```

```
# A tibble: 50 x 2
```

```
   .fitted  .resid
     <dbl>   <dbl>
 1   -1.85  18.2
 2   -1.85  -0.765
 3    9.95 -12.8
 4    9.95  18.6
 5   13.9    2.57
 6   17.8    7.03
 7   21.7  -11.1
 8   21.7  -13.2
 9   21.7  -12.6
10   25.7    3.57
# i 40 more rows
```

4.2.4.7 Plotting

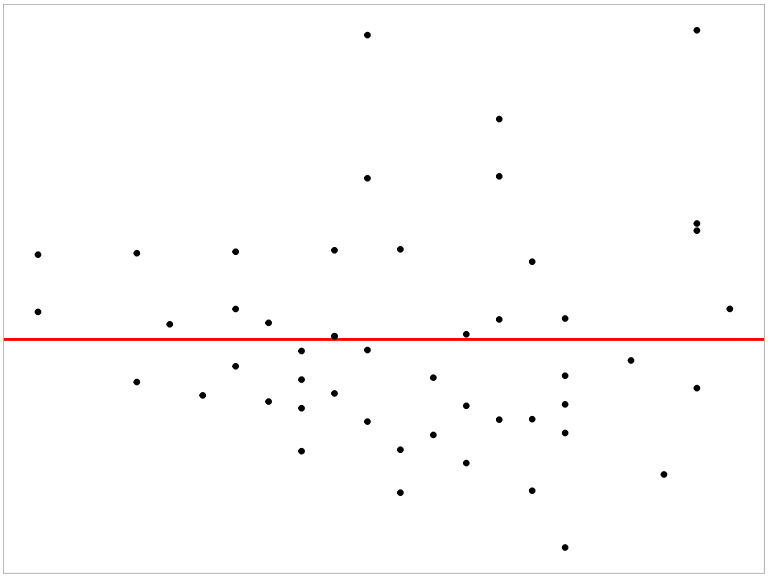
Plotting is a crucial aspect of residual plot diagnostics because aesthetic elements like marker size, marker color, and auxiliary lines impact the presentation of information. There are computer vision models trained to handle images captured in various scenarios. For example, the VGG16 model (Simonyan and Zisserman 2014) can classify objects in images taken under different lighting conditions and is robust to image rotation. However, data plots are a special type of image as the plotting style can always be consistent if controlled properly. Therefore, we assume computer vision models built for reading residual plots will be trained with residual plots of a specific aesthetic style. In this case, it is best to predict plots using the same style for optimal performance. The plotting method `plot_resid()` handles this aspect.

`plot_resid()` accepts a `data.frame` containing fitted values and residuals, along with several customization options: a `ggplot` theme, an `alpha` value to control the transparency of data points, a `size` value to set the size of data points, and a `stroke` value to define the thickness of data point edges. Additionally, it includes four Boolean arguments to toggle the display of axes, legends, grid lines, and a horizontal red line. By default, it replicates the style we used to generate the training samples for the computer vision models described in Chapter 3. In brief, the residual plot omits axis text and ticks, titles, and background grid lines, featuring only a red line at $y = 0$. It retains only the necessary components of a residual plot. If the computer vision model is trained with a different but consistent aesthetic style, `plot_resid()` should be overridden.

The method returns a `ggplot` object, which can be saved as a PNG file in the following module. If no data is provided, the method will use `get_fitted_and_resid()` to retrieve the fitted values and

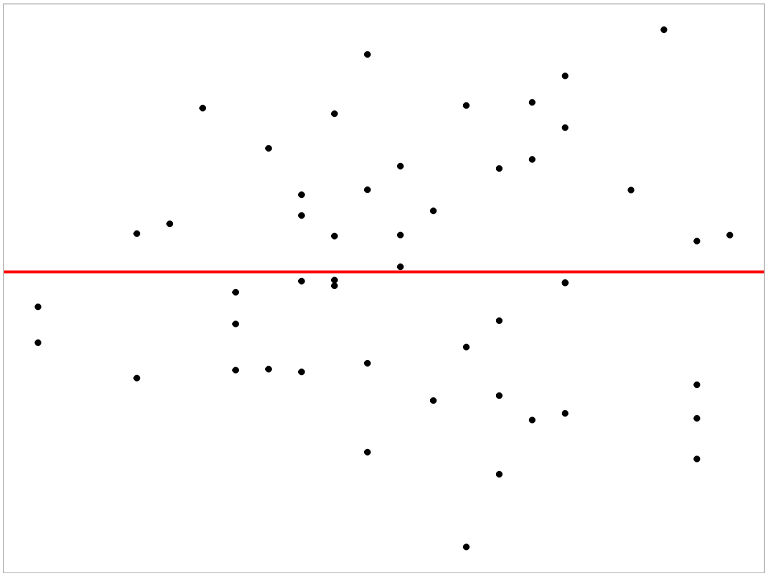
residuals from the regression model stored in the checker.

```
checker$plot_resid()
```



To manually generate true residual plots, null plots, or bootstrapped residual plots, you can pass the corresponding `data.frame` produced by the `get_fitted_and_resid()`, `null_method()`, and `boot_method()` methods to the `plot_resid()` method, respectively.

```
checker$null_method() |>
  checker$plot_resid()
```



4.2.4.8 Plot Saving

Another key aspect of a standardized residual plot is its resolution. In Chapter 3, we used an image format of 420 pixels in height and 525 pixels in width. This resolution was chosen because the original set, consisting of 20 residual plots arranged in a four by five grid, was represented by an

image of 2100 by 2100 pixels. The `save_plot()` method takes a `ggplot` object as input, saves it as a temporary PNG file, and returns the file path as a string. Note that the `save_plot()` method does not have default arguments, as it is not intended to be called without a plot. While an alternative design could be to save the true residual plot by default, this might be confusing for users, given that the method's name does not fully convey this functionality.

```
checker$plot_resid() |>
  checker$save_plot()
```

```
[1] "/var/folders/61/bv7_1qzs20x6fjb2rsv7513r0000gn/T//RtmpbGVhmt/file1cbb78ea04a1.png"
```

4.2.4.9 Image Reading and Resizing

When training computer vision models, it is common to test various input sizes for the same architecture to identify the optimal setup. This involves preparing the original training image at a higher resolution than required and then resizing it to match the input size during training. The `autovi` package includes a class, `KERAS_WRAPPER`, to simplify this process. This Keras wrapper class features a method called `image_to_array()`, which reads an image as a `PIL` image using the `pillow` Python package, resizes it to the target input size required by the Keras model, and converts it to a `Numpy` array.

To construct a `KERAS_WRAPPER` object, you need to provide the Keras model as the main argument. However, users generally do not need to interact with this class directly, as the `autovi` checker automatically invokes its methods when performing visual signal strength predictions. The `image_to_array()` method takes the path to the image file, the target height, and the target width as inputs and returns a `Numpy` array. If not specified, the target height and target width will be retrieved from the input layer of the Keras model by the `get_input_height()` and `get_input_width()` method of `KERAS_WRAPPER`.

The following code example demonstrate the way to manually generate the true residual plot, save it as PNG file, and load it back as `Numpy` array.

```
wrapper <- keras_wrapper(keras_model = checker$keras_model)
input_array <- checker$plot_resid() |>
  checker$save_plot() |>
  wrapper$image_to_array()
input_array$shape
```

```
(1, 32, 32, 3)
```

4.2.4.10 Visual Signal Strength Prediction

Visual signal strength, as discussed in Chapter 3, estimates the distance between the input residual plot and a theoretically good residual plot. It can be defined in various ways, much like different methods for measuring the distance between two points. This will not impact the `autovi` infrastructure as long as the provided Keras model can predict the intended measure.

There are several ways to obtain visual signal strength from the checker, with the most direct being the `vss()` method. By default, this method predicts the visual signal strength for the true residual plot. If a `ggplot` or a `data.frame`, such as null residuals generated by the `null_method()`, is explicitly provided, the method will use that input to predict visual signal strength accordingly. Note that if a `ggplot` is provided, auxiliary inputs must be supplied manually via the `auxiliary` argument, as we assume that auxiliary variables can not be computed directly from a `ggplot`.

Another way to obtain visual signal strength is by calling the `check()` method. This comprehensive method perform extensive diagnostics on the true residual plot and store the visual signal strength in the `check_result` field of the checker. Additionally, for obtaining visual signal strength for null residual plots and bootstrapped residual plots, there are two specialized methods, `null_vss()` and `boot_vss()`, designed for this purpose respectively.

Calling the `vss()` method without arguments will predict the visual signal strength for the true residual plot and return the result as a single-element `tibble`.

```
checker$vss()
```

```
# A tibble: 1 x 1
    vss
  <dbl>
1  3.16
```

Providing a `data.frame` of null residuals or a null residual plot yields the same visual signal strength.

```
null_resid <- checker$null_method()
checker$vss(null_resid)
```

```
# A tibble: 1 x 1
    vss
  <dbl>
1  1.02
```

```
null_resid |>
  checker$plot_resid() |>
  checker$vss()
```

```
# A tibble: 1 x 1
    vss
  <dbl>
1  1.02
```

The `null_vss()` helper method primarily takes the number of null plots as input. If the user wants to use a ad hoc null simulation scheme, it can be provided via the `null_method` argument. Intermediate results, including null residuals and null plots, can be returned by enabling `keep_null_data` and `keep_null_plot`. The visual signal strength, along with null residuals and null plots, will be stored in a `tibble` with three columns. The following code example demonstrates how to predict the visual signal strength for five null residual plots while keeping the intermediate results.

```
checker$null_vss(5L,
                 keep_null_data = TRUE,
                 keep_null_plot = TRUE)
```

```
# A tibble: 5 x 3
    vss data              plot
  <dbl> <list>            <list>
1 1.35  <tibble [50 x 2]> <gg>
2 0.629 <tibble [50 x 2]> <gg>
3 1.77  <tibble [50 x 2]> <gg>
4 1.91  <tibble [50 x 2]> <gg>
5 1.71  <tibble [50 x 2]> <gg>
```

The `boot_vss()` helper method is similar to `null_vss()`, with some differences in argument names. The following code example demonstrates how to predict the visual signal strength for five bootstrapped residual plots while keeping the intermediate results.

```
checker$boot_vss(5L,
                 keep_boot_data = TRUE,
                 keep_boot_plot = TRUE)
```

```
# A tibble: 5 x 3
```

```
    vss data              plot
  <dbl> <list>            <list>
1  1.26 <tibble [50 x 2]> <gg>
2  3.35 <tibble [50 x 2]> <gg>
3  3.16 <tibble [50 x 2]> <gg>
4  2.87 <tibble [50 x 2]> <gg>
5  2.54 <tibble [50 x 2]> <gg>
```

4.2.4.11 *P*-value Computation

Once we have obtained the visual signal strength from both the true residual plot and the null plots, we can compute the *p*-value. This *p*-value represents the ratio of plots with visual signal strength greater than or equal to that of the true residual plot. We can perform this calculation using the `check()` method. The main inputs for this method are the number of null plots and the number of bootstrapped plots to generate. If you need to access intermediate residuals and plots, you can enable the `keep_data` and `keep_plot` options. The method stores the final result in the `check_result` field of the object. To obtain the p-value using the `check()` method, you can use the following code.

```
checker$check(boot_draws = 100L, null_draws = 100L)
checker$check_result$p_value
```

```
[1] 0.01980198
```

You can also check the *p*-value by printing the checker, which includes it in the summary report.

```
checker
```

```
-- <AUTO_VI object>
Status:
 - Fitted model: lm
 - Keras model: UNKNOWN
    - Output node index: 1
 - Result:
    - Observed visual signal strength: 3.162 (p-value = 0.0198)
    - Null visual signal strength: [100 draws]
       - Mean: 1.42
       - Quantiles:
```

```
   25%    50%    75%    80%    90%    95%    99%
0.9296 1.3095 1.7277 1.7810 2.2497 2.5835 3.1570
```

- Bootstrapped visual signal strength: [100 draws]
  - Mean: 2.623 (p-value = 0.05941)
  - Quantiles:

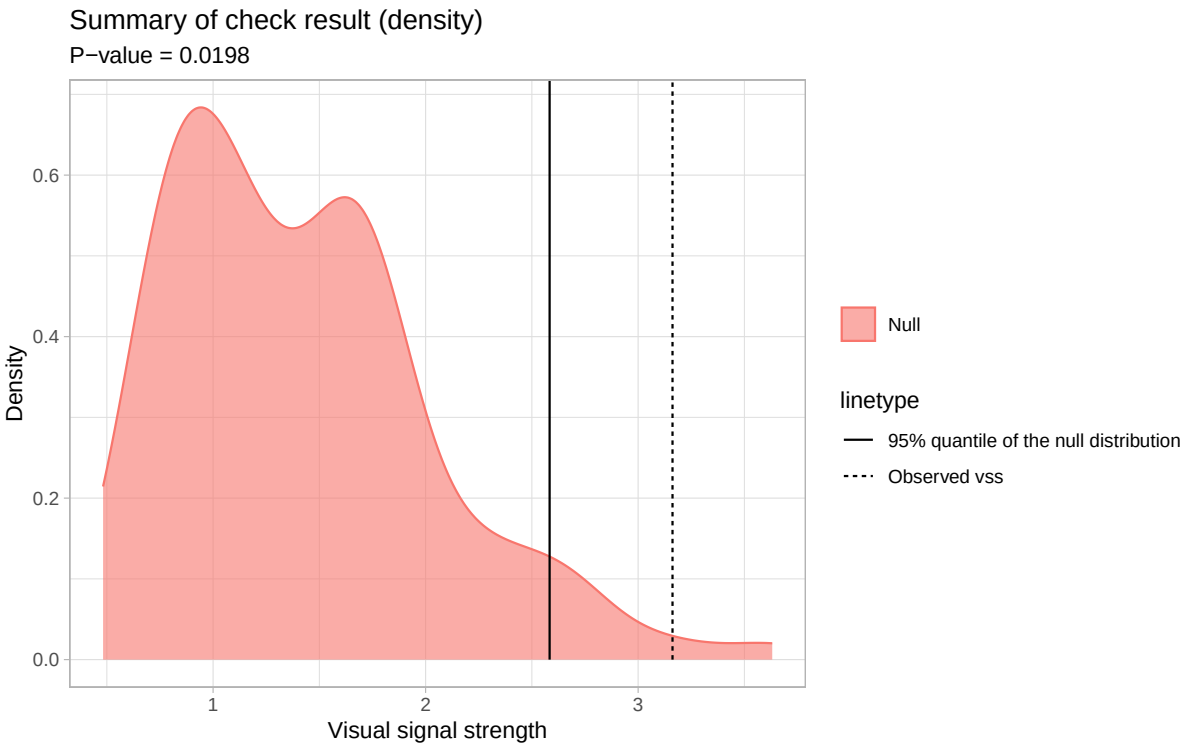
```
  25%   50%   75%   80%   90%   95%   99%
2.144 2.770 3.160 3.256 3.444 3.589 3.705
```

- Likelihood ratio: 0.5334 (boot) / 0.02943 (null) = 18.12

4.2.5 Summary Plots

After executing the `check()` method, `autovi` offers two visualization options for the assessment result through the `summary_plot()` method, including the density plot and the rank plot. We have already discussed and interpreted the density plot in an earlier section. Here, we would like to highlight the flexibility in choosing which elements to display in the density plot. For instance, you can omit the bootstrapped distribution by setting `boot_dist` to `NULL`. Similarly, you can hide the null distribution (`null_dist`), the *p*-value (`p_value`), or the likelihood ratio (`likelihood_ratio`) as needed. The following example demonstrates how to create a summary plot without the results from bootstrapped plots.

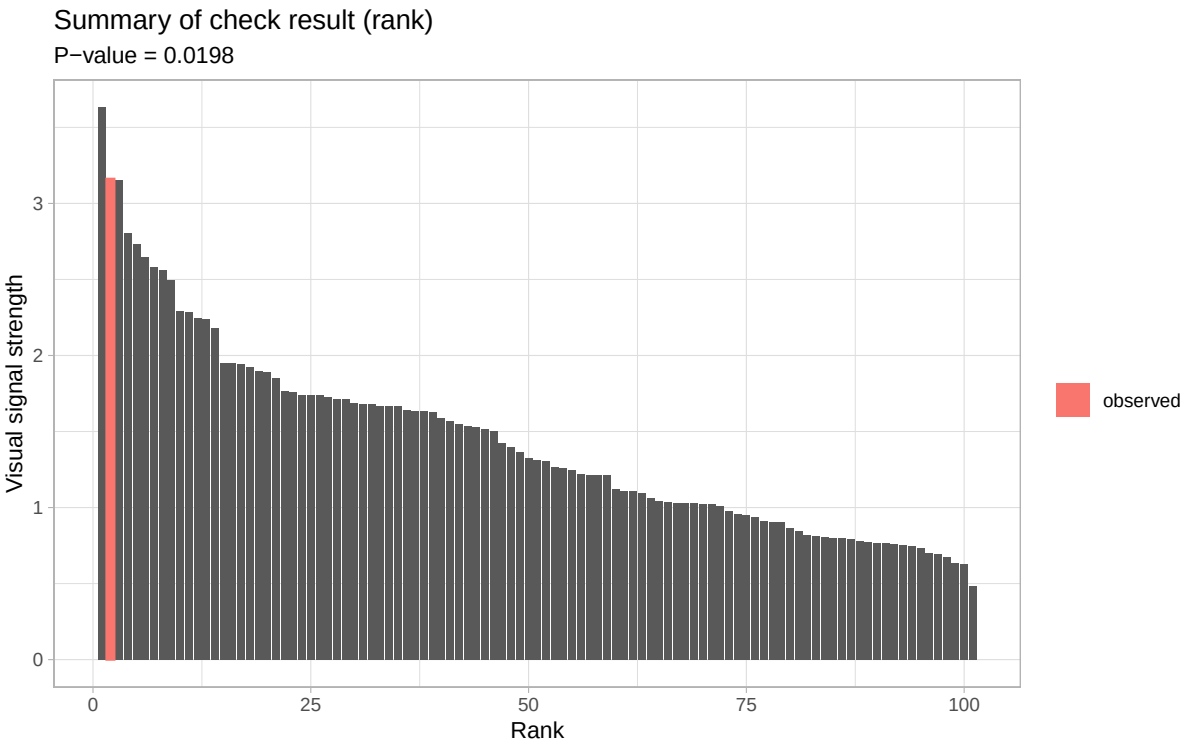
```
checker$summary_plot(boot_dist = NULL,
                     likelihood_ratio = NULL)
```



This customization allows you to focus on specific aspects of the assessment, tailoring the visualization to your analytical needs.

The rank plot, creating by setting `type` to "rank", is a bar plot where the x-axis represents the rank and the y-axis shows the visual signal strength. The bar for the true residual plot is colored in red. By examining the rank plot, you can intuitively understand how the observed visual signal strength compares to the null visual signal strengths and identify any outliers in the null distribution.

```
checker$summary_plot(type = "rank")
```



4.2.6 Feature Extraction

In addition to predicting visual signal strength and computing *p*-values, `autovi` offers methods to extract features from any layer of the Keras model. To see which layers are available in the current Keras model, you can use the `list_layer_name()` method from the `KERAS_WRAPPER` class.

The following code example lists the layer names of the currently used Keras model:

```
wrapper <- keras_wrapper(checker$keras_model)
wrapper$list_layer_name()
```

```
NULL
```

Among these layers, the "global_max_pooling2d" layer is a 2D global max pooling layer that outputs the results from the last convolutional blocks. As Simonyan and Zisserman (2014) noted, all preceding convolutional blocks can be viewed as a large feature extractor. Consequently, the output from this layer provides features that can be utilized for various purposes, such as performing transfer learning.

To obtain the features, provide the layer name using the `extract_feature_from_layer` argument in the `predict()` method. This will return a `tibble` with the visual signal strength and all features extracted from that layer. Each row corresponds to one plot. The features will be flattened into 2D and named with the prefix "f_" followed by a number from one to the total number of features.

```
checker$plot_resid() |>
  checker$save_plot() |>
  wrapper$image_to_array() |>
  wrapper$predict(auxiliary = checker$auxiliary(),
                  extract_feature_from_layer = "global_max_pooling2d")
```

```
# A tibble: 1 x 257
    vss   f_1   f_2   f_3   f_4   f_5    f_6   f_7    f_8   f_9   f_10  f_11
  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  <dbl> <dbl>  <dbl> <dbl>  <dbl> <dbl>
1  3.16 0.151     0     0     0     0 0.0203 0.109 0.0203     0 0.0834     0
# i 245 more variables: f_12 <dbl>, f_13 <dbl>, f_14 <dbl>, f_15 <dbl>,
#   f_16 <dbl>, f_17 <dbl>, f_18 <dbl>, f_19 <dbl>, f_20 <dbl>, f_21 <dbl>,
#   f_22 <dbl>, f_23 <dbl>, f_24 <dbl>, f_25 <dbl>, f_26 <dbl>, f_27 <dbl>,
#   f_28 <dbl>, f_29 <dbl>, f_30 <dbl>, f_31 <dbl>, f_32 <dbl>, f_33 <dbl>,
#   f_34 <dbl>, f_35 <dbl>, f_36 <dbl>, f_37 <dbl>, f_38 <dbl>, f_39 <dbl>,
#   f_40 <dbl>, f_41 <dbl>, f_42 <dbl>, f_43 <dbl>, f_44 <dbl>, f_45 <dbl>,
#   f_46 <dbl>, f_47 <dbl>, f_48 <dbl>, f_49 <dbl>, f_50 <dbl>, f_51 <dbl>, ...
```

Alternatively, the `AUTO_VI` class provides a way to extract features using the `vss()` method. This method is essentially a high-level wrapper around the `predict()` method of `KERAS_WRAPPER`, but it offers a more straightforward interface and better default arguments.

The results from the previous code example can be replicated with a single line of code as shown below.

```
checker$vss(extract_feature_from_layer = "global_max_pooling2d")
```

```
# A tibble: 1 x 257
    vss   f_1   f_2   f_3   f_4   f_5    f_6   f_7    f_8   f_9   f_10  f_11
  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  <dbl> <dbl>  <dbl> <dbl>  <dbl> <dbl>
1  3.16 0.151     0     0     0     0 0.0203 0.109 0.0203     0 0.0834     0
# i 245 more variables: f_12 <dbl>, f_13 <dbl>, f_14 <dbl>, f_15 <dbl>,
#   f_16 <dbl>, f_17 <dbl>, f_18 <dbl>, f_19 <dbl>, f_20 <dbl>, f_21 <dbl>,
#   f_22 <dbl>, f_23 <dbl>, f_24 <dbl>, f_25 <dbl>, f_26 <dbl>, f_27 <dbl>,
#   f_28 <dbl>, f_29 <dbl>, f_30 <dbl>, f_31 <dbl>, f_32 <dbl>, f_33 <dbl>,
#   f_34 <dbl>, f_35 <dbl>, f_36 <dbl>, f_37 <dbl>, f_38 <dbl>, f_39 <dbl>,
#   f_40 <dbl>, f_41 <dbl>, f_42 <dbl>, f_43 <dbl>, f_44 <dbl>, f_45 <dbl>,
```

```
#   f_46 <dbl>, f_47 <dbl>, f_48 <dbl>, f_49 <dbl>, f_50 <dbl>, f_51 <dbl>, ...
```

The argument `extract_feature_from_layer` is also available in other functions that build on the `vss()` method, including `null_vss()`, `boot_vss()`, and `check()`.

The package provides tools for analyzing these extracted features through the `feature_pca()` method and its associated visualization method, `feature_pca_plot()`. The `feature_pca()` method performs principal component analysis (PCA) on the features to reduce their dimensionality. However, it requires that a `check()` is performed first, as it relies on results stored in `check_result`. Alternatively, you can manually provide features using the `feature`, `null_feature`, and `boot_feature` arguments for the true residual plot, null plots, and bootstrapped plots, respectively. The `feature_pca()` method returns a tibble containing both the original features and the principal components. The rotation matrix and standard deviations of each principal component are stored as attributes.

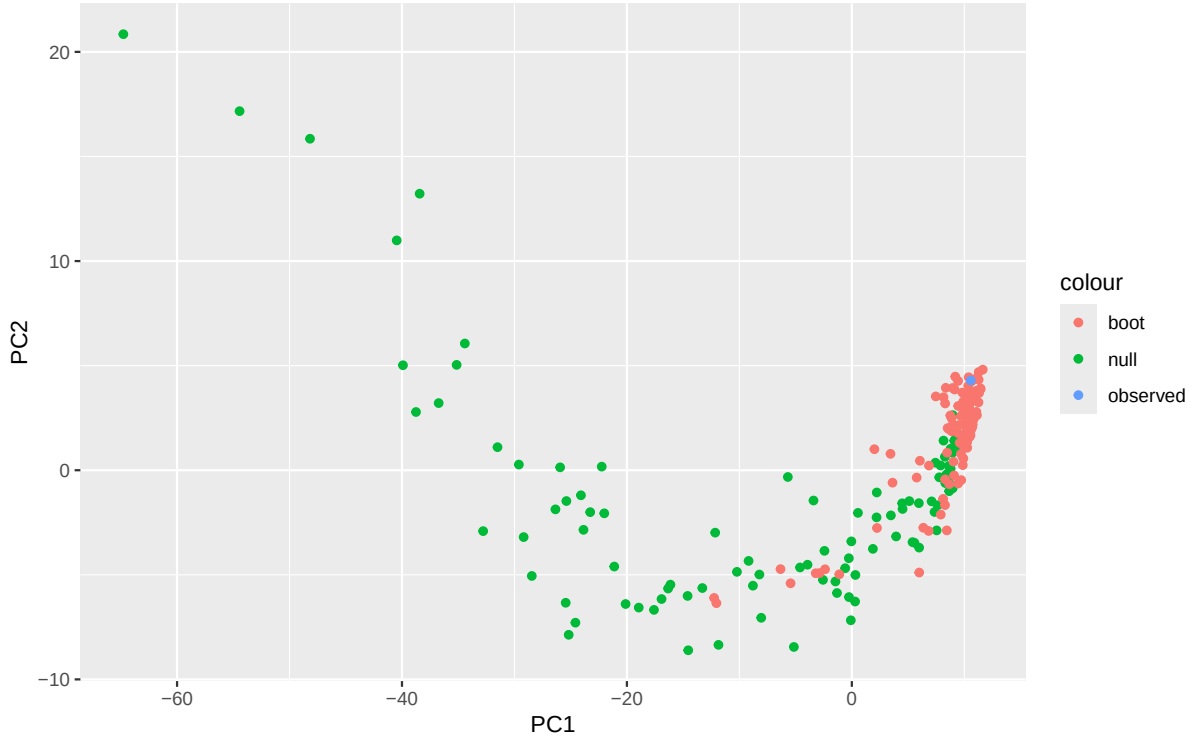
```
checker$check(null_draws = 100L,
              boot_draws = 100L,
              extract_feature_from_layer = "global_max_pooling2d")
checker$feature_pca()
```

```
# A tibble: 201 x 458
     f_1   f_2   f_3   f_4   f_5    f_6     f_7    f_8   f_9   f_10   f_11
   <dbl> <dbl> <dbl> <dbl> <dbl>  <dbl>   <dbl>  <dbl> <dbl>  <dbl>  <dbl>
 1 0.151 0     0     0     0     0.0203 0.109   0.0203 0     0.0834 0
 2 1.17  1.87  2.10  1.99  0.646 0.806  1.16    1.12   1.11  0.230  1.77
 3 0.898 1.95  1.89  1.98  0.683 0.783  1.09    1.03   1.08  0.401  1.62
 4 0.699 2.64  2.41  3.27  1.41  1.29   1.94    1.50   1.26  1.16   2.50
 5 0.494 1.22  0.836 0.867 0     0.212  0.231   0.172  0.835 0      0.589
 6 0.356 0.912 0.203 0.589 0     0      0.0225  0.142  0.485 0.0311 0.162
 7 0.514 1.25  0.817 0.900 0     0.165  0.176   0.172  0.833 0      0.589
 8 1.13  2.15  2.30  2.26  0.785 0.932  1.31    1.21   1.25  0.363  1.95
 9 0.270 0.795 0.123 0.438 0     0      0       0.0272 0.501 0      0.0608
10 0.245 0.807 0     0.357 0     0      0.00358 0.0494 0.426 0      0
# i 191 more rows
# i 447 more variables: f_12 <dbl>, f_13 <dbl>, f_14 <dbl>, f_15 <dbl>,
#   f_16 <dbl>, f_17 <dbl>, f_18 <dbl>, f_19 <dbl>, f_20 <dbl>, f_21 <dbl>,
#   f_22 <dbl>, f_23 <dbl>, f_24 <dbl>, f_25 <dbl>, f_26 <dbl>, f_27 <dbl>,
#   f_28 <dbl>, f_29 <dbl>, f_30 <dbl>, f_31 <dbl>, f_32 <dbl>, f_33 <dbl>,
```

```
#   f_34 <dbl>, f_35 <dbl>, f_36 <dbl>, f_37 <dbl>, f_38 <dbl>, f_39 <dbl>,
#   f_40 <dbl>, f_41 <dbl>, f_42 <dbl>, f_43 <dbl>, f_44 <dbl>, f_45 <dbl>, ...
```

The `feature_pca_plot()` method visualizes the results of the PCA. By default, it plots the first principal component on the x-axis and the second principal component on the y-axis, with points colored according to their origi, true residual plots, null residual plots, or bootstrapped residual plots. Users can customize the x and y axes by specifying symbols for the `x` and `y` arguments. Additionally, the `col_by_set` option can be disabled if you prefer not to use coloring.

```
checker$feature_pca_plot()
```



When interpreting the principal component scatter plot, look for any outliers within the null or bootstrapped groups. Assess whether the null group and the bootstrapped group form a single cluster or distinct clusters. Additionally, evaluate whether the observed point is distinct from the null group.

4.2.7 Trained Model Hosting

The trained computer vision models described in Chapter 3 are hosted on a GitHub repository at https://github.com/TengMCing/autovi_data. Currently, there are six models available. You can view them by calling `list_keras_model()`, which will return a `tibble` showing the input shape and a description of each model.

```
# A tibble: 6 x 9
  model_name  path              volume_path volume_size input_height input_width
```

```
  <chr>       <chr>             <chr>             <int>        <int>       <int>
1 vss_32      keras_model/vss_~ keras_mode~           4           32          32
2 vss_64      keras_model/vss_~ keras_mode~           1           64          64
3 vss_128     keras_model/vss_~ keras_mode~           8          128         128
4 vss_phn_32  keras_model/vss_~ keras_mode~           2           32          32
5 vss_phn_64  keras_model/vss_~ keras_mode~           8           64          64
6 vss_phn_128 keras_model/vss_~ keras_mode~           8          128         128
# i 3 more variables: input_channels <int>, auxiliary_input_size <int>,
#   description <chr>
```

The `get_keras_model()` function can be used to download a model to a temporary directory and load it into memory using `TensorFlow`. It requires only the model name, which is the value in the first column of the `tibble` returned by `list_keras_model()`.

4.2.8 Extending the `AUTO_VI` class

`bandicoot` is a lightweight object-oriented system with Python-like syntax that supports multiple inheritance and incorporates a Python-like method resolution order. The system is inspired by the OOP frameworks implemented in R6 [r6] and Python. In this section, we will provide essential details for extending the `autovi::AUTO_VI` class using `bandicoot`.

In `bandicoot`, a class is declared using the `bandicoot::new_class()` function, where parent classes are provided as positional arguments, and the class name is specified through the `class_name` argument. The output of `bandicoot::new_class()` is an environment with the S3 class `bandicoot_oop`. Printing a `bandicoot` object provides a summary of the object, which can be customized via the `..str..` magic method.

An extended class inherits attributes and methods from its parent class(es), so it will behave similarly to them. This can be verified using the built-in `names()` function.

```
 [1] "vss"                 "rotate_resid"        "..init.."
 [4] "plot_lineup"         "get_data"            "has_attr"
 [7] "lineup_check"        "null_vss"            "check_result"
[10] "summary_plot"        "..str.."             "..new.."
[13] "del_attr"            "plot_resid"          "null_method"
[16] "..class.."           "..method_env.."      "auxiliary"
[19] "summary"             "set_attr"            "get_attr"
[22] "summary_density_plot" "get_fitted_and_resid" "..methods.."
[25] "..class_tree.."      "..repr.."            "feature_pca"
```

```
[28] "plot_pair"           "check"              "boot_vss"
[31] "feature_pca_plot"    "boot_method"        "save_plot"
[34] "summary_rank_plot"   "instantiate"        "p_value"
[37] "..instantiated.."    "..type.."           "..dir.."
[40] "..len.."             "..bases.."          "..mro.."
[43] "likelihood_ratio"
```

To register a method for an extended class, you need to pass the class as the first argument and the method as a named argument to the `bandicoot::register_method()` function. Within a method, `self` can be used as a reference to the class or object environment. The following code example overrides the `null_method()` with a function that simulates null residuals from the corresponding normal distribution. This approach differs from the default null residual simulation scheme described in Section 4.2.4.6. Although less efficient than the default method for linear regression models, it provides an alternative way to simulate null residuals. This method is particularly useful when the fitted model is unavailable, and only the fitted values and residuals are accessible, as discussed in Section 4.3.

```
# A tibble: 50 x 2
   .fitted .resid
     <dbl>  <dbl>
 1   -1.85 -12.4
 2   -1.85 -18.8
 3    9.95  22.6
 4    9.95 -27.5
 5   13.9   28.7
 6   17.8  -13.6
 7   21.7   21.4
 8   21.7   12.5
 9   21.7  -16.2
10   25.7   -2.93
# i 40 more rows
```

To create an object in `bandicoot`, you need to call the `instantiate()` method of a class. Alternatively, you can build a convenient class constructor for your class. It is recommended to provide the full list of arguments in the class constructor instead of using `...`, as this makes it easier for integrated development environments (IDEs) like RStudio to offer argument completion hints to the

user.

4.3 Web interface: autovi.web

In Section 4.2.1, we discussed how `autovi` relies on several Python libraries, with a particularly strong dependency on `TensorFlow`. Managing a Python environment and correctly installing `TensorFlow` on a local machine can be challenging for many users. Moreover, `TensorFlow` is a massive library that undergoes continuous development, which inevitably leads to compatibility issues arising from differences in library versions. These challenges can create significant barriers for users who want to perform residual assessments with the `autovi` package.

Recognizing these potential barriers, we were motivated to design and implement a web interface called `autovi.web`. This web-based solution offers several major advantages. First, it eliminates dependency issues, so users no longer need to struggle with complex Python environments or worry about installing and maintaining the correct versions of libraries. The web interface handles all these dependencies on the server side. Second, `autovi.web` lowers the entry barrier by being user-friendly and accessible to individuals who may not be familiar with R programming. This broadens the potential user base of `autovi`, allowing more researchers and analysts to benefit from its capabilities. Third, the web interface can be updated centrally, ensuring that all users always have access to the latest features and improvements without needing to manage updates locally. Lastly, `autovi.web` offers cross-platform accessibility, allowing users to access it from any device with a web browser, increasing flexibility and convenience.

`autovi.web` is available at autoviweb.netlify.app. The implementation discussed in this chapter is based on `autovi.web` version 0.1.0. By providing this web interface, we aim to significantly reduce the technical hurdles associated with using `autovi`, making advanced residual assessment techniques more accessible to a wider audience of researchers and data analysts. This approach aligns with modern trends in data science tools, where web-based interfaces are increasingly used to make advanced analytical techniques more widely available.

4.3.1 Implementation

`autovi.web` is a web application built using the `shiny` (Chang et al. 2022) and `shinydashboard` (Chang and Borges Ribeiro 2021) R packages. Hosted on the shinyapps.io domain, the application is accessible through any modern web browser.Additionally, R packages `htmltools` (Cheng et al. 2024) and `shinycssloaders` (Sali and Attali 2020) are used to render markdown document in shiny application, and add loading animation for shiny widgets, respectively.

In our initial planning for `autovi.web`, we considered implementing the entire web application using

the `webr` framework (Moon 2020), which would have allowed the entire application to run directly in the user's browser. However, this approach was not feasible at the time of writing this chapter. The reason is that one of the R packages `autovi` depends on the R package `splancs` (Rowlingson and Diggle 2023), which uses compiled Fortran code. Unfortunately, a working Emscripten version of this package, which would be required for `webr`, was not available.

We also explored the possibility of implementing the web interface using frameworks built on other languages, such as Python. However, server hosting domains that natively support Python servers typically do not have the latest version of R installed. Additionally, calling R from Python is typically done using the `rpy2` Python library, but this approach can be awkward when dealing with language syntax related to non-standard evaluation, making it challenging to develop our application in this manner. Another option we considered was renting a server where we could have full control, such as those provided by cloud platforms like Google Cloud Platform (GCP) or Amazon Web Services (AWS). However, correctly setting up the server and ensuring a secure deployment requires significant expertise, which we did not possess at the time. Ultimately, we decided that the most practical solution was to use the `shiny` and `shinydashboard` frameworks, which are well-established in the R community and offer a solid foundation for web application development.

The server-side configuration of `autovi.web` is carefully designed to support its functionality. Most required Python libraries, including `pillow` and `NumPy`, are pre-installed on the server. These libraries are integrated into the Shiny application using the `reticulate` package, which provides an interface between R and Python.

Due to the resource allocation policy of shinyapps.io, the server enters a sleep mode during periods of inactivity, resulting in the clearing of the local Python virtual environment. Consequently, when the application "wakes up" for a new user session, these libraries need to be reinstalled. While this ensures a clean environment for each session, it may lead to slightly longer loading times for the first user after a period of inactivity.

In contrast to `autovi`, `autovi.web` does not use the native Python version of `TensorFlow`. Instead, it leverages `TensorFlow.js`, a JavaScript library that allows the execution of machine learning models directly in the browser. This choice enables native browser execution, enhancing compatibility across different user environments, and shifts the computational load from the server to the client-side. `TensorFlow.js` also offers better scalability and performance, especially when dealing with resource-intensive computer vision models on shinyapps.io.

While `autovi` requires downloading pre-trained computer vision models from GitHub, these models in ".keras" file format are incompatible with `TensorFlow.js`. Therefore, we extract and store the

model weights in JSON files and include them as extra resources in the Shiny application. When the application initializes, `TensorFlow.js` rebuilds the computer vision model using these pre-stored weights.

To allow communication between `TensorFlow.js` and other components of the Shiny application, the `shinyjs` R package is used. This package allows calling custom JavaScript code within the Shiny framework. The specialized JavaScript code for initializing `TensorFlow.js` and calling `TensorFlow.js` for visual signal strength prediction is deployed alongside the Shiny application as additional resources.

4.3.2 Design

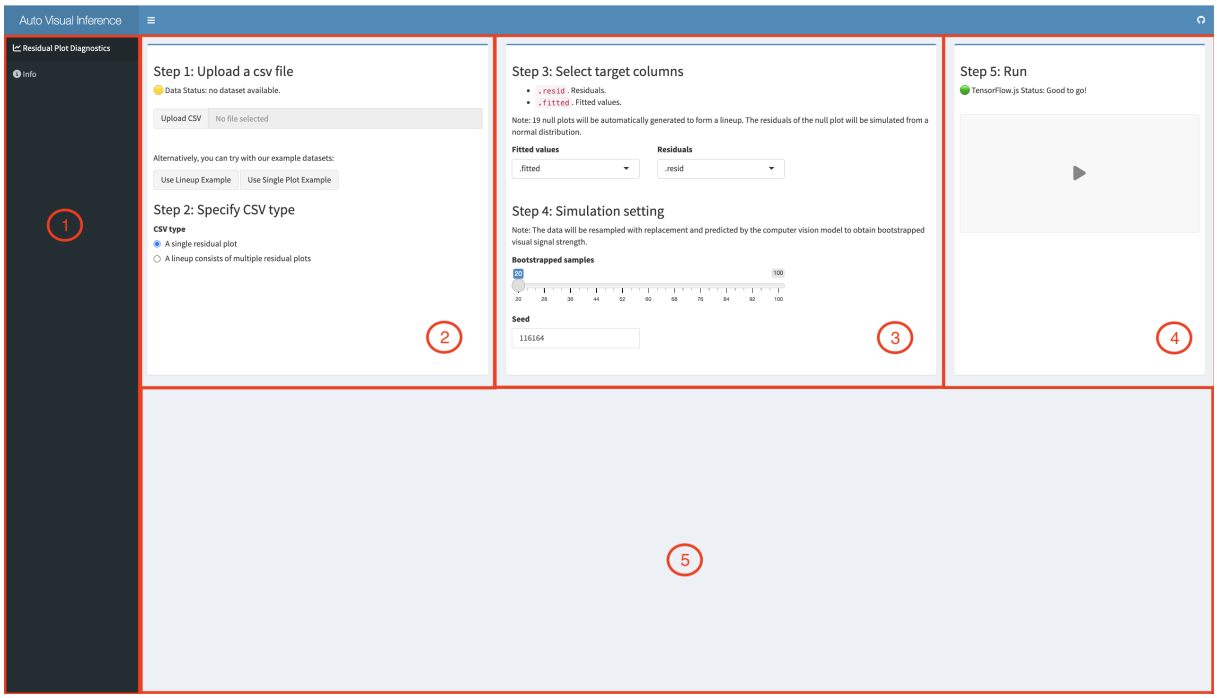


Figure 4.2: *Overview of the `autovi.web` graphical user interface (GUI). This default view may change based on user interactions. Region 1 is the sidebar menu, containing the residual assessment tab and the information tab. Region 2 is the data upload panel, where users can provide a CSV file and specify the type of data it contains. Region 3 includes dropdown menus for selecting the columns to be analyzed, a slider to control the number of bootstrapping samples, and a numeric input box for setting the simulation seed. Region 4 displays the initialization status and offers a button to start the analysis. Region 5 is empty in the default view but will be populated with results once the analysis is started.*

While the R package `autovi` aims to provide tools that can be extended to broader visual inference applicaitons, `autovi.web` is only focus on to for provdinign a straightforward and clean uesr interface. An overview of the grahpical user interface of `autovi.web` is provided in Figure 4.2. This is the default view of the web applicaiton, and there are five regions that user can mainly interact with. Region 1 of Figure 4.2 is a sidebar menu which can switch between the anlysis page and the information page. The analysis page is the focus of this section.

Region 2 of Figure 4.2 is a panel for data uploading and CSV type selection. Clicking the "upload CSV" button opens a window where the user can select a file from their local system. The data status displayed above the button provides information about the number of rows and columns in the current dataset. Additionally, there are two example datasets available beneath the "upload CSV" button: one is a lineup example using a CSV file with three columns, and the other is a single plot example using a CSV file with two columns. More details about these example datasets are be discussed in Section 4.3.4.

While the `autovi` package typically expects a fitted regression model object provided by the user, this approach is impractical for a web interface. Saving the R model object to the filesystem involves extra steps and requires users to have specific knowledge, which does not align with the goal of the web application. Moreover, the regression model object may contain sensitive, non-shareable data, making it unsuitable for uploading. Additionally, model objects are often unnecessarily large, containing extra information not needed for residual diagnostics. In contrast, a CSV file is easier to generate using various software programs, not just R. CSV files are widely accepted and can be easily viewed and modified using common desktop applications like Excel. They are generally less sensitive than raw data, as they exclude most information about the predictors.

The web application is designed to assess either a single residual plot or a lineup of residual plots. Therefore, it accepts only two types of CSV files: one with at least two columns representing the fitted values and residuals of a single residual plot, and another with at least three columns, where the additional column serves as the label or identifier for a lineup of multiple residual plots. For a single residual plot, 19 null plots are generated by simulating normal random draws from a distribution with the same variance as the original residual plot, and comparisons are made with the original residual plot. For a lineup, comparisons are made among the plots within the lineup. After uploading the CSV file, the user must select the correct format to ensure the web interface interprets the data correctly.

Region 3 of Figure 4.2 is a panel for column selection and simulation settings. As shown in Figure 4.2, if the CSV type is set to a single residual plot, there will be two dropdown menus for specifying the columns for fitted values and residuals, respectively. The default variable names for these columns are `.fitted` and `.resid`. After uploading the CSV file, the content of these dropdown menus will be updated to reflect the existing columns in the dataset. As displayed in Figure 4.3, for the CSV type that is a lineup of multiple residual plots, an additional dropdown menu will appear for specifying the column of residual plot labels. The default variable name for this column is `.sample`. If this variable name does not exist in the dataset, the dropdown menu will remain empty, allowing the user

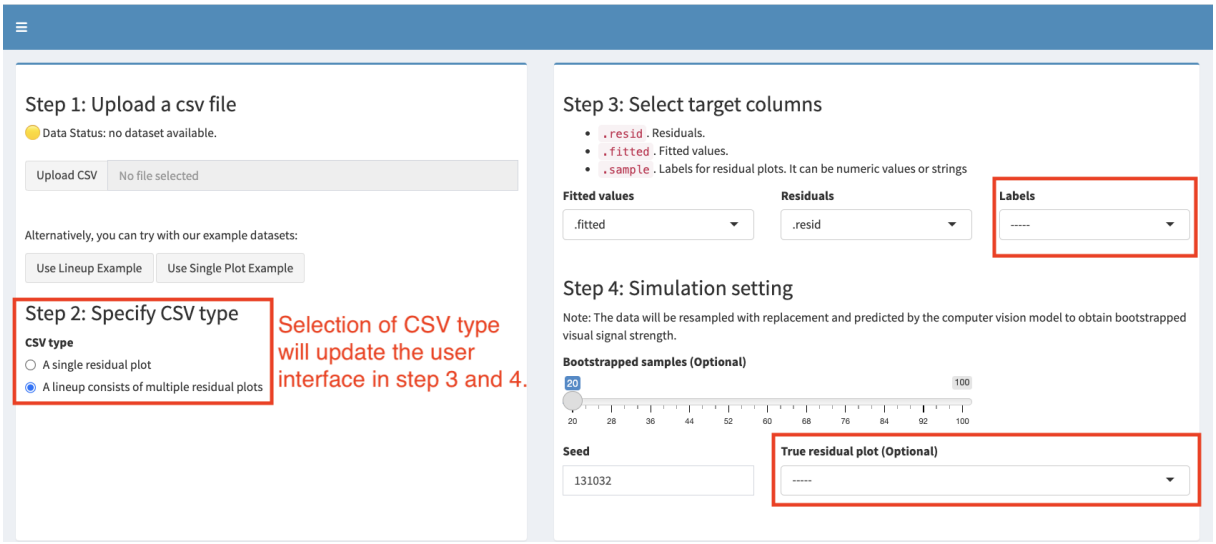


Figure 4.3: *The panels for selecting target columns and simulation settings are updated when a different CSV type is selected in the left panel. Compared to Figure 4.2, where the CSV type is a single residual plot, choosing a CSV type that includes a lineup of multiple residual plots adds a dropdown menu for specifying a column for the residual plot identifier. Additionally, an optional dropdown menu for specifying the true residual plot identifier will appear under the simulation settings.*

to specify the correct column. The number of levels for each option in this dropdown menu will be displayed to help avoid the selection of a variable with too many levels, which could significantly slow down the application due to extensive computation.

Under the simulation settings, there is a slider for specifying the number of bootstrapped samples needed for the assessment. A higher value on this slider will result in a more accurate bootstrap distribution estimation, though it will require more computation time. The simulation seed can be set in a numeric input box below the slider to control the reproducibility of the assessment. By default, a random seed is set each time the web page is refreshed. When the CSV type is a lineup of multiple residual plots, an optional dropdown menu will appear next to the simulation seed input box, allowing the user to specify an identifier for the true residual plot. If no label is provided for the true residual plot, the assessment will only estimate the visual signal strength for each residual plot in the lineup, without providing a p-value, as it cannot be computed. Consequently, some result panels may be missing due to insufficient information. This option is useful when the lineup consists solely of null plots or if the user simply wants to obtain the visual signal strength for multiple residual plots.

Region 4 of Figure 4.2 is the panel for triggering the assessment. It contains a large play button to start the assessment. Above the play button, a text message displays the status of `TensorFlow.js`, allowing users to monitor whether the JavaScript library and Keras model have been loaded correctly. The play button will remain disabled until both the data status in Region 1 and the `TensorFlow.js`

status in Region 4 indicate that everything is ready, with both showing a green status.

Once the play button is clicked, region 5 of Figure 4.2 will be populated with panels displaying the assessment results. Generally, there will be four result panels, as shown in Figure 4.4 and Figure 4.5.

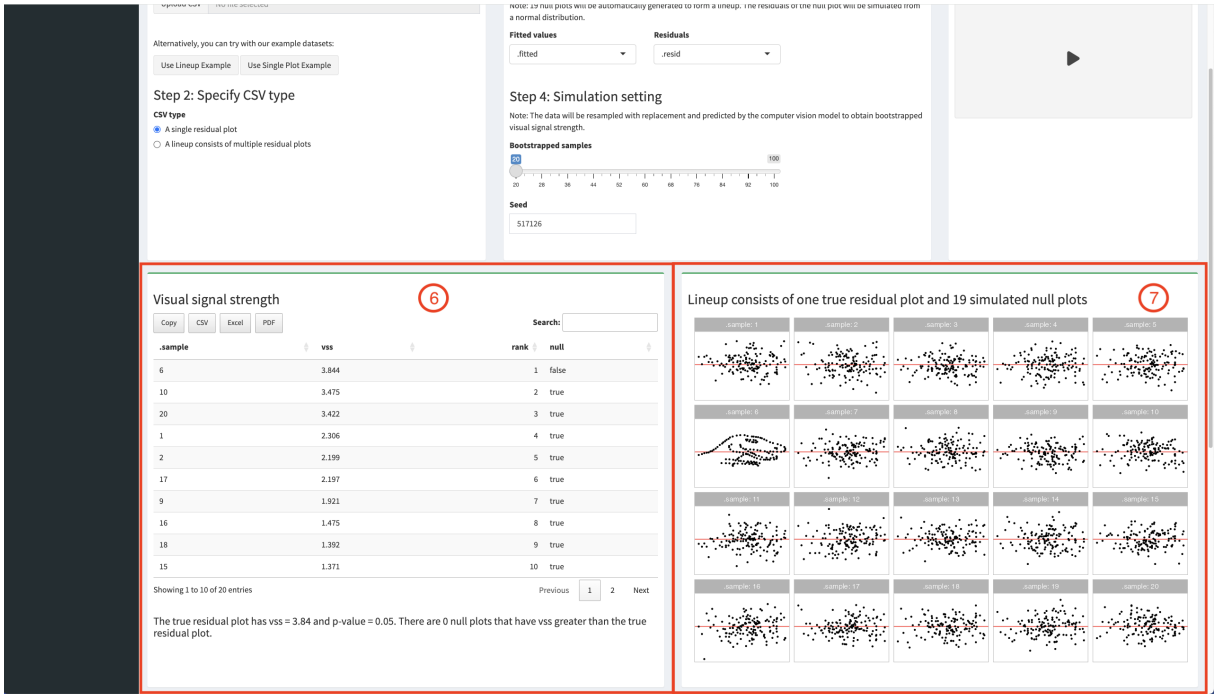


Figure 4.4: *The first two panels of results from the automated residual assessment are shown. The application provides four results panels in total, and these screenshots display the first two. In region 1, there is an interactive table detailing the visual signal strength, with a summary of the analysis provided in the paragraph below. Region 2 displays a lineup of residual plots.*

Region 6 of Figure 4.4 contains an interactive table created with the R package `DT` (Xie et al. 2024), which provides the visual signal strength. This table includes four columns: `.sample`, `vss`, `rank`, and `null`. The `.sample` column shows the residual plot labels. For a CSV type that is a lineup, these labels are taken from an identifier column in the dataset specified by the user. In the case of the CSV type is a single residual plot, labels are automatically generated from 1 to 20, with the true residual plot receiving a randomly assigned label. The `vss` column displays the visual signal strength for each residual plot, rounded to three decimal places. The `rank` column indicates the ranking of each residual plot based on visual signal strength. The `null` column reveals whether the plot is a null plot. For the CSV type that is a single residual plot, only the true residual plot will have "false" in this column, while all other plots will be marked "true." For the CSV type that is a lineup, if the true residual plot identifier has not been provided, this column will show "NA" to represent missing values. If the identifier is provided by user, the column behaves as if the CSV type is a single residual plot.

The `DT` table provides several interactive features. Users can download the table in four formats, including text, CSV, Excel, and PDF, using the buttons located above the table. Additionally, the table

Figure 4.5: *The last two panels of results from the automated residual assessment are shown. The application provides four results panels in total, and these screenshots display the final two. Region 1 presents a density plot comparing the bootstrapped visual signal strength with the null visual signal strength. Region 2 includes an attention map of the true residual plot.*

is searchable via the text input field also positioned above it. Below the table, a text message displays the *p*-value of the assessment for the true residual plot and summarizes the number of null plots with visual signal strength greater than that of the true residual plot. This helps the user determine whether the true residual plot shows visual patterns that suggest model violations.

Region 7 of Figure 4.4 provides a lineup of plots corresponding to each `.sample` value from the table in Region 6. Due to space limitations, a maximum of 20 residual plots will be displayed, ensuring that the true residual plot, if known, will be included in the lineup. The plots are generated using `ggplot2`, the same as in `autovi`. Users can perform a visual test with this lineup to check if the true residual plot is distinguishable from the other plots, helping to determine the significance of model violations.

Region 8 of Figure 4.5 displays the density plot for bootstrapped visual signal strength and null visual signal strength. The densities are shown in distinct colors that are friendly for colorblind users. A solid vertical line marks the visual signal strength of the true residual plot, while rug lines at the bottom of the plot provide a clearer view of individual cases. Below the plot, a text message indicates the number and percentage of bootstrapped residual plots that would be rejected by the visual test when compared to the null plots. Note that the bootstrapped residual plots in this application are generated differently from `autovi`. Since we do not have the R model object, we can not refit the

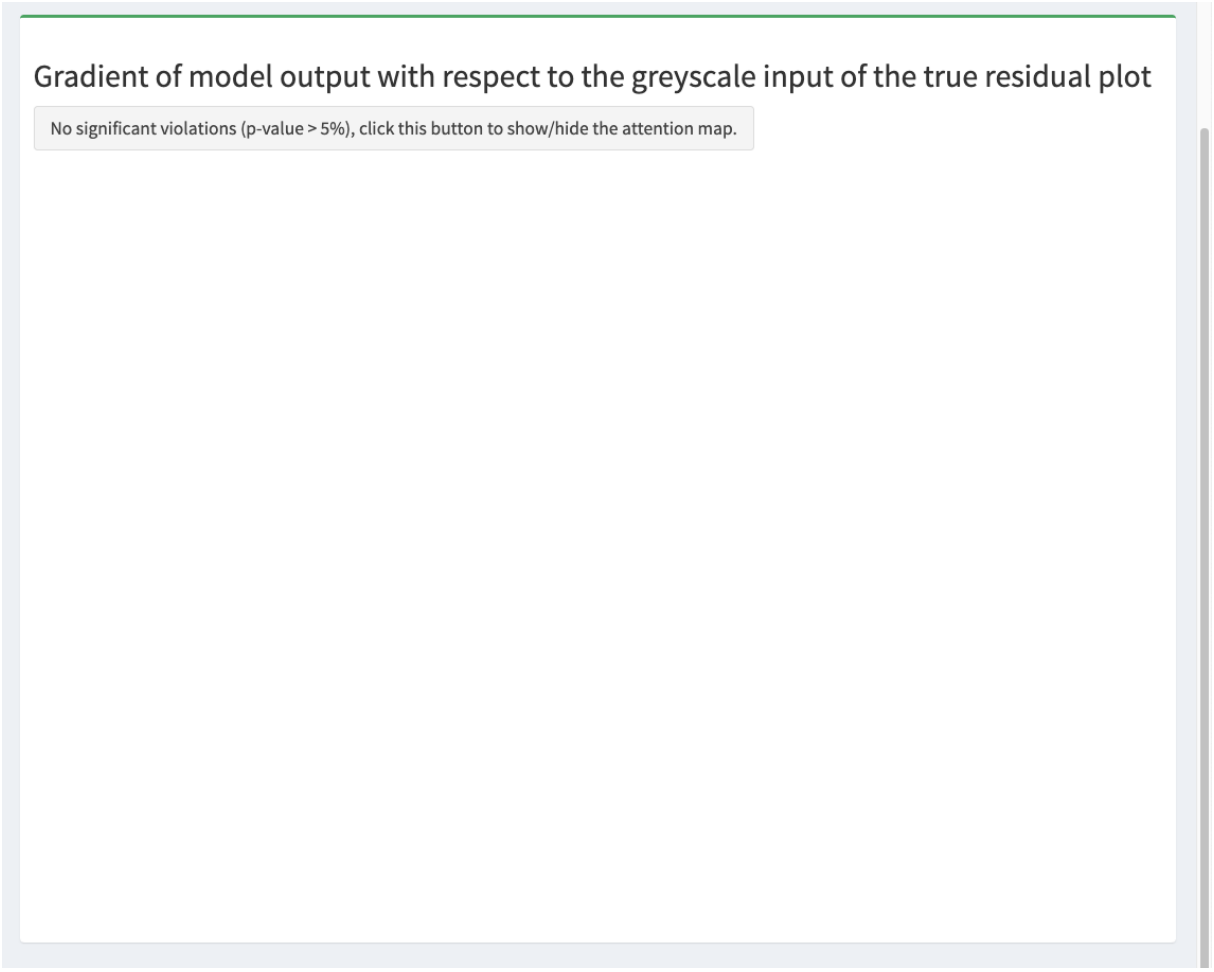


Figure 4.6: *The attention map is hidden if the assessment indicates a p-value greater than 0.05. A button is available to toggle the display of the attention map.*

regression model with bootstrapped data. Instead, we bootstrap the residuals of the true residual plot directly to obtain bootstrapped residual plots. As as result, this panel will disappear when the true residual plot is unknonw.

Region 9 of Figure 4.5 shows an attention map of the true residual plot, by computing the graident of Keras model output with repspect to the greyscale input of the true residual plot. We choose to show the attention map because it is useful for understanding how the Keras model predict the visual signal strength, and which area it is particurally focusing on. The greyscale input is used because it is relatively difficult to show a clear attention map in RGB channels and it usually will not provide more information than the greyscale since most of the important infromation of the plot are drawn in black.

Region 9 of Figure 4.5 displays an attention map for the true residual plot, generated by computing the gradient of the Keras model's output with respect to the greyscale input of the plot. The attention map helps to understand how the Keras model predicts visual signal strength and which areas it is focusing on. We use a greyscale input because it is easier to generate a clear attention map in this

format, and it usually conveys all the essential information, as most of the important details of the plot are drawn in black. If the p-value of the true residual plot is greater than 0.05, checking the attention map is not necessary. However, to provide users with the option to review it if they wish, a button will be available, as shown in Figure 4.6. This button allows users to toggle the display of the attention map.

4.3.3 Workflow

The workflow of `autovi.web` is designed to be straightforward, with numbered steps displayed in the main user interface as shown in Figure 4.2. Since the design details have already been covered in Section 4.3.2, we will not repeat the functionality of each panel in this section.

For general users, the workflow starts with Step 1: uploading a CSV file by clicking the "upload CSV" button. A window will pop up, allowing the user to select the CSV file. The CSV file should contain at least two columns representing fitted values and residuals of a residual plot. If a lineup needs to be evaluated, the CSV file should contain an additional column for labels of reisidual plot. In Step 2, the user specifies the CSV type based on the uploaded file, choosing between a single residual plot or a lineup of multiple residual plots. Step 3 involves selecting the appropriate columns for fitted values, residuals, and, optionally, labels of residual plots, from the dataset. In Step 4, the user sets the number of bootstrapped draws needed for the analysis; more draws provide a better estimate of the bootstrapped distribution but result in slower computation. The user should also set the seed to control the simulation. If the dataset contains a lineup and the true residual plot is known, select its label. Finally, clicking the play button will initiate the analysis, and the user can review the result panels.

4.3.4 Example

4.4 Conclusions

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., and others (2016), "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*.

Belsley, D. A., Kuh, E., and Welsch, R. E. (1980), *Regression diagnostics: Identifying influential data and sources of collinearity*, John Wiley & Sons.

Box, G. E. (1976), "Science and statistics," *Journal of the American Statistical Association*, Taylor & Francis, 71, 791–799.

Breusch, T. S., and Pagan, A. R. (1979), "A simple test for heteroscedasticity and random coefficient variation," *Econometrica: Journal of the Econometric Society*, JSTOR, 1287–1294.

Brunetti, A., Buongiorno, D., Trotta, G. F., and Bevilacqua, V. (2018), "Computer vision and deep learning techniques for pedestrian detection and tracking: A survey," *Neurocomputing*, Elsevier, 300, 17–33.

Buja, A., Cook, D., Hofmann, H., Lawrence, M., Lee, E.-K., Swayne, D. F., and Wickham, H. (2009b), "Statistical inference for exploratory data analysis and model diagnostics," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, The Royal Society Publishing, 367, 4361–4383.

Buja, A., Cook, D., Hofmann, H., Lawrence, M., Lee, E.-K., Swayne, D. F., and Wickham, H. (2009a), "Statistical inference for exploratory data analysis and model diagnostics," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 367, 4361–4383. https://doi.org/10.1098/rsta.2009.0120.

Chang, W., and Borges Ribeiro, B. (2021), *Shinydashboard: Create dashboards with 'shiny'*.

Chang, W., Cheng, J., Allaire, J., Sievert, C., Schloerke, B., Xie, Y., Allen, J., McPherson, J., Dipert, A., and Borges, B. (2022), *Shiny: Web application framework for r*.

Chen, Y., Su, S., and Yang, H. (2020), "Convolutional neural network analysis of recurrence plots for anomaly detection," *International Journal of Bifurcation and Chaos*, World Scientific, 30, 2050002.

Cheng, J., Sievert, C., Schloerke, B., Chang, W., Xie, Y., and Allen, J. (2024), *Htmltools: Tools for HTML*.

Chollet, F. (2021), *Deep learning with python*, Simon; Schuster.

Chollet, F., and others (2015), "Keras," https://keras.io.

Chopra, S., Hadsell, R., and LeCun, Y. (2005), "Learning a similarity metric discriminatively, with application to face verification," in *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, IEEE, pp. 539–546.

Chowdhury, N. R., Cook, D., Hofmann, H., and Majumder, M. (2018), "Measuring lineup difficulty by matching distance metrics with subject choices in crowd-sourced data," *Journal of Computational and Graphical Statistics*, Taylor & Francis, 27, 132–145.

Chu, H., Liao, X., Dong, P., Chen, Z., Zhao, X., and Zou, J. (2019), "An automatic classification method of well testing plot based on convolutional neural network (CNN)," *Energies*, MDPI, 12, 2846.

Clark, A., and others (2015), "Pillow (pil fork) documentation," *readthedocs*.

Cleveland, W. S., and McGill, R. (1984), "Graphical perception: Theory, experimentation, and application to the development of graphical methods," *Journal of the American Statistical Association*, Taylor & Francis, 79, 531–554.

Cook, R. D., and Weisberg, S. (1982), *Residuals and influence in regression*, New York: Chapman; Hall.

Cook, R. D., and Weisberg, S. (1999), *Applied regression including computing and graphics*, John Wiley & Sons.

Davies, R., Locke, S., and D'Agostino McGowan, L. (2022), *datasauRus: Datasets from the datasaurus dozen*.

Davison, A. C., and Hinkley, D. V. (1997), *Bootstrap methods and their application*, Cambridge university press.

De Leeuw, J. R. (2015), "jsPsych: A JavaScript library for creating behavioral experiments in a web browser," *Behavior Research Methods*, Springer, 47, 1–12.

Draper, N. R., and Smith, H. (1998), *Applied regression analysis*, John Wiley & Sons.

Efron, B., and Tibshirani, R. J. (1994), *An introduction to the bootstrap*, Chapman; Hall/CRC.

Emami, S., and Suciu, V. P. (2012), "Facial recognition using OpenCV," *Journal of Mobile, Embedded and Distributed Systems*, 4, 38–43.

Farrar, T. J. (2020), *Skedastic: Heteroskedasticity diagnostics for linear regression models*, Bellville, South Africa: University of the Western Cape.

Fieberg, J., Freeman, S., and Signer, J. (2024), "Using lineups to evaluate goodness of fit of animal movement models," *Methods in Ecology and Evolution*, Wiley Online Library.

Frisch, R., and Waugh, F. V. (1933), "Partial time regressions as compared with individual trends," *Econometrica: Journal of the Econometric Society*, JSTOR, 387–401.

Fukushima, K., and Miyake, S. (1982), "Neocognitron: A new algorithm for pattern recognition

tolerant of deformations and shifts in position," *Pattern recognition*, Elsevier, 15, 455–469.

Goode, K., and Rey, K. (2019), *ggResidpanel: Panels and interactive versions of diagnostic plots using 'ggplot2'*.

Goodfellow, I., Bengio, Y., and Courville, A. (2016), *Deep learning*, MIT press.

Goscinski, W. J., McIntosh, P., Felzmann, U., Maksimenko, A., Hall, C. J., Gureyev, T., Thompson, D., Janke, A., Galloway, G., Killeen, N. E., and others (2014), "The multi-modal australian ScienceS imaging and visualization environment (MASSIVE) high performance computing infrastructure: Applications in neuroscience and neuroinformatics research," *Frontiers in Neuroinformatics*, Frontiers Media SA, 8, 30.

Grinberg, M. (2018), *Flask web development: Developing web applications with python*, " O'Reilly Media, Inc.".

Hailesilassie, T. (2019), "Financial market prediction using recurrence plot and convolutional neural network," Preprints.

Harris, C. R., Millman, K. J., Van Der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., and others (2020), "Array programming with NumPy," *Nature*, Nature Publishing Group UK London, 585, 357–362.

Harrison Jr, D., and Rubinfeld, D. L. (1978), "Hedonic housing prices and the demand for clean air," *Journal of environmental economics and management*, Elsevier, 5, 81–102.

Hartig, F. (2022), *DHARMa: Residual diagnostics for hierarchical (multi-level / mixed) regression models*.

Hastie, T. J. (2017), "Generalized additive models," in *Statistical models in s*, Routledge, pp. 249–307.

Hatami, N., Gavet, Y., and Debayle, J. (2018), "Classification of time-series images using deep convolutional neural networks," in *Tenth international conference on machine vision (ICMV 2017)*, SPIE, pp. 242–249.

Hebbali, A. (2024), *Olsrr: Tools for building OLS regression models*.

Hermite, M. (1864), *Sur un nouveau développement en série des fonctions*, Imprimerie de Gauthier-Villars.

Hofmann, H., Wickham, H., and Kafadar, K. (2017), "Value plots: Boxplots for large data," *Journal of Computational and Graphical Statistics*, Taylor & Francis, 26, 469–477.

Hornik, K. (2012), "The comprehensive r archive network," *Wiley interdisciplinary reviews: Computational statistics*, Wiley Online Library, 4, 394–398.

Hyndman, R. J., and Fan, Y. (1996), "Sample quantiles in statistical packages," *The American Statistician*, Taylor & Francis, 50, 361–365.

Jamshidian, M., Jennrich, R. I., and Liu, W. (2007), "A study of partial f tests for multiple linear regression models," *Computational Statistics & Data Analysis*, Elsevier, 51, 6269–6284.

Jarque, C. M., and Bera, A. K. (1980), "Efficient tests for normality, homoscedasticity and serial independence of regression residuals," *Economics Letters*, Elsevier, 6, 255–259.

Johnson, P. E. (2022), *Rockchalk: Regression estimation and presentation*.

Kahle, D. (2013), "Mpoly: Multivariate polynomials in R," *The R Journal*, 5, 162–170.

Kahneman, D. (2011), *Thinking, fast and slow*, macmillan.

Kimball, A. (1957), "Errors of the third kind in statistical consulting," *Journal of the American Statistical Association*, Taylor & Francis, 52, 133–142.

Kingma, D. P., and Ba, J. (2014), "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*.

Kirk, R. E. (1996), "Practical significance: A concept whose time has come," *Educational and psychological measurement*, Sage Publications Sage CA: Thousand Oaks, CA, 56, 746–759.

Krishnan, G., and Hofmann, H. (2021), "Hierarchical decision ensembles-an inferential framework for uncertain human-AI collaboration in forensic examinations," *arXiv preprint arXiv:2111.01131*.

Kullback, S., and Leibler, R. A. (1951), "On information and sufficiency," *The Annals of Mathematical Statistics*, JSTOR, 22, 79–86.

Langsrud, Ø. (2005), "Rotation tests," *Statistics and computing*, Springer, 15, 53–60.

Laplace, P.-S. (1820), *Théorie analytique des probabilités*, Courcier.

Lee, H., and Chen, Y.-P. P. (2015), "Image based computer aided diagnosis system for cancer detection," *Expert Systems with Applications*, Elsevier, 42, 5356–5365.

Li, W. (2024), "Bandicoot: Light-weight python-like object-oriented system."

Li, W., Cook, D., Tanaka, E., and VanderPlas, S. (2024), "A plot is worth a thousand tests: Assessing residual diagnostics with the lineup protocol," *Journal of Computational and Graphical Statistics*, Taylor & Francis, 1–19.

Long, J. A. (2022), *Jtools: Analysis and presentation of social scientific data*.

Loy, A. (2021), "Bringing visual inference to the classroom," *Journal of Statistics and Data Science Education*, Taylor & Francis, 29, 171–182.

Loy, A., Follett, L., and Hofmann, H. (2016), "Variations of q–q plots: The power of our eyes!" *The American Statistician*, Taylor & Francis, 70, 202–214.

Loy, A., and Hofmann, H. (2013), "Diagnostic tools for hierarchical linear models," *Wiley Interdisciplinary Reviews: Computational Statistics*, Wiley Online Library, 5, 48–61.

Loy, A., and Hofmann, H. (2014), "HLMdiag: A suite of diagnostics for hierarchical linear models in r," *Journal of Statistical Software*, 56, 1–28.

Loy, A., and Hofmann, H. (2015), "Are you normal? The problem of confounded residual structures in hierarchical linear models," *Journal of Computational and Graphical Statistics*, Taylor & Francis,

24, 1191–1209.

Majumder, M., Hofmann, H., and Cook, D. (2013b), "Validation of visual statistical inference, applied to linear models," *Journal of the American Statistical Association*, Taylor & Francis, 108, 942–956.

Majumder, M., Hofmann, H., and Cook, D. (2013a), "Validation of visual statistical inference, applied to linear models," *Journal of the American Statistical Association*, 108, 942–956. https://doi.org/10.1080/01621459.2013.808157.

Mason, H., Lee, S., Laa, U., and Cook, D. (2022), *Cassowaryr: Compute scagnostics on pairs of numeric variables in a data set*.

Montgomery, D. C., Peck, E. A., and Vining, G. G. (1982), *Introduction to linear regression analysis*, John Wiley & Sons.

Moon, K.-W. (2020), *Webr: Data and functions for web-based analysis*.

Nair, V., and Hinton, G. E. (2010), "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814.

O'Malley, T., Bursztein, E., Long, J., Chollet, F., Jin, H., Invernizzi, L., and others (2019), "Keras Tuner," https://github.com/keras-team/keras-tuner.

Ojeda, S. A. A., Solano, G. A., and Peramo, E. C. (2020), "Multivariate time series imaging for short-term precipitation forecasting using convolutional neural networks," in *2020 international conference on artificial intelligence in information and communication (ICAIIC)*, IEEE, pp. 33–38.

Olvera Astivia, O. L., Gadermann, A., and Guhn, M. (2019), "The relationship between statistical power and predictor distribution in multilevel logistic regression: A simulation-based approach," *BMC Medical Research Methodology*, BioMed Central, 19, 1–20.

Palan, S., and Schitter, C. (2018), "Prolific. Ac—a subject pool for online experiments," *Journal of Behavioral and Experimental Finance*, Elsevier, 17, 22–27.

PythonAnywhere LLP (2023), "PythonAnywhere."

R Core Team (2022), *R: A language and environment for statistical computing*, Vienna, Austria: R Foundation for Statistical Computing.

Ramsey, J. B. (1969), "Tests for specification errors in classical linear least-squares regression analysis," *Journal of the Royal Statistical Society: Series B (Methodological)*, Wiley Online Library, 31, 350–371.

Rawat, W., and Wang, Z. (2017), "Deep convolutional neural networks for image classification: A comprehensive review," *Neural computation*, MIT Press, 29, 2352–2449.

Reinhart, A. (2024), *Regressinator: Simulate and diagnose (generalized) linear models*.

Rowlingson, B., and Diggle, P. (2023), *Splancs: Spatial and space-time point pattern analysis*.

Roy Chowdhury, N., Cook, D., Hofmann, H., Majumder, M., Lee, E.-K., and Toth, A. L. (2015), "Using

visual statistical inference to better understand random class separations in high dimension, low sample size data," *Computational Statistics*, Springer, 30, 293–316. https://doi.org/10.1007/s00180-014-0534-x.

Sali, A., and Attali, D. (2020), *Shinycssloaders: Add loading animations to a 'shiny' output while it's recalculating*.

Shapiro, S. S., and Wilk, M. B. (1965), "An analysis of variance test for normality (complete samples)," *Biometrika*, JSTOR, 52, 591–611.

Silverman, B. W. (2018), *Density estimation for statistics and data analysis*, Routledge.

Silvey, S. D. (1959), "The lagrangian multiplier test," *The Annals of Mathematical Statistics*, JSTOR, 30, 389–407.

Simonyan, K., and Zisserman, A. (2014), "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*.

Singh, K., Gupta, G., Vig, L., Shroff, G., and Agarwal, P. (2017), "Deep convolutional neural networks for pairwise causality," *arXiv preprint arXiv:1701.00597*.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014), "Dropout: A simple way to prevent neural networks from overfitting," *The journal of machine learning research*, JMLR. org, 15, 1929–1958.

Tukey, J. W., and Tukey, P. A. (1985), "Computer graphics and exploratory data analysis: An introduction," in *Proceedings of the sixth annual conference and exposition: Computer graphics*, pp. 773–785.

Ushey, K., Allaire, J., and Tang, Y. (2024), *Reticulate: Interface to 'python'*.

VanderPlas, S., and Hofmann, H. (2016), "Spatial reasoning and data displays," *IEEE Transactions on Visualization and Computer Graphics*, 22, 459–468. https://doi.org/10.1109/TVCG.2015.2469125.

VanderPlas, S., Röttger, C., Cook, D., and Hofmann, H. (2021), "Statistical significance calculations for scenarios in visual inference," *Stat*, Wiley Online Library, 10, e337.

Vo, N. N., and Hays, J. (2016), "Localizing and orienting street views using overhead imagery," in *Computer vision–ECCV 2016: 14th european conference, amsterdam, the netherlands, october 11–14, 2016, proceedings, part i 14*, Springer, pp. 494–509.

Wang, Z., Bovik, A. C., Sheikh, H. R., and Simoncelli, E. P. (2004), "Image quality assessment: From error visibility to structural similarity," *IEEE transactions on image processing*, IEEE, 13, 600–612.

White, H. (1980), "A heteroskedasticity-consistent covariance matrix estimator and a direct test for heteroskedasticity," *Econometrica: Journal of the Econometric Society*, JSTOR, 817–838.

Wickham, H. (2016), *ggplot2: Elegant graphics for data analysis*, Springer-Verlag New York.

Wickham, H., Chowdhury, N. R., Cook, D., and Hofmann, H. (2020), *Nullabor: Tools for graphical inference*.

Wickham, H., Cook, D., Hofmann, H., and Buja, A. (2010), "Graphical inference for infovis," *IEEE Transactions on Visualization and Computer Graphics*, 16, 973–979. https://doi.org/10.1109/TVCG.2010.161.

Widen, H. M., Elsner, J. B., Pau, S., and Uejio, C. K. (2016), "Graphical inference in geographical research," *Geographical Analysis*, Wiley Online Library, 48, 115–131.

Wilkinson, L., Anand, A., and Grossman, R. (2005), "Graph-theoretic scagnostics," in *Information visualization, IEEE symposium on*, IEEE Computer Society, pp. 21–21.

Xie, Y., Cheng, J., and Tan, X. (2024), *DT: A wrapper of the JavaScript library 'DataTables'*.

Yin, T., Majumder, M., Roy Chowdhury, N., Cook, D., Shoemaker, R., and Graham, M. (2013), "Visual mining methods for RNA-seq data: Data structure, dispersion estimation and significance testing," *Journal of Data Mining in Genomics and Proteomics*, 4. https://doi.org/10.4172/2153-0602.1000139.

Zhang, Y., Hou, Y., Zhou, S., and Ouyang, K. (2020), "Encoding time series as multi-scale signed recurrence plots for classification using fully convolutional networks," *Sensors*, MDPI, 20, 3818.

Zhang, Z., and Yuan, K.-H. (2018), *Practical statistical power analysis using webpower and r*, Isdsa Press.

Appendix A

Appendix to "A Plot is Worth a Thousand Tests: Assessing Residual Diagnostics with the Lineup Protocol"

A.1 Additional Details of Testing Procedures

A.1.1 Statistical Significance

Within the context of visual inference, with K independent observers, the visual p-value can be seen as the probability of having as many or more participants detect the data plot than the observed result.

The approach used in Majumder et al. (2013b) is as follows. Define $X_j = \{0, 1\}$ to be a Bernoulli random variable measuring whether participant j detected the data plot, and $X = \sum_{j=1}^{K} X_j$ be the total number of observers who detected the data plot. Then, by imposing a relatively strong assumption that all K evaluations are fully independent, under H_0 $X \sim \text{Binom}_{K,1/m}$. Therefore, the p-value of a lineup of size m evaluated by K observer is estimated with $P(X \geq x) = 1 - F(x) + f(x)$, where $F(.)$ is the binomial cumulative distribution function, $f(.)$ is the binomial probability mass function and x is the realization of number of observers choosing the data plot.

As pointed out by VanderPlas et al. (2021), this basic binomial model is deficient. It does not take into account the possible dependencies in the visual test due to repeated evaluations of the same lineup, or account for when participants are offered the option to select one or more "most different" plots, or none, from a lineup. They suggest three common lineup scenarios: (1) K different lineups are shown to K participants, (2) K lineups with different null plots but the same data plot are shown

to K participants, and (3) the same lineup is shown to K participants. Scenario 3 is the most feasible to apply, but has the most dependencies to accommodate for the p-value calculation. For Scenario 3, VanderPlas et al propose modelling the probability of plot i being selected from a lineup as θ_i, where $\theta_i \sim \text{Dirichlet}(\alpha)$ for $i = 1, ..., m$ and $\alpha > 0$. The number of times plot i being selected in K evaluations is denoted as c_i. In case participant j makes multiple selections, $1/s_j$ will be added to c_i instead of one, where s_j is the number of plots participant j selected for $j = 1, ...K$. This ensures $\sum_i c_i = K$. Since we are only interested in the selections of the data plot i, the marginal model can be simplified to a beta-binomial model and thus the visual p-value is given as

$$P(C \geq c_i) = \sum_{x=c_i}^{K} \binom{K}{x} \frac{B(x+\alpha, K-x+(m-1)\alpha)}{B(\alpha,(m-1)\alpha)}, \quad \text{for} \quad c_i \in \mathbb{Z}_0^+ \tag{A.1}$$

where $B(.)$ is the beta function defined as

$$B(a,b) = \int_0^1 t^{\alpha-1}(1-t)^{b-1}dt, \quad \text{where} \quad a,b > 0. \tag{A.2}$$

We extend the equation to non-negative real number c_i by applying a linear approximation

$$P(C \geq c_i) = P(C \geq \lceil c_i \rceil) + (\lceil c_i \rceil - c_i)P(C = \lfloor c_i \rfloor), \quad \text{for} \quad c_i \in \mathbb{R}_0^+, \tag{A.3}$$

where $P(C \geq \lceil c_i \rceil)$ is calculated using Equation A.1 and $P(C = \lfloor c_i \rfloor)$ is calculated by

$$P(C = c_i) = \binom{K}{c_i} \frac{B(c_i+\alpha, K-c_i+(m-1)\alpha)}{B(\alpha,(m-1)\alpha)}, \quad \text{for} \quad c_i \in \mathbb{Z}_0^+. \tag{A.4}$$

The parameter α used in Equation A.1 and Equation A.4 is usually unknown and will need to be estimated from the survey data. An interpretation of α is that when it is low only a few plots are attractive to the observers and tend to be selected, and when high, most plots are equally likely to be chosen. VanderPlas et al define c-interesting plot to be if c or more participants select the plot as the most different. The expected number of plots selected at least c times, $\text{E}[Z_c]$, is then calculated as

$$\text{E}[Z_c(\alpha)] = \frac{m}{B(\alpha,(m-1)\alpha)} \sum_{\lceil c \rceil}^{K} \binom{K}{x} B(x+\alpha, K-x+(m-1)\alpha). \tag{A.5}$$

With Equation A.5, α can be estimated using maximum likelihood estimation. Precise estimation of α, is aided by evaluation of Rorschach lineups, where all plots are null plots. In a Rorschach, in

theory all plots should be equally likely, but in practice some (irrelevant) visual elements may be more eye-catching than others. This is what α captures, the capacity for extraneous features to distract the observer for a particular type of plot display.

A.1.2 Effect Size Derivation

Effect size can be defined as the difference of a parameter for a particular model or distribution, or a statistic derived from a sample. Importantly, it needs to reflect the treatment we try to measure. Centred on a conventional statistical test, we usually can deduce the effect size from the test statistic by substituting the null parameter value. When considering the diagnostics of residual departures, there exist many possibilities of test statistics for a variety of model assumptions. Meanwhile, diagnostic plots such as the residual plot have no general agreement on measuring how strong a model violation pattern is. To build a bridge between various residual-based tests, and the visual test, we focus on the shared information embedded in the testing procedures, which is the distribution of residuals. When comes to comparison of distribution, Kullback-Leibler divergence (Kullback and Leibler 1951) is a classical way to represent the information loss or entropy increase caused by the approximation to the true distribution, which in our case, the inefficiency due to the use of false model assumptions.

Following the terminology introduced by Kullback and Leibler (1951), P represents the measured probability distribution, and Q represents the assumed probability distribution. The Kullback-Leibler divergence is defined as $\int_{-\infty}^{\infty} \log(p(x)/q(x))p(x)dx$, where $p(.)$ and $q(.)$ denote probability densities of P and Q.

Let $\boldsymbol{X}$ denotes the $p+1$ predictors with n observations, $\boldsymbol{b} = (\boldsymbol{X}'\boldsymbol{X})^{-1}\boldsymbol{X}'\boldsymbol{y}$ denotes the OLS solution, $\boldsymbol{R} = \boldsymbol{I}_n - \boldsymbol{X}(\boldsymbol{X}'\boldsymbol{X})^{-1}\boldsymbol{X}'$ denotes the residual operator, and let $\boldsymbol{\varepsilon} \sim N(\boldsymbol{0}, \sigma^2\boldsymbol{I})$ denotes the error. The residual vector

$$
\begin{aligned}
\boldsymbol{e} &= \boldsymbol{y} - \boldsymbol{X}\boldsymbol{b} \\
&= \boldsymbol{y} - \boldsymbol{X}(\boldsymbol{X}'\boldsymbol{X})^{-1}\boldsymbol{X}'\boldsymbol{y} \\
&= (\boldsymbol{I} - \boldsymbol{X}(\boldsymbol{X}'\boldsymbol{X})^{-1}\boldsymbol{X}')\boldsymbol{y} \\
&= \boldsymbol{R}\boldsymbol{y} \\
&= \boldsymbol{R}(\boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}) \\
&= \boldsymbol{R}\boldsymbol{\varepsilon}.
\end{aligned}
$$

Because $\text{rank}(\boldsymbol{R}) = n - p - 1 < n$, $\boldsymbol{e}$ follows a degenerate multivariate normal distribution and does

not have a density. Since the Kullback-Leibler divergence requires a proper density function, we need to simplify the covariance matrix of $\boldsymbol{e}$ by setting all the off-diagonal elements to 0. Then, the residuals will be assumed to follow $N(\mathbf{0}, diag(\boldsymbol{R}\sigma^2))$ under the null hypothesis that the model is correctly specified. If the model is however misspecified due to omitted variables $\boldsymbol{Z}$, or a non-constant variance $\boldsymbol{V}$, the distribution of residuals can be derived as $N(\boldsymbol{RZ\beta}_z, \text{diag}(\boldsymbol{R}\sigma^2))$ and $N(\mathbf{0}, \text{diag}(\boldsymbol{RVR}'))$ respectively.

By assuming both P and Q are multivariate normal density functions, the Kullback-Leibler divergence can be rewritten as

$$KL = \frac{1}{2}\left(\log\frac{|\Sigma_p|}{|\Sigma_q|} - n + \text{tr}(\Sigma_p^{-1}\Sigma_q) + (\mu_p - \mu_q)'\Sigma_p^{-1}(\mu_p - \mu_q)\right).$$

Then, we can combine the two residual departures into one formula

$$KL = \frac{1}{2}\left(\log\frac{|\text{diag}(\boldsymbol{RVR}')|}{|\text{diag}(\boldsymbol{R}\sigma^2)|} - n + \text{tr}(\text{diag}(\boldsymbol{RVR}')^{-1}\text{diag}(\boldsymbol{R}\sigma^2)) + \boldsymbol{\mu}_z'(\boldsymbol{RVR}')^{-1}\boldsymbol{\mu}_z\right).$$

When there are omitted variables but constant error variance, the formula can be reduced to

$$KL = \frac{1}{2}\left(\boldsymbol{\mu}_z'(\text{diag}(\boldsymbol{R}\sigma^2))^{-1}\boldsymbol{\mu}_z\right).$$

And when the model equation is correctly specified but the error variance is non-constant, the formula can be reduced to

$$KL = \frac{1}{2}\left(log\frac{|\text{diag}(\boldsymbol{RVR}')|}{|\text{diag}(\boldsymbol{R}\sigma^2)|} - n + \text{tr}(\text{diag}(\boldsymbol{RVR}')^{-1}\text{diag}(\boldsymbol{R}\sigma^2))\right).$$

To compute the effect size for each lineup we simulate a sufficiently large number of samples from the same model with the number of observations n fixed for each sample. We then compute the effect size for each sample and take the average as the final value. This ensures lineups constructed with the same experimental factors will share the same effect size.

A.1.3 Sensitivity Analysis for α

The parameter α used for the p-value calculation needs to be estimated from responses to null lineups. With a greater value of $\hat{\alpha}$, the p-value will be smaller, resulting in more lineups being rejected. However, The way we generate Rorschach lineup is not strictly the same as what suggested in VanderPlas et al. (2021) and Buja et al. (2009b). Therefore, we conduct a sensitivity analysis in this section to examine the impact of the variation of the estimator α on our primary findings.

Table A.1: *Examining how decisions might change if $\hat{\alpha}$ was different. Percentage of lineups that where the p-value would switch to above or below 0.05, when $\hat{\alpha}$ is multiplied by a multiplier.*

Multiplier	Reject to not reject	%	Not reject to reject	%
0.500	7	2.51	0	0.00
0.750	4	1.43	0	0.00
0.875	3	1.08	0	0.00
1.125	0	0.00	3	1.08
1.250	0	0.00	4	1.43
1.500	0	0.00	5	1.79

The analysis is conducted by setting up several scenarios, where the α is under or overestimated by 12.5%, 25% and 50%. Using the adjusted $\hat{\alpha}$, we recalculate the p-value for every lineup and show the results in Figure **??**. It can be observed that there are some changes to p-values, especially when the $\hat{\alpha}$ is multiplied by 50%. However, Table **??** shows that adjusting $\hat{\alpha}$ will not result in a huge difference in rejection decisions. There are only a small percentage of cases where the rejection decision change. It is very unlikely the downstream findings will be affected because of the estimate of α.

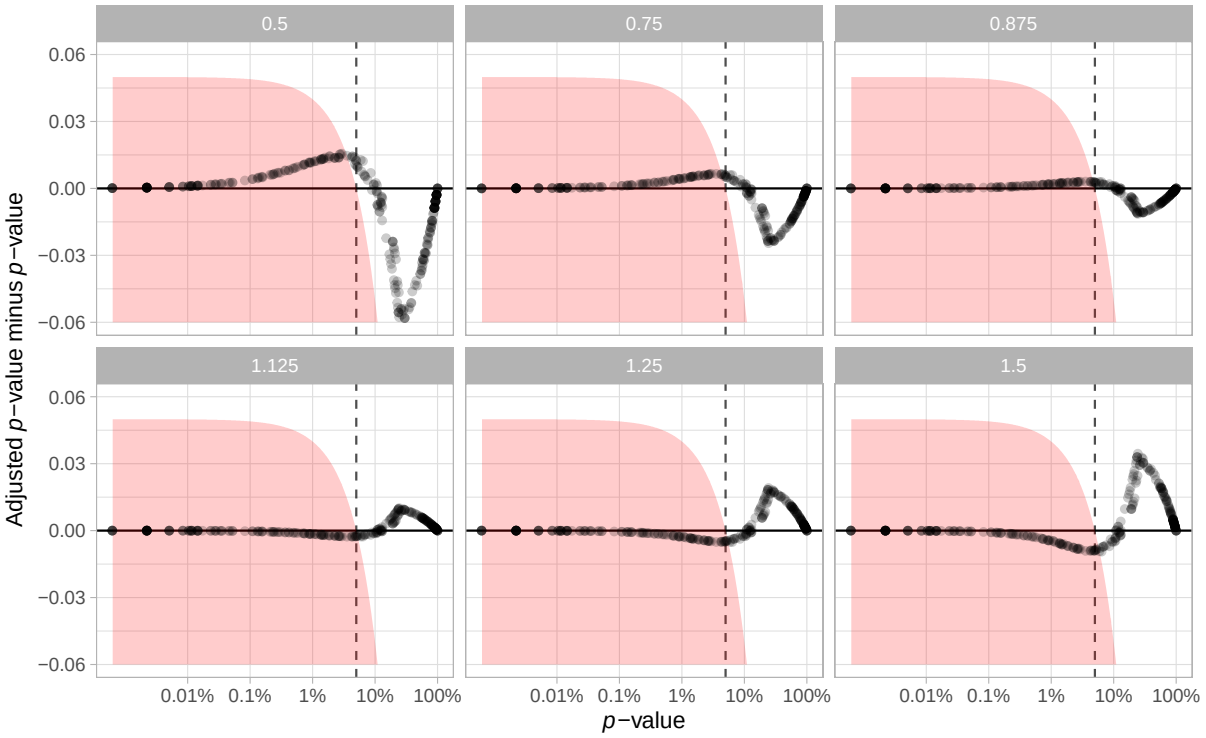


Figure A.1: *Change of p-values with $\hat{\alpha}$ multiplied by* 0.5*,* 0.75*,* 0.875*,* 1.125*,* 1.25 *and* 1.5*. Only lineups with uniform fitted value distribution is used. The vertical dashed line indicates p-value* $= 0.05$*. The area coloured in red indicates adjusted p-value* < 0.05*. The x-axis is drawn on logarithmic scale. For multipliers smaller than 1, the adjusted p-value will initially increase and decline when the p-value increases. The trend is opposite with multipliers greater than* 1*, but the difference eventually reaches* 0*.*

A.1.4 Effect of Number of Evaluations on the Power of a Visual Test

When comparing power of visual tests across different fitted value distributions, we have discussed the number of evaluations on a lineup will affect the power of the visual test. Using the lineups with uniform fitted value distribution, we show in Figure A.2 the change of power of visual tests due to different number of evaluations. It can be learned that as the number of evaluations increases, the power will increase but the margin will decrease. Considering we have eleven evaluations on lineups with uniform fitted value distribution, and five evaluations on other lineups, it is necessary to use the same number of evaluations for each lineup in comparison.

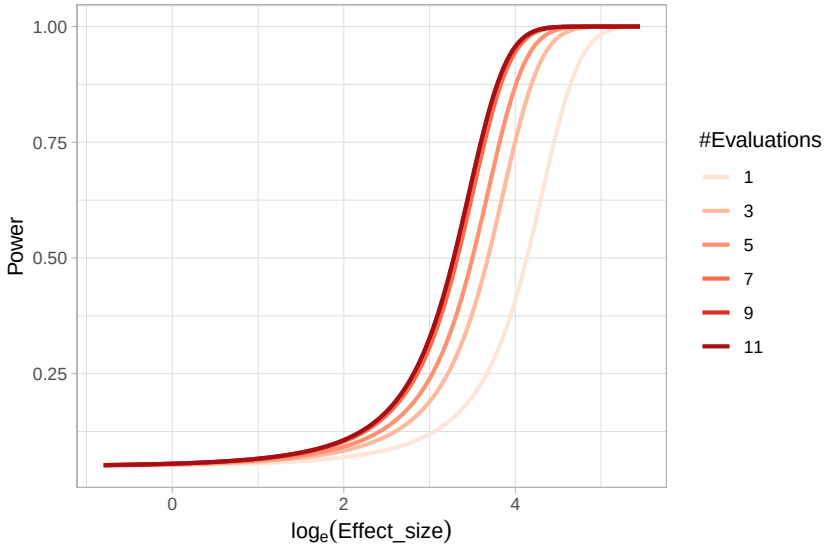


Figure A.2: *Change of power of visual tests for different number of evalutions on lineups with uniform fitted value distribution. The power will increase as the number of evaluations increases, but the margin will decrease.*

A.1.5 Power of a RESET Test under Different Auxiliary Regression Formulas

It is found in the result that the power of a RESET test will be affected by the highest order of fitted values included in the auxiliary formula. And we suspect that the current recommendation of the highest order - four, is insufficient to test complex non-linear structures such as the "triple-U" shape designed in this paper. Figure A.3 illustrates the change of power of RESET test while testing the "U" shape and the "triple-U" shape with different highest orders. Clearly, when testing a simple shape like the "U" shape, the highest order has very little impact on the power. But for testing the "triple-U" shape, there will be a loss of power if the recommended order is used. To avoid the loss of power, the highest order needs to be set to at least six.

A.1.6 Conventional Test Rejection Rate for Varying Significance Levels

In the main paper, Section 2.5.1 and Section 2.5.2 compared the power, and the decisions made by the conventional tests and the visual test. The power curves for the visual test is effectively a right-shift

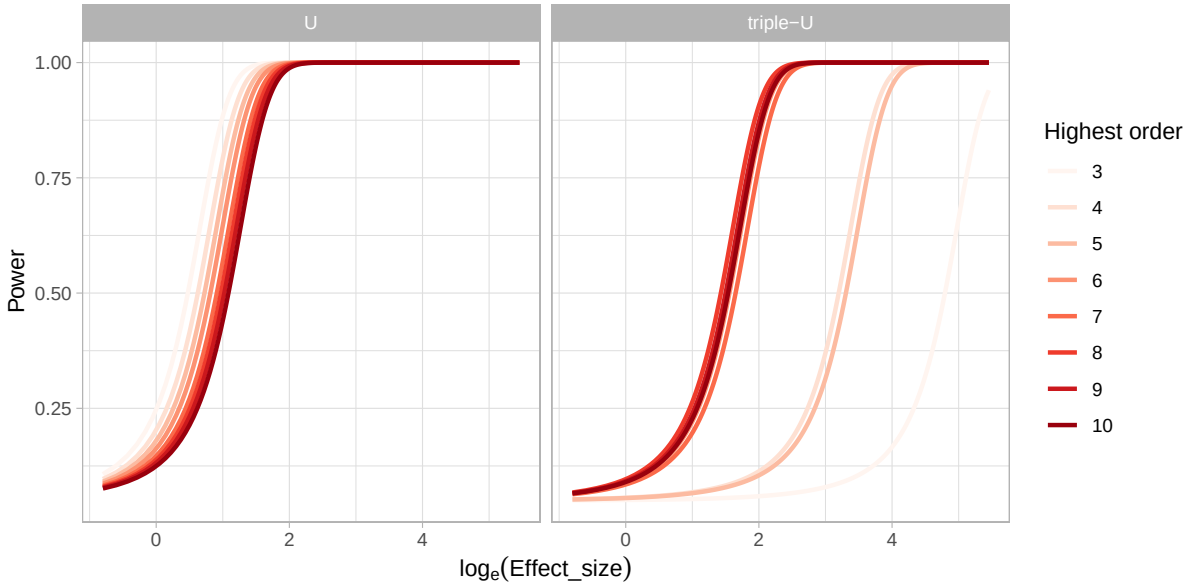


Figure A.3: *Change of power of RESET tests for different orders of fitted values included in the auxiliary formula. The left panel is the power of testing the "U" shape and the right panel is the power of testing the "triple-U" shape. The power will not be greatly affected by the highest order in the case of testing the "U" shape. In the case of testing the "triple-U" shape, the highest order needs to be set to at least six to avoid the loss of power.*

from the conventional test. The effect is that the visual test rejects less often than the conventional test, at the same significance level. We also saw that the visual test rejected a subset of those that the conventional tests rejected. This means that they agreed quite well - only residual plots rejected by the conventional tests were rejected by the visual test. There was little disagreement, where residual plots not rejected by the conventional test were rejected by the visual test. The question arises whether the decisions made conventional test could be made similar to that of the visual test by reducing the significance level. Reducing the significance level from 0.05, to 0.01, 0.001, ... will have the effect of rejecting fewer of the residual plots.

It would be interesting if a different conventional test significance level results in both the visual tests and conventional tests reject only the same residual plots, and fails to reject the same residual plots. This would be a state where both systems agree perfectly. Figure **??** examines this. Plot A shows the percentage of residual plots rejected by the visual test, given the conventional test rejected (solid lines) or failed to reject (dashed lines). The vertical grey line marks significance level 0.05. When the significance level gets smaller, it is possible to see that the visual tests reject (nearly) 100% of the time that the conventional test rejects. However, there is no agreement, because the visual tests also increasingly reject residual plots where the conventional test failed to reject. Plot B is comparable to an ROC curve, where the percentage visual test rejection conditional on conventional test decision is plotted: Reject conditional on reject is plotted against reject conditional on fail to reject, for different

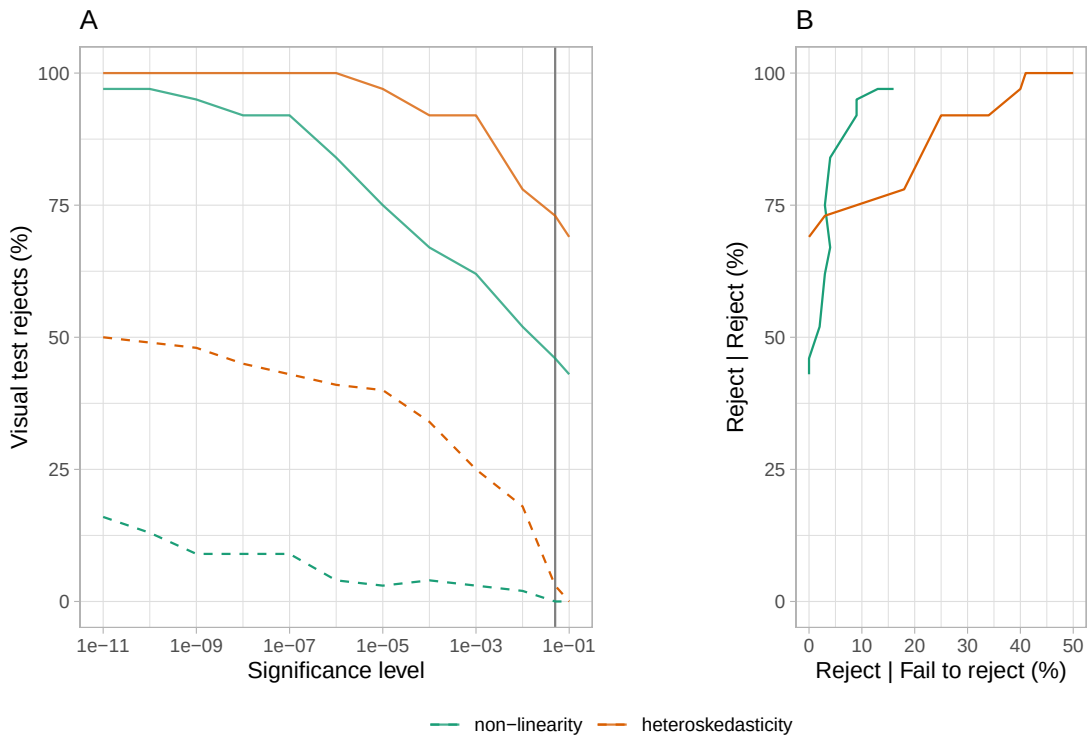


Figure A.4: *Changing the significance level of the conventional test will change the rejection rate. A: Percentage of conventional tests also rejected by the visual test: rejected (solid), not rejected (dashed). As the significance level is reduced, the percentage rejected by the visual test that has been rejected by the conventional test approaches 100. The percentage of tests not rejected by the conventional test also increases, as expected, but the percentage of these that are rejected by the visual test increases too. B: ROC curve shows that forcing the conventional test to not reject creates a discrepancy with the visual test. Many of the residual plots not rejected by the conventional test are rejected by the visual test. It is not possible to vary the significance level of the conventional test to match the decisions made by the visual test.*

significance levels. The non-linearity pattern results are close to being ideal, that the percentage of reject relative to fail to reject increases very slowly as the reject relative to reject converges to 100. The heteroskedasticity pattern is more problematic, and shows that the cost of rejecting less with the conventional test is disagreement with the visual test.

A.2 Additional Details of Experimental Setup

A.2.1 Non-linearity Model

The non-linearity model used in the experiment includes a non-linear term $\boldsymbol{z}$ constructed using Hermite polynomials on random vector $\boldsymbol{x}$ formulated as

$$\boldsymbol{y} = \mathbf{1}_n + \boldsymbol{x} + \boldsymbol{z} + \boldsymbol{\varepsilon},$$

$$\boldsymbol{x} = g(\boldsymbol{x}_{raw}, 1),$$

$$\boldsymbol{z} = g(\boldsymbol{z}_{raw}, 1),$$
$$\boldsymbol{z}_{raw} = He_j(g(\boldsymbol{x}, 2)),$$

where $\boldsymbol{y}$, $\boldsymbol{x}$, $\boldsymbol{\varepsilon}$, $\boldsymbol{x}_{raw}$, $\boldsymbol{z}_{raw}$ are vectors of size n, $\mathbf{1}_n$ is a vector of ones of size n, $He_j(.)$ is the jth-order probabilist's Hermite polynomials (Hermite 1864; originally by Laplace 1820), $\varepsilon \sim N(\mathbf{0}_n, \sigma^2 \boldsymbol{I}_n)$, and $g(\boldsymbol{x}, k)$ is a scaling function to enforce the support of the random vector to be $[-k, k]^n$ defined as

$$g(\boldsymbol{x}, k) = 2k \cdot \frac{\boldsymbol{x} - x_{min}\mathbf{1}_n}{x_{max} - x_{min}} - k, \quad \text{for} \quad k > 0. \tag{A.6}$$

where $x_{min} = \min_{i \in \{1,\ldots,n\}} x_i$, $x_{max} = \max_{i \in \{1,\ldots,n\}} x_i$ and x_i is the i-th entry of $\boldsymbol{x}$. The function `hermite` from the R package `mpoly` (Kahle 2013) is used to simulate $\boldsymbol{z}_{raw}$ to generate Hermite polynomials.

The null regression model used to fit the realizations generated by the above model is formulated as

$$\boldsymbol{y} = \beta_0 \mathbf{1}_n + \beta_1 \boldsymbol{x} + \boldsymbol{u}, \tag{A.7}$$

where $\boldsymbol{u} \sim N(\mathbf{0}, \sigma^2 \boldsymbol{I}_n)$. Here $\boldsymbol{z}$ is a higher order term of $\boldsymbol{x}$ and leaving it out in the null regression results in model misspecification.

A.2.2 Heteroskedasticity Model

The heteroskedasticity model used in the experiment is formulated as

$$\boldsymbol{y} = \mathbf{1}_n + \boldsymbol{x} + \boldsymbol{\varepsilon},$$
$$\boldsymbol{x} = g(\boldsymbol{x}_{raw}, 1),$$
$$\boldsymbol{\varepsilon} \sim N(\mathbf{0}_n, \mathbf{1}_n + (2 - |a|)(\boldsymbol{x} - a\mathbf{1}_n)'(\boldsymbol{x} - a\mathbf{1}_n) b \boldsymbol{I}_n),$$

where $\boldsymbol{y}$, $\boldsymbol{x}$, $\boldsymbol{\varepsilon}$ are vectors of size n and $g(.)$ is the scaling function defined in Equation A.6. The null regression model used to fit the realizations generated by the above model is formulated exactly the same as Equation A.7.

For $b \neq 0$, the variance-covariance matrix of the error term $\boldsymbol{\varepsilon}$ is correlated with the predictor $\boldsymbol{x}$, which will lead to the presence of heteroskedasticity.

Since $\text{supp}(X) = [-1, 1]$, choosing a to be -1, 0 and 1 can generate "left-triangle", "butterfly" and "right-triangle" shapes. The term $(2 - |a|)$ maintains the magnitude of residuals across different values

of a.

A.2.3 Controlling the Signal Strength

The three parameters n, σ and b are important for controlling the strength of the signal to generate lineups with a variety difficulty levels. This will ensure that estimated power curves will be smooth and continuous, and that participants are allocated a set of lineups with a range of difficulty.

Parameter $\sigma \in \{0.5, 1, 2, 4\}$ and $b \in \{0.25, 1, 4, 16, 64\}$ are used in data collection periods I and II respectively. A large value of σ will increase the variation of the error of the non-linearity model and decrease the visibility of the visual pattern. The parameter b controls the variation in the standard deviation of the error across the support of the predictor. A larger value of b generates a larger ratio between it is smallest and highest values, making the visual pattern more obvious. The sample size n sharpens (or blurs) a pattern when it is larger (or smaller).

A.2.4 Lineup Allocation to Participants

There are a total of $4 \times 4 \times 3 \times 4 = 192$ and $3 \times 5 \times 3 \times 4 = 180$ combinations of parameter values for the non-linearity model and heteroskedasticity models respectively. Three replications for each combination results in $192 \times 3 = 576$ and $180 \times 3 = 540$ lineups, respectively.

Each lineup needs to be evaluated by at least five participants. From previous work, and additional pilot studies for this experiment, we decided to record evaluations from 20 lineups for each participant. Two of the 20 lineups with clear visual patterns were used as attention checks, to ensure quality data. Thus, $576 \times 5/(20-2) = 160$ and $540 \times 5/(20-2) = 150$ participants were needed to cover the experimental design for the data collection periods I and II, respectively. The factor levels and range of difficulty was assigned relatively equally among participants.

Data collection period III was primarily to obtain Rorschach lineup evaluations to estimate α, and also to obtain additional evaluations of lineups made with uniform fitted value distribution to ensure consistency in the results. To construct a Rorschach lineup, the data is generated from a model with zero effect size, while the null data are generated using the residual rotation technique. This procedure differs from that of the canonical Rorschach lineup, where all 20 plots are generated directly from the null model, so the method suggested in VanderPlas et al. (2021) for typical lineups containing a data plot is used to estimate α. The $3 \times 4 = 12$ treatment levels of the common factors, replicated three times results in 36 lineups. And 6 more evaluations on the 279 lineups with uniform fitted value distribution, results in at least $(36 \times 20 + 279 \times 3 \times 6)/(20-2) = 133$ participants needed.

A.2.5 Mapping of Participants to Experimental Factors

Mapping of participants to experimental factors is an important part of experimental design. Essentially, we want to maximum the difference in factors exposed to a participant. For this purpose, we design an algorithm to conduct participant allocation. Let L be a set of available lineups and S be a set of available participants. According to the experimental design, the availability of a lineup is associated with the number of participants it can assign to. For lineups with uniform fitted value distribution, this value is 11. And other lineups can be allocated to at most five different participants. The availability of a participant is associated with the number of lineups that being allocated to this participant. A participant can view at most 18 different lineups.

The algorithm starts from picking a random participant $s \in S$ with the minimum number of allocated lineups. It then tries to find a lineup $l \in L$ that can maximise the distance metric D and allocate it to participant s. Set L and S will be updated and the picking process will be repeated until there is no available lineups or participants.

Let $F_1, ..., F_q$ be q experimental factors, and $f_1, ..., f_q$ be the corresponding factor values. We say f_i exists in L_s if any lineup in L_s has this factor value. Similarly, $f_i f_j$ exists in L_s if any lineup in L_s has this pair of factor values. And $f_i f_j f_k$ exists in L_s if any lineup in L_s has this trio of factor values. The distance metric D is defined between a lineup l and a set of lineups L_s allocated to a participant s if L_s is non-empty:

$$D = C - \sum_{1 \le i \le q} I(f_i \text{ exists in } L_s) - \sum_{\substack{1 \le i \le q-1 \\ i < j \le q}} I(f_i f_j \text{ exists in } L_s) - \sum_{\substack{1 \le i \le q-2 \\ i < j \le q-1 \\ j < k \le q}} I(f_i f_j f_k \text{ exists in } L_s)$$

where C is a sufficiently large constant such that $D > 0$. If L_s is empty, we define $D = 0$.

The distance measures how different a lineup is from the set of lineups allocated to the participant in terms of factor values. Thus, the algorithm will try to allocate the most different lineup to a participant at each step.

In Figure A.5 and Figure A.6, we present examples of lineup allocations generated by our algorithm for data collection periods I and II, respectively. These figures illustrate the factor values presented to the first 20 participants recruited during each period. It can be observed that the algorithm effectively distributed almost all possible one-way and two-way factor combinations among the participants, thereby ensuring a diverse set of lineups for each participant.

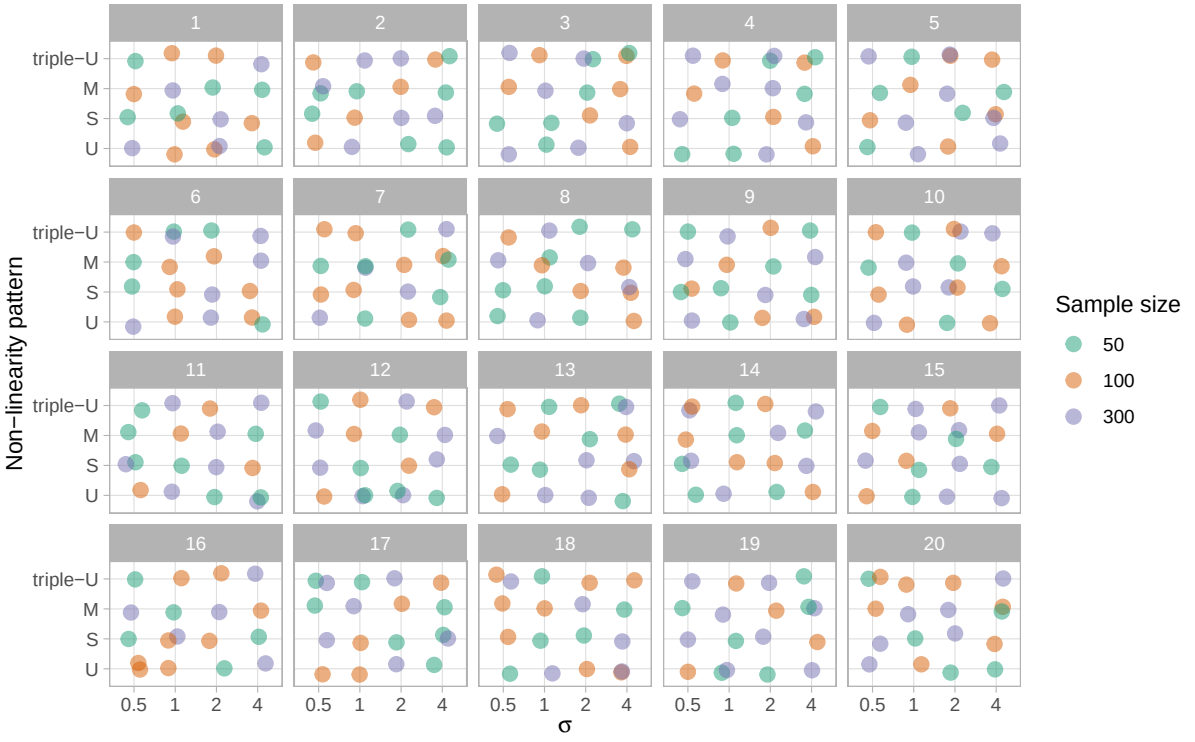


Figure A.5: *Factor values assigned to the first 20 participants recruited during data collection period I, with data points slightly jittered to prevent overlap. Each participant have been exposed to all one-way and two-way factor combinations.*

A.2.6 Data Collection Process

The survey data is collected via a self-hosted website designed by us. The complete architecture is provided in Figure A.7. The website is built with the `Flask` (Grinberg 2018) web framework and hosted on `PythonAnywhere` (PythonAnywhere LLP 2023). It is configured to handle HTTP requests such that participants can correctly receive webpages and submit responses. Embedded in the resources sent to participants, the `jsPsych` front-end framework (De Leeuw 2015) instructs participants' browsers to render an environment for running behavioural experiments. During the experiment, this framework will automatically collect common behavioural data such as response time and clicks on buttons. participants' responses are first validated by a scheduled `Python` script run on the server, then push to a Github repository. Lineup images shown to users are saved in multiple Github repositories and hosted in corresponding Github pages. The URLs to these images are resolved by `Flask` and bundled in HTML files.

Once the participant is recruited from Prolific (Palan and Schitter 2018), it will be redirected to the entry page of our study website. An image of the entry page is provided in Figure A.8. Then, the participant needs to submit the online consent form and fill in the demographic information as shown in Figure A.9 and Figure A.10 respectively. Before evaluating lineups, participant also need to read

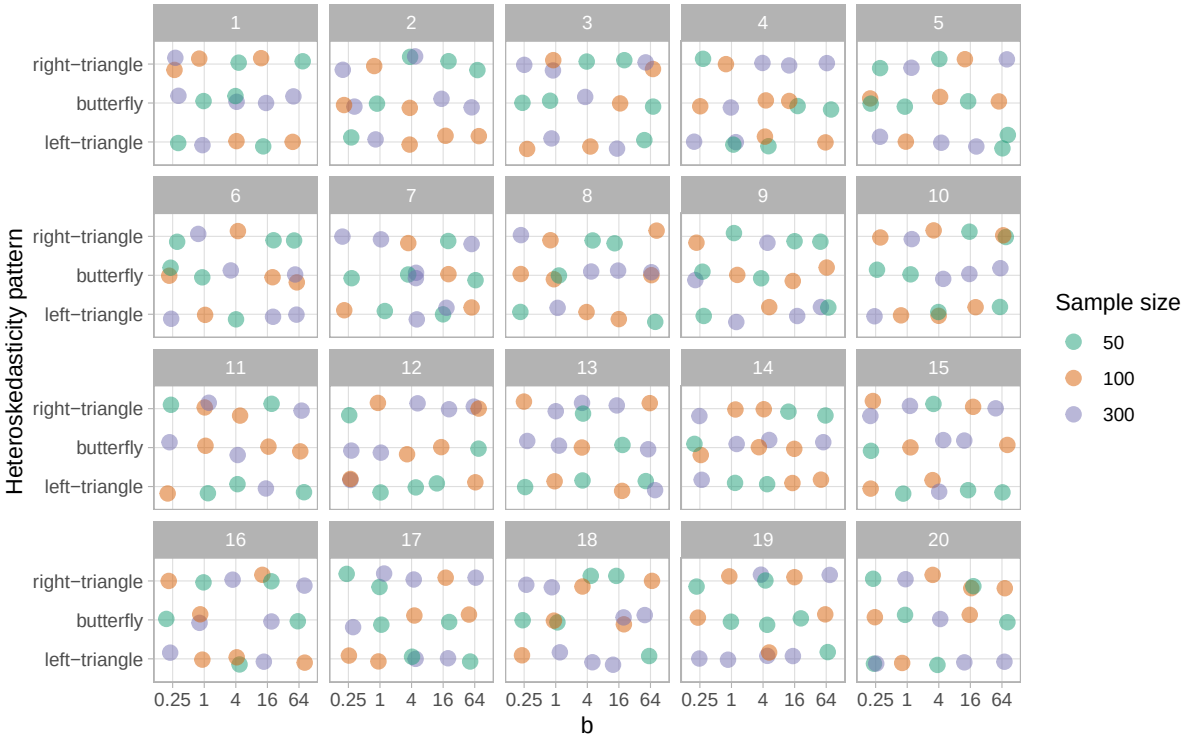


Figure A.6: *Factor values assigned to the first 20 participants recruited during data collection period II, with data points slightly jittered to prevent overlap. Each participant appears to have been exposed to almost all one-way and two-way factor combinations. However, two participants, namely participant 16 and participant 18, were not presented with lineups containing a butterfly shape with* $b = 4$*. Additionally, participant 7 did not encounter a lineup featuring a butterfly shape with* $b = 1$*.*

the training page as provide in Figure A.11 to understand the process. An example of the lineup page is given in Figure A.12. A half of the page is taken by the lineup image to attract participant's attention. The button to skip the selections for the current lineup is intentionally put in the corner of the bounding box with smaller font size, such that participants will not misuse this functionality.

A.3 Analysis of Results Relative to Data Collection Process

A.3.1 Demographics

Throughout the study, we have collected 7254 evaluations on 1116 non-null lineups. Table A.2 gives further details about the number of evaluations, lineups and participants over pattern types and data collection periods.

Along with the responses to lineups, we have collected a series of demographic information including age, pronoun, education background and previous experience in studies involved data visualization. Table A.3, Table A.4, Table A.5 and Table A.6 provide summary of the demographic data.

It can be observed from the tables that most participants have Diploma or Bachelor degrees, followed

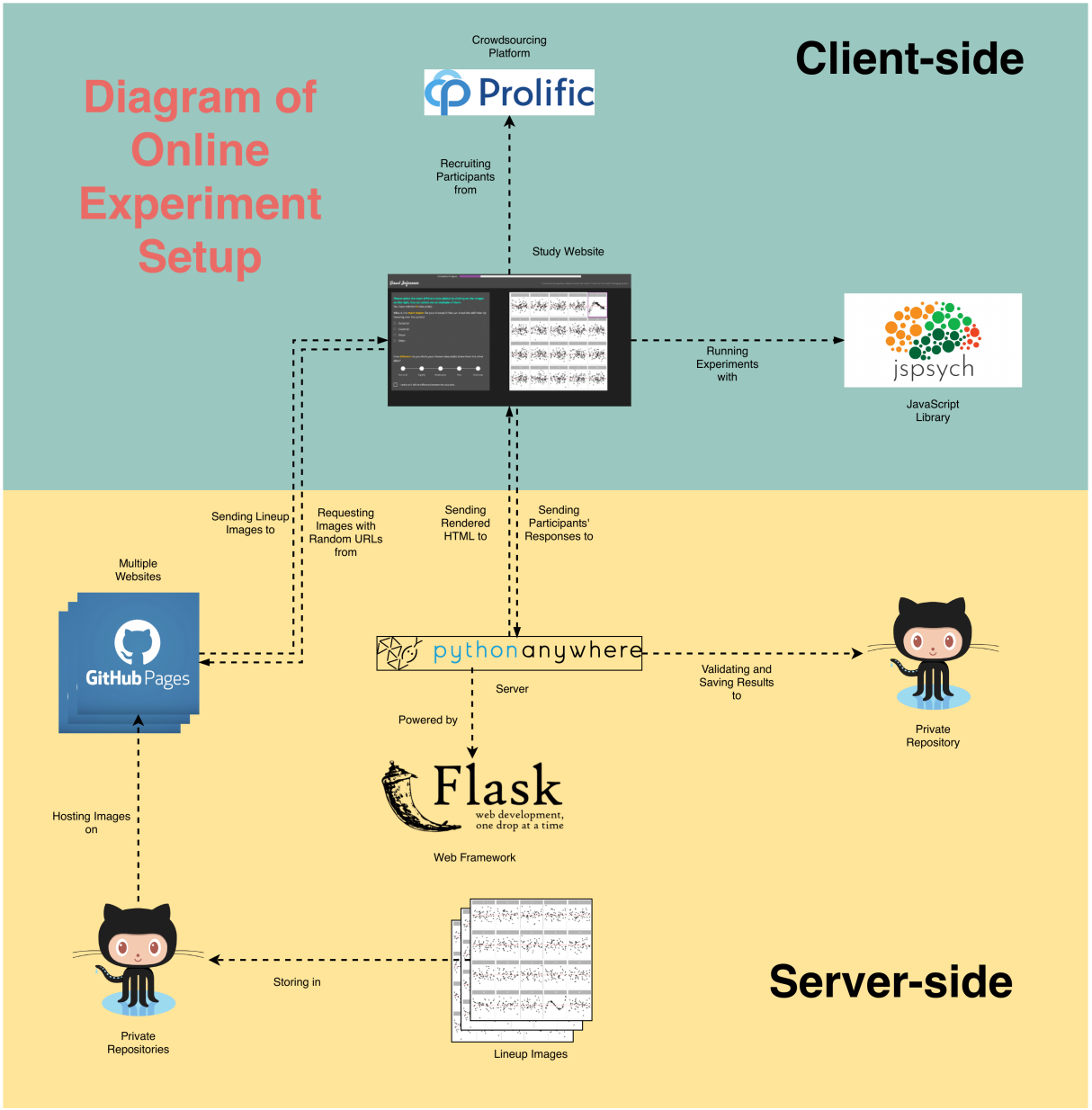


Figure A.7: *Diagram of online experimental setup. The server-side of the study website uses Flask as backend hosted on PythonAnywhere. And the client-side uses jsPsych to run experiment.*

Table A.2: *Count of lineups, evaluations and participants over departure types and data collection periods.*

	Non-linearity			Heteroskedasticity			
Number	I	II	III	I	II	II	Total
Lineups	576	0	144	0	540	135	1116
Evaluations	2880	0	864	0	2700	810	7254
Participants	160	0	123	0	160	123	443

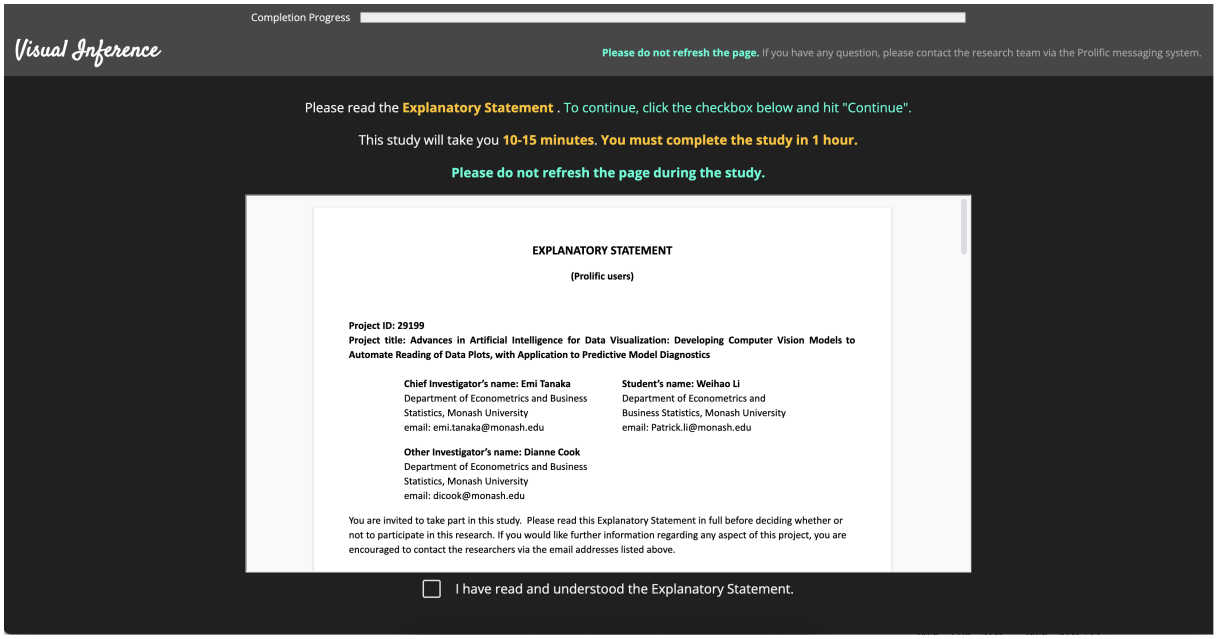


Figure A.8: *The entry page of the study website.*

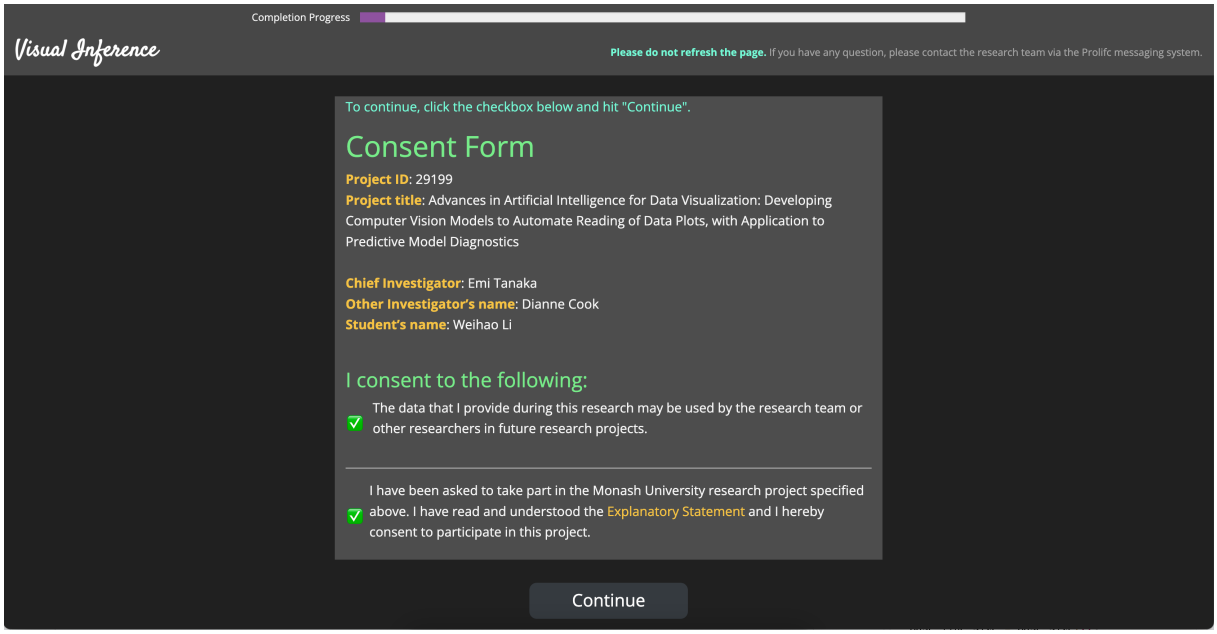


Figure A.9: *The consent form provided in the study website.*

by High school or below and the survey data is gender balanced. Majority of participants are between 18 to 39 years old and there are slightly more participants who do not have previous experience than those who have.

A.3.2 Data Collection Periods

We have the same type of model collected over different data collection periods, that may lead to unexpected batch effect. Figure A.13 and Figure A.14 provide two lineups to examine whether there is an actual difference across data collection periods for non-linearity model and heteroskedasticity

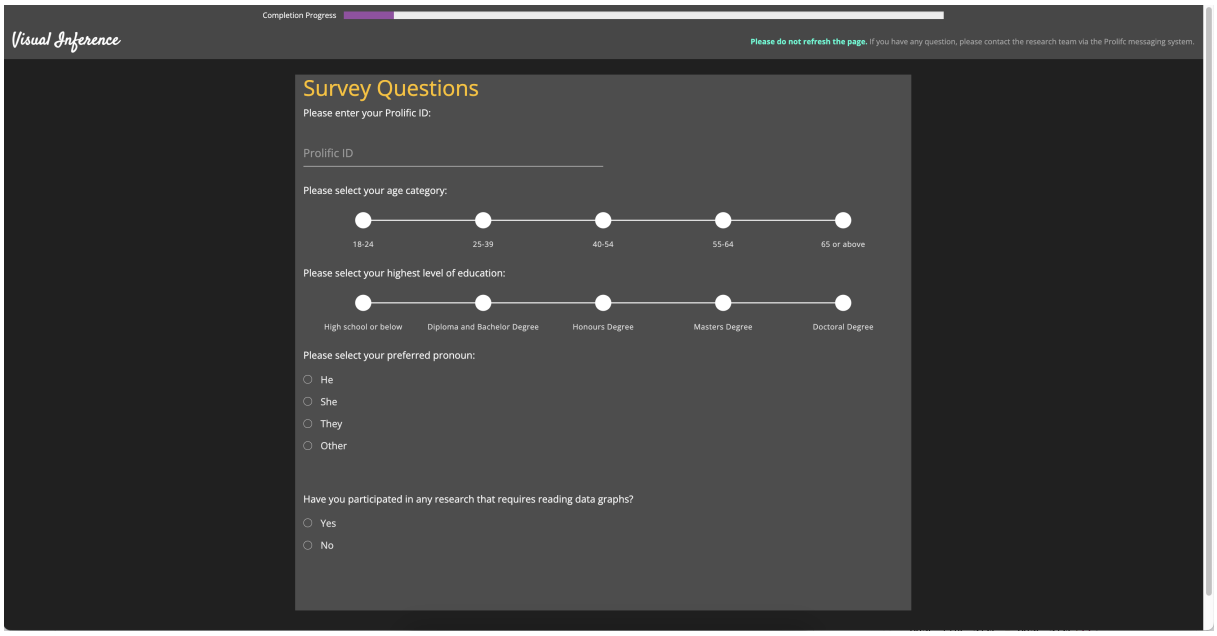


Figure A.10: *The form to provide demographic information.*

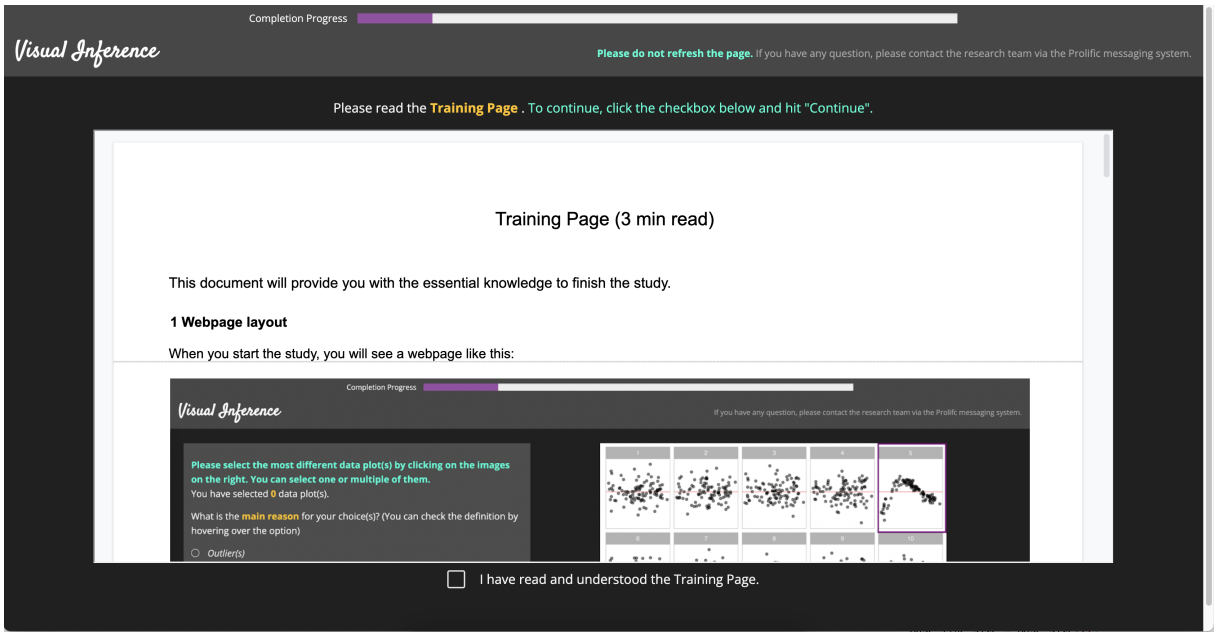


Figure A.11: *The training page of the study website.*

Table A.3: *Summary of pronoun distribution of participants recruited in this study.*

Pronoun	Period I	%	Period II	%	Period III	%	Total	%
He	77	17.4	79	17.8	61	13.8	217	49.0
She	78	17.6	77	17.4	61	13.8	216	48.8
Other	5	1.1	4	0.9	1	0.2	10	2.3
	160	36.1	160	36.1	123	27.8	443	100.0

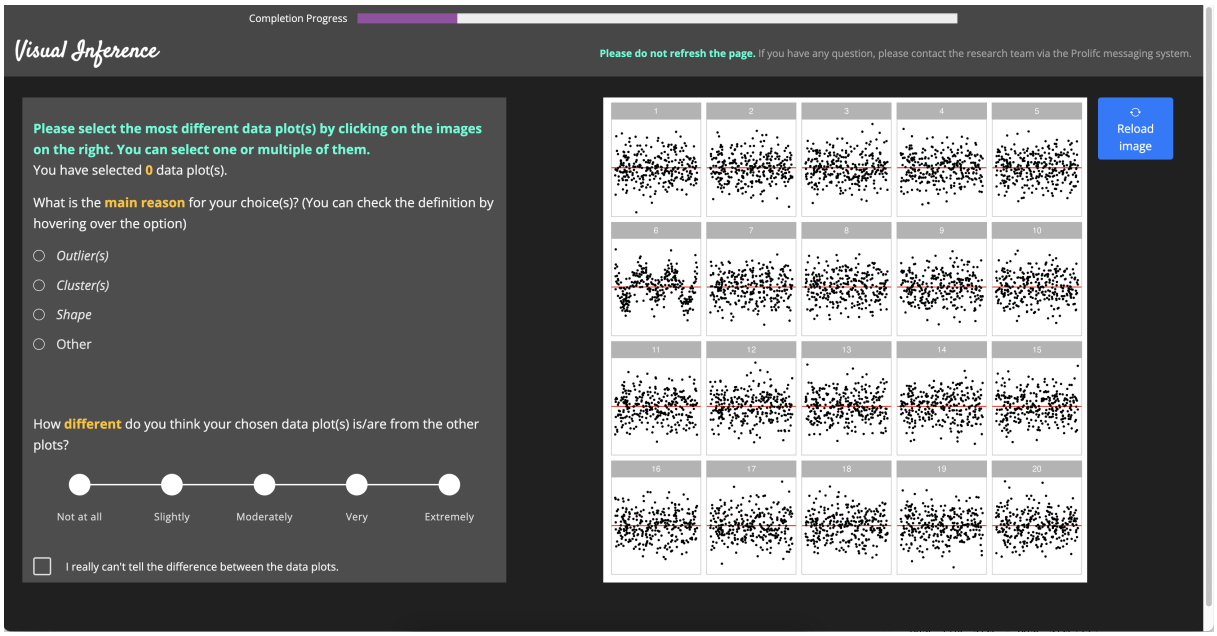


Figure A.12: *The lineup page of the study website.*

Table A.4: *Summary of age distribution of participants recruited in this study.*

Age group	Period I	%	Period II	%	Period III	%	Total	%
18-24	83	18.7	86	19.4	51	11.5	220	49.7
25-39	69	15.6	63	14.2	63	14.2	195	44.0
40-54	6	1.4	8	1.8	6	1.4	20	4.5
55-64	2	0.5	3	0.7	3	0.7	8	1.8
	160	36.1	160	36.1	123	27.8	443	100.0

Table A.5: *Summary of education distribution of participants recruited in this study.*

Education	Period I	%	Period II	%	Period III	%	Total	%
High School or below	41	9.3	53	12.0	33	7.4	127	28.7
Diploma and Bachelor Degree	92	20.8	79	17.8	66	14.9	237	53.5
Honours Degree	6	1.4	15	3.4	6	1.4	27	6.1
Masters Degree	21	4.7	13	2.9	16	3.6	50	11.3
Doctoral Degree	0	0.0	0	0.0	2	0.5	2	0.5
	160	36.1	160	36.1	123	27.8	443	100.0

Table A.6: *Summary of previous experience distribution of participants recruited in this study.*

Previous experience	Period I	%	Period II	%	Period III	%	Total	%
No	96	21.7	88	19.9	67	15.1	251	56.7
Yes	64	14.4	72	16.3	56	12.6	192	43.3
	160	36.1	160	36.1	123	27.8	443	100.0

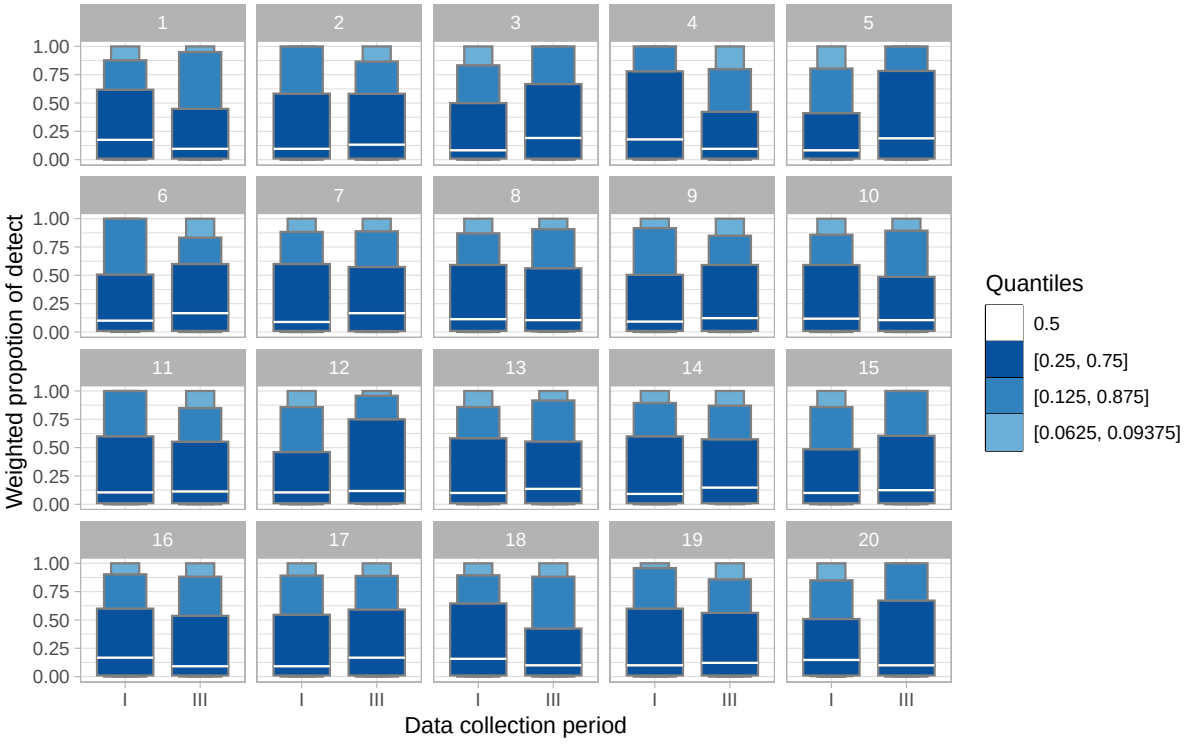


Figure A.13: *A lineup of "letter-value" boxplots of weighted propotion of detect for lineups over different data collection periods for non-linearity model. Can you find the most different boxplot? The data plot is positioned in panel* $2^3 - 1$.

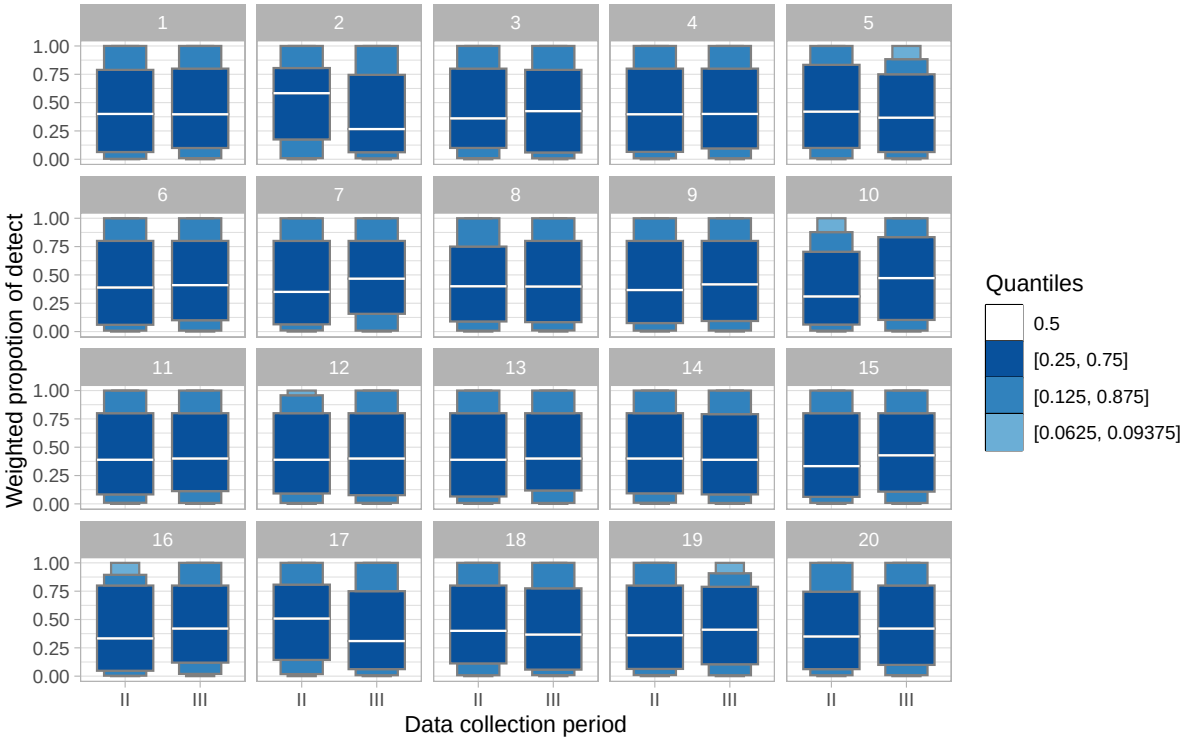


Figure A.14: *A lineup of "letter-value" boxplots of weighted propotion of detect for lineups over different data collection periods for heteroskedasticity model. Can you find the most different boxplot? The data plot is positioned in panel* $2^4 - 2$.

model respectively. To emphasize the tail behaviour and display fewer outliers, we use the "letter-value" boxplot (Hofmann et al. 2017) which is an extension of the number of "letter value" statistics to check the weighed proportion of detect over different data collection period. The weighted proportion of detect is calculated by taking the average of c_i of a lineup over a data collection period. Within our research team, we can not identify the data plot from the null plots for these two lineups, result in p-values much greater than 5%. Thus, there is no clear evidence of batch effect.

Appendix B

Appendix to "Automated Assessment of Residual Plots with Computer vision Models"

B.1 Neural Network Layers Used in the Study

This study used seven types of neural network layers, all of which are standard components frequently found in modern deep learning models. These layers are well-documented in textbooks like Goodfellow et al. (2016) and Chollet (2021), which offer thorough explanations and mathematical insights. In this section, we will offer a concise overview of these layers, drawing primarily from the insights provided in Goodfellow et al. (2016).

B.1.1 Dense Layer

The Dense layer, also known as the fully-connected layer, is the fundamental unit in neural networks. It conducts a matrix multiplication operation between the input matrix $\boldsymbol{I}$ and a weight matrix $\boldsymbol{W}$ to generate the output matrix $\boldsymbol{O}$, which can be written as

$$\boldsymbol{O} = \boldsymbol{I}\boldsymbol{W} + b,$$

where b is the intercept.

B.1.2 ReLu Layer

The ReLU layer, short for rectified linear unit, is an element-wise non-linear function introduced by Nair and Hinton (2010). It sets the output elements to zero if the corresponding input element is

negative; otherwise, it retains the original input. Mathematically, it can be expressed as:

$$\boldsymbol{O}(i,j) = max\{0, \boldsymbol{I}(i,j)\},$$

where $\boldsymbol{O}(i,j)$ is the ith row and jth column entry of matrix $\boldsymbol{O}$, and $\boldsymbol{I}(i,j)$ is the ith row and jth column entry of matrix $\boldsymbol{I}$.

B.1.3 Convolutaional Layer

In Dense layers, matrix multiplication leads to each output unit interacting with every input unit, whereas convolutional layers operate differently with sparse interactions. An output unit in a convolutional layer is connected solely to a subset of input units, and the weight is shared across all input units. Achieving this involves using a kernel, typically a small square matrix, to conduct matrix multiplication across all input units. Precisely, this concept can be formulated as:

$$\boldsymbol{O}(i,j) = \sum_m \sum_n \boldsymbol{I}(i-m, j-n)K(m,n),$$

where m and n are the row and columns indices of the kernel K.

If there are multiple kernels used in one covolutional layer, then each kernel will have its own weights and the output will be a three-dimensional tensor, where the length of the third channel is the number of kernels.

B.1.4 Pooling Layer

A pooling layer substitutes the input at a specific location with a summary statistic derived from nearby input units. Typically, there are two types of pooling layers: max pooling and average pooling. Max pooling computes the maximum value within a rectangular neighborhood, while average pooling calculates their average. Pooling layers helps making the representation approximately invariant to minor translations of the input. The output matrix of a pooling layer is approximately s times smaller than the input matrix, where s represents the length of the rectangular area. A max pooling layer can be formulated as:

$$\boldsymbol{O}(i,j) = \max_{m,n} \boldsymbol{I}(si+m, sj+n).$$

B.1.5 Global Pooling Layer

A global pooling layer condenses an input matrix into a scalar value by either extracting the maximum or computing the average across all elements. This layer acts as a crucial link between the convolutional

structure and the subsequent dense layers in a neural network architecture. When convolutional layers uses multiple kernels, the output becomes a three-dimensional tensor with numerous channels. In this scenario, the global pooling layer treats each channel individually, much like distinct features in a conventional classifier. This approach facilitates the extraction of essential features from complex data representations, enhancing the network's ability to learn meaningful patterns. A global max pooling layer can be formulated as

$$O(i,j) = \max_{m,n,k} I(si+m, sj+n, k),$$

where k is the kernel index.

B.1.6 Batch Normalization Layer

Batch normalization is a method of adaptive reparametrization. One of the issues it adjusts is the simultaneous update of parameters in different layers, especially for network with a large number layers. At training time, the batch normalization layer normalizes the input matrix I using the formula

$$\boldsymbol{O} = \frac{\boldsymbol{I} - \boldsymbol{\mu}_I}{\boldsymbol{\sigma}_I},$$

where $\boldsymbol{\mu}_I$ and $\boldsymbol{\sigma}_I$ are the mean and the standard deviation of each unit respectively.

It reparametrizes the model to make some units always be standardized by definition, such that the model training is stabilized. At inference time, $\boldsymbol{\mu}_I$ and $\boldsymbol{\sigma}_I$ are usually replaced with the running mean and running average obtained during training.

B.1.7 Dropout Layer

Dropout is a computationally inexpensive way to apply regularization on neural network. For each input unit, it randomly sets to be zero during training, effectively training a large number of subnetworks simultaneously, but these subnetworks share weights and each will only be trained for a single steps in a large network. It is essentially a different implementation of the bagging algorithm. Mathematically, it is formulated as

$$\boldsymbol{O}(i,j) = \boldsymbol{D}(i,j)\boldsymbol{I}(i,j),$$

where $\boldsymbol{D}(i,j) \sim B(1,p)$ and p is a hyperparameter that can be tuned.