NP4VTT

A new software for estimating the value of travel time with nonparametric models

Hernández, José Ignacio; van Cranenburgh, Sander

DOI
10.1016/j.jocm.2023.100427

Publication date
2023

Document Version
Final published version

Published in
Journal of Choice Modelling

Citation (APA)
Hernández, J. I., & van Cranenburgh, S. (2023). NP4VTT: A new software for estimating the value of travel time with nonparametric models. *Journal of Choice Modelling*, *48*, Article 100427. https://doi.org/10.1016/j.jocm.2023.100427

Journal of Choice Modelling 48 (2023) 1–16

Contents lists available at ScienceDirect

Journal of Choice Modelling

journal homepage: www.elsevier.com/locate/jocm

NP4VTT: A new software for estimating the value of travel time with nonparametric models

José Ignacio Hernández *, Sander van Cranenburgh

Transport and Logistics Group, Department of Engineering Systems and Services, Faculty of Technology, Policy and Management, Delft University of Technology, the Netherlands

A B S T R A C T

Two-attribute-two-alternative stated choice experiments are widely used to infer the Value-of-Travel-Time (VTT) distribution. Two-attribute-two-alternative stated choice experiments have the advantage that their data can be analysed using nonparametric models, which allow for the inference of the VTT distribution without having to impose assumptions on its shape. However, a software package that enables researchers to estimate nonparametric models promptly is currently lacking. As a result, nonparametric models are underused. This paper aims to fill this software void. It presents NP4VTT, a Python package that enables researchers to estimate and compare nonparametric models in a fast and convenient way. It comprises five nonparametric models for estimating the VTT distribution from data coming from two-attribute-two-alternative stated choice experiments. We illustrate the use of NP4VTT by applying it to the Norwegian 2009 VTT data. We hope this software package will help researchers studying the VTT make more informed decisions concerning the shape of the VTT distribution and encourages the use and development of nonparametric models for choice behaviour analyses.

1. Introduction

Accurate inference of the Value-of-Travel-Time (VTT) is of great importance for policy appraisal (Small, 2012). In current practice, inference of the VTT and its distribution is commonly based on data from Stated Choice (SC) experiments. In the VTT literature, there are two streams of designing VTT SC experiments. The first stream propagates the use of choice tasks with three (or more) attributes. Thus, besides travel cost and travel time, the choice task comprises a third (and possibly a fourth) attribute, such as congestion level or reliability. Proponents of this stream argue that choice tasks with three or more attributes contribute lead behaviourally realistic choice tasks (Hess et al., 2020). The second stream propagates the use of choice tasks with only two attributes: travel cost and travel time. Proponents of this stream argue that two-attribute-two-alternative designs enable respondents to trade off travel cost and travel time in a clean and crisp way, thereby yielding more reliable VTT estimates than one would obtain from an experiment that is 'polluted' by context effects or interactions arising from the use of more attributes. This stream has a strong base in Western and Northern European countries, including the United Kingdom, the Netherlands, Denmark, and Sweden (e.g., Fosgerau et al., 2007; Kouwenhoven et al., 2014).

From a modelling perspective, two-attribute-two-alternative experimental designs can be analysed with nonparametric and semi-nonparametric models. In contrast to their parametric counterpart — which is dominant in the analysis of data from three or more attribute SC experiments — (semi-) nonparametric models do not impose prior assumptions on the shape of the VTT distribution. Hence, in nonparametric models, the analyst does not need to predefine the distribution of the VTT (e.g., lognormal, normal, log uniform, etc.) before estimation. As a result of their agnosticism to the shape of the VTT distribution, nonparametric models can more

* Corresponding author. Jaffalaan 5, 2628BX, Delft, the Netherlands.
E-mail addresses: J.I.Hernandez@tudelft.nl (J.I. Hernández), s.vancranenburgh@tudelft.nl (S. van Cranenburgh).

https://doi.org/10.1016/j.jocm.2023.100427
Received 30 October 2022; Received in revised form 31 May 2023; Accepted 2 June 2023

Table 1
Data structure required by NP4VTT.

Variable	Unique identifier of the respondent	Choice	Cost alternative 1	Time alternative 1	Cost alternative 2	Time alternative 2
Data type	Integer	Integer	Float	Float	Float	Float
	1	1	25	30	30	20
	1	1	25	40	20	45
	1	2	25	40	15	45
	2	1	15	45	20	30
	2	2	20	30	25	20
	…	…	…	…	…	…

accurately recover the VTT distribution. This desirable feature of nonparametric models explains their increasing use in recent VTT studies (e.g. Börjesson and Eliasson, 2014). In particular, nonparametric models are used to help the analyst decide which parametric distribution to use for the estimation of a (parametric) Random Utility Maximisation (RUM) model, which will be used for policy appraisal.

However, although the merits of nonparametric models are increasingly acknowledged in VTT studies, the burden of using them is high. Whereas for their parametric counterparts — i.e. discrete choice models — numerous software packages and libraries are nowadays available, such as PandasBiogeme (Bierlaire, 2020), Apollo (Hess and Palma, 2019), and Stata (StataCorp, 2005), for nonparametric models this is not the case. Analysts that wish to estimate nonparametric models often need to code their models from scratch. This hampers the more widespread use of nonparametric models for VTT analysis.

This paper fills this software void. It presents a new software package called *NP4VTT*. NP4VTT is a Python package that enables analysts to estimate a range of (semi-)nonparametric models to recover the VTT distribution. More specifically, NP4VTT comprises five nonparametric models that have been used in recent VTT studies, namely: Local constant model[1] (Fosgerau, 2006, 2007), Local logit (Fosgerau, 2007), Rouwendal model (Rouwendal et al., 2010), Artificial Neural Network (ANN) based VTT model (van Cranenburgh and Kouwenhoven, 2021), and a Logistic Regression based VTT model (van Cranenburgh and Kouwenhoven, 2021). For completeness and serving as a benchmark, we also added a parametric Random Valuation (RV) model to this package.

While the models included in NP4VTT vary significantly in terms of how they operate, they have in common that they all work on two-attribute-two-alternative choice data. The strength of NP4VTT is that it enables analysts to apply all these nonparametric models in an easy and convenient way and enables comparison of their results, considering their respective strengths and weaknesses. Furthermore, NP4VTT is not confined to VTT applications, it can more generally be used to uncover the distribution of the substitution rate between any two attributes of a two-attribute-two-alternative choice task. Furthermore, to illustrate the use of NP4VTT inside and outside VTT applications, we provide two examples in a Git repository.[2] The first example illustrates how each nonparametric model included in NP4VTT is used to uncover the VTT distribution from the Norwegian 2009 VTT study (Ramjerdi et al., 2010). The second example shows how NP4VTT can be used to calculate the distribution of the "willingness to pay for reducing environmental damage" in a hypothetical case study: we designed a hypothetical two-attribute-two-alternative choice experiment based on the Contingent Valuation (CV) survey to assess the damages of the Exxon Valdez oil spill in 1989 (Carson et al., 2003).[3]

The remaining part of this paper is organised as follows. Section 2 introduces the data format and describes the five nonparametric models implemented in this software package. Descriptions of the nonparametric models are kept short. They are meant to convey the general idea of a model, not to provide in-depth expositions. Readers interested in the technical details are referred to the original works where the nonparametric models are introduced. Section 3 presents the NP4VTT software. Section 4 illustrates the use of the package by applying it to the data from the Norwegian 2009 VTT study. Section 5 provides a brief discussion of future developments and the next steps.

2. Data format and nonparametric models

2.1. Data format

Table 1 shows the data structure and data types that NP4VTT requires. NP4VTT operates on data for two-attribute-two-alternative choice experiments. In this format, each choice observation comprises at least six input variables: the travel costs and travel times of the two alternatives, plus the choice. Additionally, each respondent must have a unique identifier, which is repeated for multiple choice situations answered by a same respondent.

NP4VTT distinguishes between cross-sectional, balanced and unbalanced panel data. Cross-sectional data is when each respondent answers exactly one choice situation. Balanced panel data is when all respondents answered two or more choice situations and all

[1] Regarding, terminology: nonparametric models are also referred to as "estimators". However, throughout this paper, we use the word "models" for reasons of coherence instead of mixing "estimator", "model" and "method". Likewise, for reasons of coherence, we stick with the word "estimation", despite that for some "models", "training" is more appropriate. Finally, we refer to all models as "nonparametric", despite that a further distinction can be made into "semi" and "full" nonparametric models.

[2] The URL of the Git repository is https://github.com/ighdez/py-np4vtt.

[3] We appreciate the suggestion of one anonymous reviewer to elaborate more on applications of NP4VTT outside the transportation field.

respondents answered the same number of choice situations. Hence, unbalanced panel data is when all respondents answered two or more choice situations, but the number of choice situations answered across respondents differs. NP4VTT automatically identifies whether the dataset contains cross-sectional, balanced or unbalanced panel data. In case the dataset does contain either cross-sectional or unbalanced panel data, only the Local constant, Local logit and RV models are enabled.

Dominant choice tasks are not permitted: each choice observation must have a *slow and cheap* alternative and a *fast and expensive* alternative. In the case of a dominant alternative, it must be removed. NP4VTT runs integrity checks before estimation and will prompt error messages if dominant alternatives are present in the data.

NP4VTT computes the Boundary-Value-of-Travel-Time (BVTT). The BVTT is the implicit price of time in a two-attribute-two-alternative choice task (Cameron and James, 1987). Often the BVTT is perceived as a valuation threshold (Ojeda-Cabral and Chorus, 2016), meaning that a respondent choosing the fast and expensive alternative reveals having a VTT above the BVTT; a respondent choosing the slow and cheap alternative reveals having a VTT below the BVTT. The formula to compute the BVTT is given in equation (1), where t_1 and c_1 denote the travel time and travel cost of the *slow and cheap* alternative and t_2 and c_2 denote the travel time and travel cost of the *fast and expensive* alternative.

$$BVTT = -\frac{c_1 - c_2}{t_1 - t_2}, \tag{1}$$

2.2. Local constant model

The Local constant model is pioneered by Fosgerau (2006, 2007) for studying the VTT distribution from cross-sectional data. The Local constant model is an approach based on the regression model $y = f(BVTT) + \varepsilon$, in which the aim is to get an approximation of the function f in a nonparametric fashion using a kernel density (Nadaraya-Watson) estimator.

We formalise the Local constant model used in this paper as follows. Let y be an indicator variable which equals one if a respondent chooses the slow and cheap alternative and zero otherwise. Define the conditional utility of each alternative as $\alpha_t t_i + \alpha_c c_i$, in which α_t and α_c are random parameters independent across choice tasks, while the subscript i denotes the slow and cheap (1) and fast and expensive (2) alternatives. Then, the condition under which a respondent chooses the slow and cheap alternative is given in equation (2):

$$\alpha_t t_1 + \alpha_c c_1 > \alpha_t t_2 + \alpha_c c_2. \tag{2}$$

As a result, y can be rewritten as equation (3):

$$y = 1\{\alpha_t t_1 + \alpha_c c_1 > \alpha_t t_2 + \alpha_c c_2\}, = 1\left\{\frac{\alpha_t}{\alpha_c} < -\frac{c_1 - c_2}{t_1 - t_2}\right\}, = 1\{w < BVTT\}. \tag{3}$$

Equation (3) suggests that we observe a respondent choosing the slow and cheap alternative, i.e., $y = 1$), when his/her unobserved VTT (represented by $w = \alpha_t/\alpha_c$), is lower than the BVTT. Conversely, a respondent chooses the fast and expensive alternative when his/her VTT is higher than the BVTT.

Furthermore, notice that $P(y = 1) = P(w < BVTT) = F_w(BVTT)$, where $F_w(\cdot)$ is a Cumulative Distribution Function (CDF) of w. Therefore, we can define the model as in equation (4):

$$y = F_w(BVTT) + \eta. \tag{4}$$

The objective is to get an estimate of $F_w(BVTT)$. Fosgerau (2006) proposes to approximate F_w in a nonparametric way, using the Nadaraya-Watson estimator. Following Fosgerau (2007), given a point x_0 defined in the support of the VTT, the estimate of F_w around x_0 is given by equation (5):

$$\widehat{F}_w(x_0) = \frac{\sum_{i \le N} K\left(\frac{BVTT_i - x_0}{h}\right) y_i}{\sum_{i \le N} K\left(\frac{BVTT_i - x_0}{h}\right)}, \tag{5a}$$

where $K(\cdot)$ is a kernel function and h is a user-defined smoothing bandwidth parameter. The idea behind of equation (5) is to have a weighted average of the probability on each point of the VTT support. $K(\cdot)$ weights the distance between each point of the BVTT and x_0. *BVTT* s close to the support point x_0, $K(\cdot)$ weight comparatively heavily, and vice versa, *BVTT* s that are far from the support point weight comparatively lightly. The bandwidth parameter h controls the smoothness of $\widehat{F}_w$. A large value of h will result in a smooth estimate of the CDF, at the cost of underfitting; a small value of h will result in a comparatively more erratic estimate of the CDF.

NP4VTT implements a standard normal (gaussian) kernel function for the Local constant model. The construction of the VTT distribution is done by estimating $\widehat{F}_w$ for the mid points of a user-defined grid of VTT support points. The user controls the minimum and maximum values of the grid, as well as the number of support points in between.

2.3. Local logit

The Local logit model is a model for cross-sectional data first proposed by Fan et al. (1995), and later pioneered by Fosgerau (2007) in the VTT literature. As the name suggests, the Local logit model involves estimation of a series of 'local' logits. In this context, 'local'

refers to the notion that the logit models are estimated on a subset of the data. In the VTT context, this means logit models are estimated on subsets that are created based on the BVTT. For instance, a first subset may contain all choice observations for which holds 0 €/hr $<$ $BVTT < 10$ €/hr, a second subset may contain all choice observations for which holds 10 €/hr $\leq BVTT < 20$ €/hr, and so on. The centre of the first bin then sits at 5 €/hr, of the second bin at 15 €/hr, and so. On each of these subsets, a Logistic regression (hence logit) is estimated.

This logit model comprises two coefficients: an intercept for the mid-point and a linear term capturing the distance from the centre of the bin. In the log-likelihood function, the weight of each data point is computed using a kernel function. The further away (in terms of BVTT space) a data point is from the centre of the bin, the lower the impact of the prediction of that data point in the log-likelihood function. After all, a data point further away from the bin centre contains less information on the VTT at the bin. In NP4VTT, we use a triangular kernel function. This means that the contribution of an observation to the log-likelihood function decreases linearly with the distance from the centre of the bin.

2.4. Rouwendal model

Rouwendal et al. (2010) propose a nonparametric model to estimate the VTT and the Values-of-Statistical-Life (VOSL) from SC data consisting of three attributes: cost, time and safety. NP4VTT implements a recent adaption of this model for balanced panel data, proposed by van Cranenburgh and Kouwenhoven (2021, appendix B) to estimate the VTT from two-alternative-two-attribute data. This nonparametric model is built on two assumptions. First, each respondent has a VTT that is constant across the presented choice tasks. Second, each choice that is made by a respondent is subject to a given probability q of being inconsistent with the respondent's underlying VTT. Therefore, the probability of observing a series of T choices $Y_n = \{y_{n1}, y_{n2}, \ldots, y_{nT}\}$ for a respondent n is given by equation (5):

$$P(Y_n|v) = q^{\tau_n}(1-q)^{T-\tau_n}, \tag{5b}$$

where $\tau_n(v)$ denotes the number of choices that is consistent with when the respondent's VTT equals v. The unconditional probability for observing a respondent n to making the series of choices Y_n is computed by integrating over . Practically, we discretise the VTT space in B evenly spaced bins, and we integrate numerically, as shown in equation (6):

$$P(Y_n) = \sum_{b=1}^{B} f(v_b) q^{\tau_n}(1-q)^{T-\tau_n}, \tag{6}$$

where $f(v_b)$ denotes the probability density function of the VTT at v_b. Estimating this model involves estimating one density parameter per bin, plus the probability for an inconsistent choice q. An estimate of the CDF is obtained by computing the cumulative sum of the density parameters, previously converted to probability space.

2.5. ANN-based VTT model

van Cranenburgh and Kouwenhoven (2021) propose an ANN-based approach to uncover the VTT distribution (henceforth referred to as ANN-based VTT model), which is implemented in NP4VTT for balanced panel data. This ANN-based VTT model builds on the notion that the VTT can be inferred through finding the BVTT that makes the respondent indifferent between the slow and cheap and fast and expensive alternatives. If a respondent is indifferent between the slow and cheap and the fast and expensive alternative, then the BVTT must equal the VTT of the respondent. This approach takes the following steps to recover the BVTT that makes a respondent indifferent.

First, the data are reorganised. For each respondent, a randomly selected choice observation is singled out as the dependent choice task. The remaining $T-1$ choice observations, including the associated choices, are used as independent variables. Second, an ANN is trained to predict the choice in the dependent choice task. In other words, the ANN is trained to learn a mapping $g(\cdot)$, based on the choices and BVTTs of the $T-1$ choice tasks and the BVTT of the dependent choice task. Equation (7) details such mapping, where $BVTT_{-r}$ and y_{-r} denote respectively the BVTTs and choices of the $T-1$ choice observations that serve as independent variables, and $BVTT_r$ denote the BVTT of the dependent choice task:

$$P_r = g(BVTT_{-r}, y_{-r}, BVTT_r). \tag{7}$$

Third, after training the ANN, it is used to simulate the effect of $BVTT_r$ on the choice probablility. The VTT for each respondent is recovered by finding the $BVTT_r$ that yields $P_r = 0.5$. Finally, the recovered VTTs of all respondents in the sample are taken together to produce the VTT distribution.

2.6. Logistic regression-based VTT model

This model is proposed by van Cranenburgh and Kouwenhoven (2021) as a variation of their ANN-based VTT model, and implemented in NP4VTT for balanced panel data. It is motivated by the observation that the ANN-based VTT model cannot be readily interpreted because of the opaqueness of the ANN. The main idea of this model is to create a transparent counterpart, by replacing the ANN with a Logistic regression. In other words, in this model, the mapping $g(\cdot)$ of equation (7) is learned by a simple linear regressor.

Doing so will decrease the model's flexibility, leading to deteriorated model performance (e.g., lower log-likelihood), but will increase the interpretability of the model. Because of its simple linear structure, van Cranenburgh and Kouwenhoven (2021) show that the regression problem reduces to the form described in equations (8a)–(8b):

$$P_r = \frac{1}{1+\exp(-V_n)}, \tag{8a}$$

$$V_n = \delta + \beta_{y,BVTT} \sum_{t=1}^{T-1} y_{nt}^{FE} \cdot BVTT_{nt} + \beta_{BVTT} \cdot BVTT_{nr}. \tag{8b}$$

In equation (8b), δ is an intercept and y_{nt}^{FE} is an indicator equal to 1 if respondent n chooses the fast and expensive alternative in choice task t. $\beta_{y,BVTT}$ is interpretable as the marginal effect of a choice for the fast and expensive alternative in a choice task with $BVTT_{nt}$. $\beta_{y,BVTT}$ is expected to be positive. β_{BVTT} captures the marginal effect of the $BVTT_r$ of the dependent choice task. As a higher $BVTT_r$ lowers the probability of choosing the fast and expensive alternative. Hence, we expect β_{BVTT} to be negative.

2.7. Random Valuation

While the Random Valuation (RV) model is not a nonparametric model, it is nonetheless added to this software package to compute a benchmark VTT. The RV model is a parametric model that allows to compute the mean VTT in cross-sectional two-alternative-two-attribute data, first proposed by Cameron and James (1987) and described in Ojeda-Cabral and Chorus (2016) and Ojeda-Cabral et al. (2016). The RV model assumes that the choice task is a "time market" in which the price is the BVTT of equation (1). Then, a respondent chooses the slow and cheap alternative if their VTT is lower than the BVTT of the choice task, otherwise, they choose the fast and expensive alternative. Adding an additive Extreme Value stochastic term, the choice probabilities of the RV model are represented by equation (9):

$$y = 1\{VTT < BVTT + \varepsilon\}. \tag{9}$$

Following Ojeda-Cabral et al. (2016), the utility of each alternative in the RV model is parametrized as in equations (10a) and (10b):

$$U_1 = \mu \cdot BVTT + \varepsilon_1, \tag{10a}$$

$$U_2 = \mu \cdot VTT + \varepsilon_2, \tag{10b}$$

where μ is a scale parameter. Since ε_1 and ε_2 are Extreme Value stochastic terms, the RV choice probabilities collapse to a binary logit model. The mean VTT is directly obtained from the estimation results since it enters to the logit model as a parameter to be estimated.

3. The NP4VTT software

NP4VTT is provided as a Python 3 package. Users can install NP4VTT from the Python Package Index (PyPi) [4] using the regular procedure to instal packages (i.e., python -m pip install py-np4vtt in the command line interface). NP4VTT requires Python version 3.8 or higher and depends on the following packages:

- Pandas version 1.3.1 or higher,
- SciPy version 1.7.1 or higher,
- Scikit-learn version 1.0.2 or higher,
- Matplotlib version 3.5.1 or higher, and
- Numdifftools version 0.9.40 or higher.

NP4VTT is developed as open-source software. This means that users have complete access to the source code of NP4VTT. They can download the source code and suggest changes and additions. The source code of NP4VTT is stored in a Git repository and can be accessed by anyone. While we, as maintainers of NP4VTT, aim to provide a reliable tool for research and education, this software comes with no warranty. Thus, neither the authors nor the Delft University of Technology are liable for any consequences from the use of the software, waiving that responsibility to the users.

Fig. 1 details the structure of the NP4VTT package. The main module (py-np4vtt) contains six submodules that contain each model, plus three submodules dedicated to arranging variables (data_format), creating the necessary arrays from a Pandas data frame (data_import) and utilitarian functions (utils). The submodules that contain each model are:

[4] The package is available in https://pypi.org/project/py-np4vtt/.

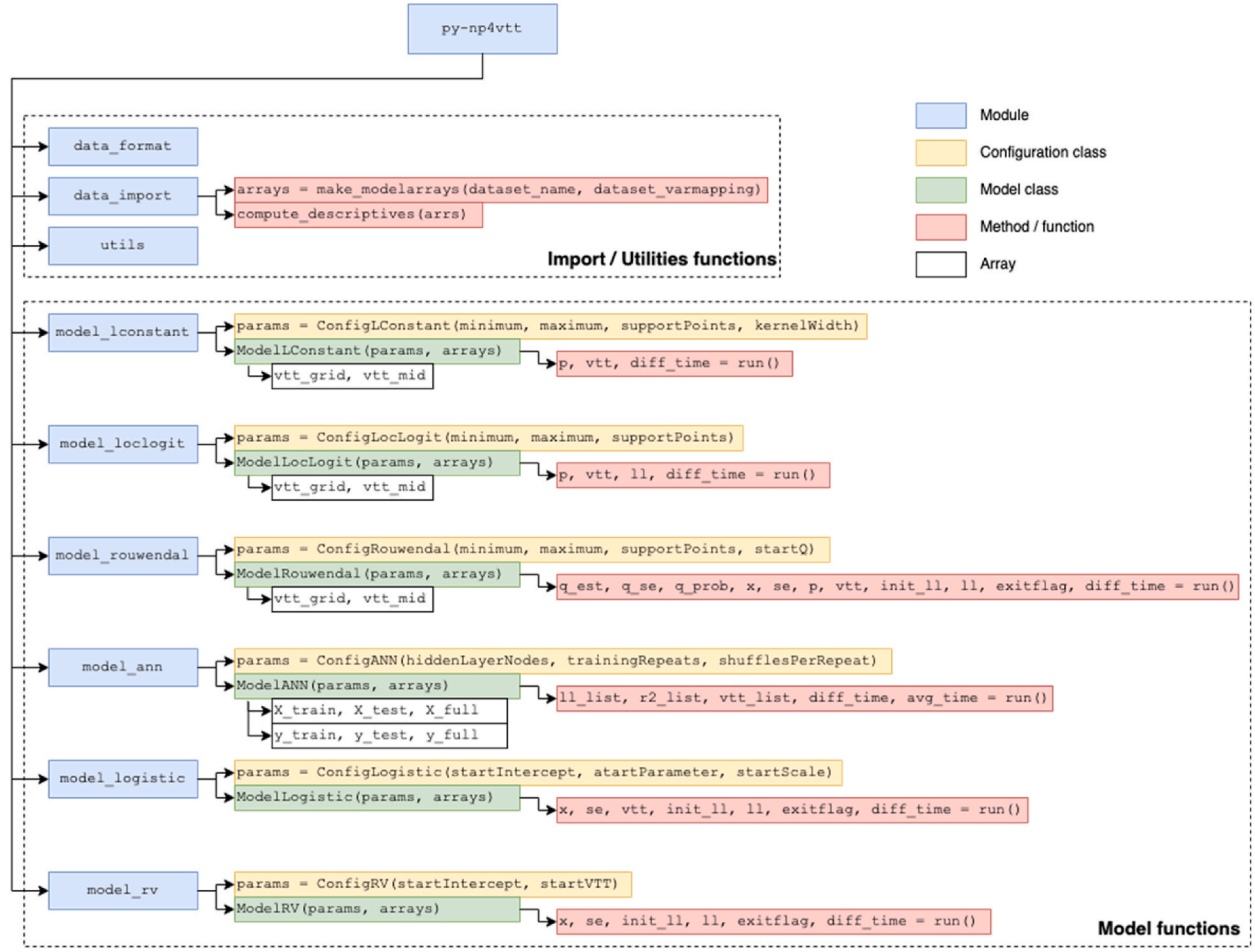

Fig. 1. Structure of NP4VTT.

1. Local constant (see section 2.2): model_lconstant,
2. Local logit (see section 2.3): model_loclogit,
3. Rouwendal model (see section 2.4): model_rouwendal,
4. ANN-based VTT model (see section 2.5): model_ann,
5. Logistic regression (see section 2.6): model_logistic, and
6. Random Valuation model (see section 2.7): model_rv.

Each model submodule contains a configuration class and a model class. The purpose of a configuration class is to receive the specific parameters of the correspondent model, perform integrity checks (e.g., the number of support points of the Local logit model must be a positive integer), and pass that information to the correspondent model class. Model classes contain the routines and methods to prepare specific arrays and estimate their correspondent model (run). After estimating a model (i.e., using the method run), the model class stores specific output to compute the VTT distribution based on the configuration parameters and the data. Table 2 describes the configuration parameters and outcomes of each model. A complete description of the classes and functions included in NP4VTT is provided in the reference manual as supplementary material to this paper.

Irrespective of the model, the procedure to use NP4VTT consists of three stages. Fig. 2 depicts these three stages. In the first stage, the user creates the model arrays using the method make_modelarrays, which takes a Pandas data frame that contains the dataset and a dictionary that maps each necessary array with the variable names in the dataset as inputs. In the second stage, the user provides configuration parameters through a configuration class and creates the model object. For example, suppose the user wants to estimate a Local logit model. In that case, the user must provide the minimum and maximum VTT values, and the number of support points through the ConfigLocLogit configuration class. Then, the user creates a model object using the ModelLocLogit model class, the configuration class and the arrays object created with make_modelarrays. Some methods store specific arrays once the model object is created that can be accessed by the user. For instance, the ModelLocLogit object contains the VTT grid array that can be accessed using the "dot" notation (i.e., ModelLocLogit.vtt_grid) and the mid points of the VTT grid (i.e., ModelLocLogit.vtt_mid). In the third stage, the user estimates the model using the method run. The output of the method run corresponds to the specific outcomes of the estimated

Table 2
Input of configuration class and model class outputs.

Model and model submodule	Configuration class and inputs	Description of configuration inputs	Model class and output
Local constant: model_lconstant	ConfigLConstant: minimum [float] maximum [float] supportPoints [integer] kernelWidth [float]	The minimum, maximum and number of support points of the VTT grid in which the supportPoints-1] estimates of the CDF will be estimated, plus the kernel width of the nonparametric (Nadaraya-Watson) estimator.	ModelLConstant: At initialisation: vtt_grid: VTT grid of points vtt_mid: mid-points of the VTT grid After estimation: p: set of estimates of the CDF evaluated at each mid point of the VTT grid. vtt: set of VTTs of size NP, derived from p est_time: the estimation time in seconds.
Local logit: model_loclogit	ConfigLocLogit: minimum [float] maximum [float] supportPoints [integer]	The minimum, maximum and number of support points of the VTT grid in which the [supportPoints-1] estimates of the CDF will be estimated.	ModelLocLogit: At initialisation: vtt_grid: VTT grid of points vtt_mid: mid-points of the VTT grid After estimation: p: set of estimates of the CDF evaluated at each interval of the VTT grid. vtt: set of VTTs of size NP, derived from p ll: log-likelihood value at the optimum est_time: the estimation time in seconds.
Rouwendal model: model_rouwendal	ConfigRouwendal: minimum [float] maximum [float] supportPoints [integer] startQ [float]	The minimum, maximum and number of support points of the VTT grid in which the [supportPoints −1] estimates of the CDF will be estimated, in addition to the starting value (startQ) of the probability of consistent choice	ModelRouwendal: At initialisation: vtt_grid: VTT grid of points vtt_mid: mid-points of the VTT grid After estimation: q_est: raw estimate of the probability of consistent choice q_se: Std. Error of q_est q_prob: (logit) probability of q_est x: density parameter estimates se: standard errors of x p: set of estimates of the CDF evaluated at each interval of the VTT grid vtt: set of VTTs of size NP, derived from p init_ll: value of log-likelihood function at starting values ll: value of log-likelihood function at the optimum exitflag: convergence result (if zero, estimation was successful) est_time: the estimation time in seconds.
ANN-based model: model_ann	ConfigANN: hiddenLayerNodes [list] trainingRepeats [int] shufflesPerRepeat [int] seed [hashable]	The topology of the neural network (hiddenLayerNodes), the number of training repeats and the number of shuffles per repeat	ModelANN: At initialisation: X_train, X_test, X_full: input data (BVTT and T-1 choices) for training, testing and full sample y_train, y_test, y_full: output data (T choice) for training, testing and full sample After estimation: ll_list: array of log-likelihoods at convergence per training repeat r2_list: array of rho-square values per training repeat vtt_list: array with VTTs per respondent per training repeat est_time: the estimation time in seconds. avg_time: average estimation time per repetition.
Logistic regression: model_logistic	ConfigLogistic: startIntercept [float] startParameter [float] startScale [float] maxIterations [int] seed [hashable]	The starting values of the parameters to be estimated in the Logistic regression model, and the number of iterations of the optimization routine.	ModelLogistic: At initialisation: (none) After estimation: x: model estimates (scale, intercept and parameter se: standard errors

(*continued on next page*)

Table 2 (*continued*)

Model and model submodule	Configuration class and inputs	Description of configuration inputs	Model class and output
Random Valuation: model_rv	ConfigRV: mleScale [float] maxIterations [int] mleVTT [float]	The starting values of the parameters of the RV model, and the number of iterations of the optimization routine.	vtt: VTT per respondent, based on x. init_ll: log-likelihood at starting values ll: log-likelihood value at convergence exitflag: convergence result (if zero, estimation was successful) est_time: the estimation time in seconds. ModelRV: At initialisation: (None) After estimation: x: model estimates (scale and VTT) se: standard errors init_ll: log-likelihood at starting values ll: log-likelihood value at convergence exitflag: convergence result (if zero, estimation was successful) est_time: the estimation time in seconds.

model. For example, in a Local logit model, the estimation routine stores the set of estimates of the CDF points at each interval of the VTT grid, the estimated VTT for each respondent of the sample and the final log-likelihood value.

4. Demonstration of NP4VTT

4.1. Demonstration data

We demonstrate the use of NP4VTT using the Norway 2009 VTT data (Ramjerdi et al., 2010). This is a balanced panel dataset that contains 5832 respondents, each having made $T = 9$ choice tasks. Each respondent was presented with a two-alternative choice task, characterised by two attributes: travel time in minutes and travel cost in Norwegian crowns.

Table 3 shows one example of a choice tack from this SC experiment. To ease interpretation, we converted the currency from Norwegian krone to Euro, using an exchange rate of 9 NOK = 1 EUR. The minimum and maximum values of the BVTT are 0.67 EUR/hr. and 113.56 EUR/hr., respectively. The mean chosen BVTT is 10.30 EUR/hr.

These data comprise six variables:

- RespID: A unique identifier that maps every single respondent of the dataset,
- Chosen: The respondent's chosen alternative. It can take values 1 or 2,
- CostL: Travel costs of alternative 1 [NOK],
- CostR: Travel costs of alternative 2 [NOK],

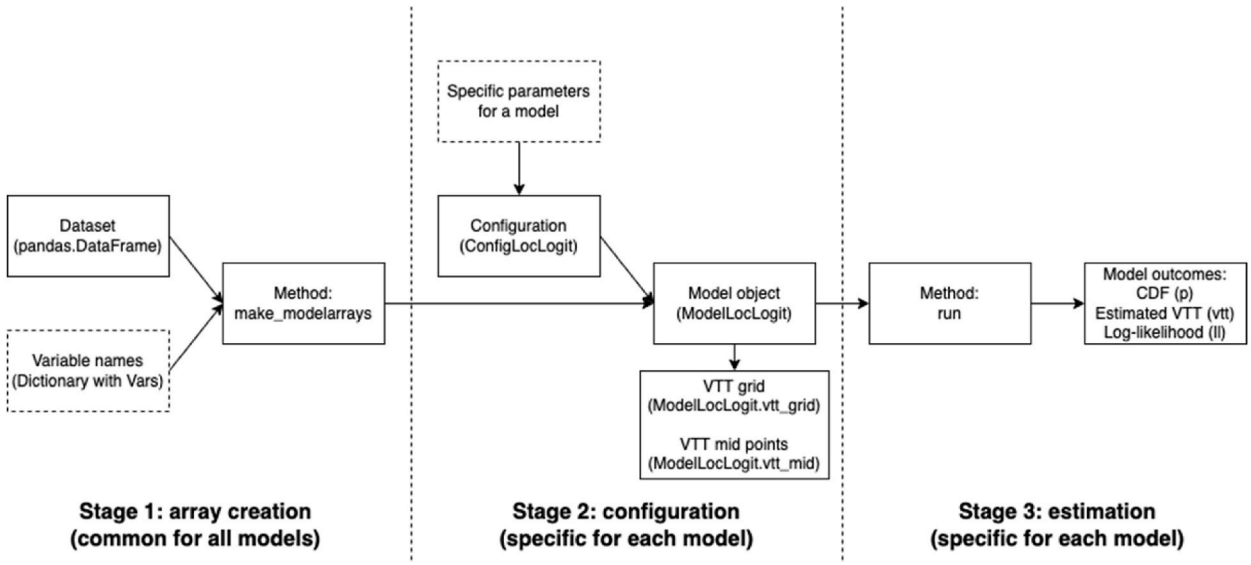

Fig. 2. Usage procedure of NP4VTT (using a Local logit as an example).

Table 3
Example choice task of the Norwegian 2009 VTT data.

	Alternative 1	Alternative 2
Travel time (minutes)	15	12
Travel cost (Euro)[a]	8	10
YOUR CHOICE	**O**	**O**

[a] Norwegian krone are converted to euros.

- TimeL: Travel time of alternative 1 [minutes], and
- TimeR: Travel time of alternative 2 [minutes].

4.2. Loading data, array creation and descriptive statistics

First, the user must load the dataset and generate the necessary arrays. Box 1 shows a code snippet that exemplifies this process in NP4VTT.

After the user loads the required modules (i.e., Pandas and the sub-modules of NP4VTT for array creation), the data are read from a CSV-file and stored as a Pandas data frame named df.

To map the required variables with the names that appear in the dataset, the user must a dictionary object — named in this example as columnarrays — with the variable names contained in the data, for the following keys:

Box 1
Code to load the data, create arrays and compute descriptive statistics in NP4VTT

```
# Load modules
import pandas as pd
from py_np4vtt.data_format import Vars
from py_np4vtt.data_import import make_modelarrays, compute_descriptives

# Load dataset
df = pd.read_table('../data/Norway2009VTT_demodata.txt')

# Convert travel cost to EUR
NOK2euro_exchange_rate = 9
df[['CostL','CostR']] = df[['CostL','CostR']] .div(NOK2euro_exchange_rate)

# Convert travel time to hours
df[['TimeL','TimeR']] = df[['TimeL','TimeR']] .div(60)

# Create dictionary that maps the required variables with the variables of
the dataset
columnarrays = {
    Vars.Id: 'RespID',
    Vars.ChosenAlt: 'Chosen',
    Vars.Cost1: 'CostL',
    Vars.Cost2: 'CostR',
    Vars.Time1: 'TimeL',
    Vars.Time2: 'TimeR',
}

# Create the necessary arrays
model_arrays = make_modelarrays(df, columnarrays)

# Compute descriptives
descriptives = compute_descriptives(model_arrays)
```

- Vars.Id: The unique identifier variable,
- Vars.ChosenAlt: Choice indicator,
- Vars.Cost1 and Vars.Cost2: Cost variables, and
- Vars.Time1 and Vars.Time2: Time variables.

Finally, the user creates the necessary arrays using the make_modelarrays function. The arrays are stored in the model_arrays object. The function make_modelarrays takes the Pandas data frame and the dictionary that maps the variables as inputs. This function outputs a list with the following elements:

• NP:	An integer describing the number of respondents,
• T:	An integer describing the number of choice tasks per respondent,
• ID:	A 1-dimensional NumPy array of dimension *NP* that contains the unique identifiers of each respondent,
• BVTT:	A 2-dimensional NumPy array of dimension ($NP \times T$), that contains the BVTT computed using Equation (1) from the observed travel costs and travel time, for each choice task and each respondent,
• Choice:	A 2-dimensional NumPy array of dimension ($NP \times T$), in which each cell takes value 1 if the respondent chose the expensive and fast alternative, and zero otherwise, and
• Accepts:	A 1-dimensional NumPy array of dimension *NP* that contains the number of times a respondent accepted the fast and expensive alternative.

Finally, the function compute_descriptives takes model_arrays as input and produces a set of descriptive statistics:

- The number of respondents (NP) and number of choice tasks (T),
- Number of non-traders of the expensive-but-fast and cheap-but-slow alternatives,
- Mean of the chosen BVTT, and
- Minimum and maximum values of the BVTT in the data.

4.3. Estimation

After the necessary arrays are created, we estimate the nonparametric models. Each model included in NP4VTT takes the arrays created with the function model_arrays plus one object that stores the specific configuration parameters of the model. The model is initialised with this information. Estimation produces a set of arrays that allow the user to describe the VTT distribution, as in Table 2.

In the following subsections, we demonstrate NP4VTT by showing how to invoke the following four models: Local constant, Local logit, Rouwendal and ANN-based VTT.

4.3.1. Estimating a Local constant model

Estimating a Local constant model involves configuring the specific parameters, creating the model object based on the specific configuration and arrays and executing the estimation routine. Box 2 details the code to configure and estimate the Local constant model. In this example, we define a VTT grid from 0 to 100 with 21 support points and a kernel width of 2 euros. The Local constant model will get an estimate of the CDF at each mid point of the VTT grid.

The object ConfigLConstant performs integrity checks to avoid invalid entries for the configuration parameters. Then, the user creates the Local constant object using the object ModelLConstant that receives the configuration object and the necessary arrays. The model is stored in the object called lc. The Local constant model object creates an array called vtt_grid, corresponding to the VTT grid as specified in the configuration parameters, as well as its corresponding mid points (vtt_mid). The VTT mid points are computed to allow the user to describe the VTT distribution in plots as the Local constant model computes the probabilities pointwise, thus complicating the interpretation of histograms. The VTT grid is stored in the Local constant model object and can be accessed using the "dot" notation (i.e., lc.vtt_grid).

To start the estimation routine, the user executes the method "run" (i.e., lc.run()). After completion, the estimation routine returns the following objects:

- p: set of estimates of the CDF evaluated at the mid point of the VTT grid,
- vtt: set of VTTs of size NP, derived from p, and
- est_time: the estimation time in seconds.

The user can use the model outputs to describe the VTT distribution. In section 4.4, we demonstrate a visualisation of the VTT distribution.

4.3.2. Estimating a Local logit model

As with all models of NP4VTT, estimating a Local logit model involves configuring the specific parameters, creating the model object, and executing the estimation routine. Box 3 details the code to configure and estimate a Local logit model. In this example, we define a VTT grid between 0 and 100, with 21 support points. The Local logit model will estimate the CDF on equally sized intervals with a length of 5.

First, the user creates the configuration object ConfigLocLogit that verifies that the parameters of the VTT grid are valid. Then, the

Box 2
Configuration and estimation of a Local constant model

```
# Configure a Local constant model
from py_np4vtt.model_lconstant import ModelLConstant, ConfigLConstant
config = ConfigLConstant(
    minimum=0,
    maximum=100,
    supportPoints=21,
    kernelWidth = 2)

# Create Local constant model object
lc = ModelLConstant(config, model_arrays)

# The VTT grid and mid points can be accessed using the 'dot' notation
vtt_grid = lc.vtt_grid
vtt_mid = lc.vtt_mid

# Estimate the Local constant model
p, vtt, est_time = lc.run()
```

Local logit model object is created using the configuration and the arrays using the object ModelLocLogit. The Local logit model object creates the VTT grid (vtt_grid) as specified by the configuration parameters and its corresponding mid points (vtt_mid). The VTT mid points are computed to allow the user to describe the VTT distribution in plots, as the Local logit model estimates the CDF at intervals of the VTT grid. The VTT grid and mid points can be accessed using the "dot" notation.

To estimate the Local logit model, the user executes the "run" method. The outputs of the estimated Local logit model are:

- p: set of estimates of the CDF evaluated at each interval of the VTT grid,
- vtt: set of VTTs of size NP, derived from p,
- ll: the log-likelihood function at the optimum of the estimation process, and
- est_time: the estimation time in seconds.

Box 3
Configuration and estimation of a Local logit model

```
from py_np4vtt.model_loclogit import ModelLocLogit, ConfigLocLogit
config = ConfigLocLogit(
    minimum=0,
    maximum=100,
    supportPoints=21)

# Create the Local logit model object
loclogit = ModelLocLogit(config, model_arrays)

# The created VTT grid and midpoints can be accessed using the 'dot' notation
vtt_grid = lc.vtt_grid
vtt_mid = loclogit.vtt_mid

# Estimate the Local logit model
p, vtt, ll, est_time = loclogit.run()
```

The user can visually describe the VTT distribution using the estimates of the CDF and the VTT mid points or directly use the estimated VTT. Section 4.4. illustrates the VTT distribution for this specific example.

4.3.3. Estimating the rouwendal model

Box 4 details the code to configure and estimate the Rouwendal model. The configuration of the Rouwendal model parameters is done with the ConfigRouwendal object. We define a VTT grid from 0 to 100 with 21 support points. Hence, the Rouwendal model will estimate the CDF on equally sized intervals of 5 units. Additionally, we set the starting value of the probability of consistent choice in $q = 0.9$.

After creating the configuration object, the user creates the Rouwendal model object ModelRouwendal using the configuration object and the arrays. The model object creates the VTT grid (vtt_grid) as specified by the configuration parameters and its corresponding mid points (vtt_mid), as the Rouwendal model estimates the CDF at intervals of the VTT grid. The VTT grid and mid points can be accessed using the "dot" notation.

The estimation of the Rouwendal model is done using the run method. The estimated Rouwendal model returns the following outputs:

- q_est: raw estimate of the probability of consistent choice,
- q_se: standard error of q_est,
- q_prob: (logit) probability of q_est,
- x: density parameter estimates,
- se: standard errors of x,
- p: set of estimates of the CDF evaluated at each interval of the VTT grid,
- vtt: set of VTTs of size NP derived from p,
- init_ll: Value of log-likelihood function at starting values. Starting values of density parameters are equal to zero,
- ll: Value of log-likelihood function in the optimum,
- exitflag: Convergence result. If exitflag = 0, the optimisation succeeded. Otherwise, check the configuration parameters, and
- est_time: the estimation time in seconds.

The user can describe the VTT distribution with the set of estimates of the CDF at each VTT mid point or directly plot a histogram of the estimated VTT. Section 4.4 illustrates the VTT distribution for the Rouwendal model compared with the other models employed in this example.

Box 4
Configuration and estimation of the Rouwendal model

```
# Configure Rouwendal model
from py_np4vtt.model_rouwendal import ConfigRouwendal, ModelRouwendal
config = ConfigRouwendal(
    minimum= 0,
    maximum= 100,
    supportPoints= 21,
    startQ= 0.95)

# Create Rouwendal model object
rouwendal = ModelRouwendal(config, model_arrays)

# VTT grid and VTT mid points can be accessed using the 'dot' notation
vtt_grid = rouwendal.vtt_grid
vtt_grid

vtt_mid = rouwendal.vtt_mid
vtt_mid

# Estimate the Rouwendal model
q_est, q_se, q_prob, x, se, p, vtt, init_ll, ll, exitflag, est_time =
rouwendal.run()
```

Box 5
Configuration and estimation of an ANN-based model

```
# Configure the ANN-based model
from py_np4vtt.model_ann import ModelANN, ConfigANN
config = ConfigANN(
    hiddenLayerNodes=[10, 10],
    trainingRepeats= 5,
    shufflesPerRepeat= 50,
    seed = None)

# Create the ANN-based model
ann = ModelANN(config, model_arrays)

# Access to data (e.g., training input data) using the 'dot' notation
X_train = ann.X_train

# Estimate the ANN-based model
ll_list, r2_list, vtt_list, est_time, avg_time = ann.run()
```

4.3.4. Estimating an ANN-based model

The final example is the configuration and estimation process of an ANN-based model. Box 5 shows the code for configuring an ANN-based model with our example. In this example, we specify an ANN with two hidden layers and ten hidden nodes per layer. Additionally, we define five estimation repeats and 50 random shuffles of the estimation data per repeat. Optionally, the user can set the random seed for being able to replicate results purposes by adding the parameter seed in the configuration object.

After creating the configuration object, the user creates the ANN-based model object using ModelANN, the configuration object and the arrays. The model object creates the input and output data arrays for training (estimation) and testing. The user can access these objects using the "dot" notation.

To estimate the ANN-based model, the user executes the run method. The estimated ANN-based model returns the following outputs:

- ll_list: array of log-likelihoods at convergence per training repeat,
- r2_list: array of Rho-squared values per training repeat,
- vtt_list: array of VTTs per respondent per training repeat,
- est_time: the estimation time in seconds, and
- avg_time: the average estimation time per repetition.

The user can use the outputs to depict the VTT distribution. In section 4.4, we show the results of the VTT distribution for the ANN-based model, together with the other models.

4.4. Recovering and visualising the VTT distribution

Table 4 summarises the estimation time of each nonparametric model. The estimations were performed in a MacBook Pro 2019 laptop with a 4-core Intel Core i5 CPU with 2.4GHz and 8GB of RAM. Similar results were obtained with a Linux machine with similar characteristics. The local constant and local Logit models have an estimation time of less than 1 second. In contrast, the Rouwendal' model took approximately 1 min (68.99 seconds) to be estimated, and the ANN-based VTT model took almost 4 minutes (224 seconds) to be trained. The larger estimation time of the Rouwendal's model is explained by the computation of the standard errors. For the ANN-based model, the increased training time is explained because it relies on repeated training, while on average, each repetition takes around 45 seconds.

Table 4
Estimation time per model.

Model	Estimation time (in seconds)
Local constant	0.09 secs.
Local Logit	0.71 secs.
Rouwendal's model	68.99 secs.
ANN-based VTT model	Total: 223.91 secs. - average/repetition: 44.78 secs.

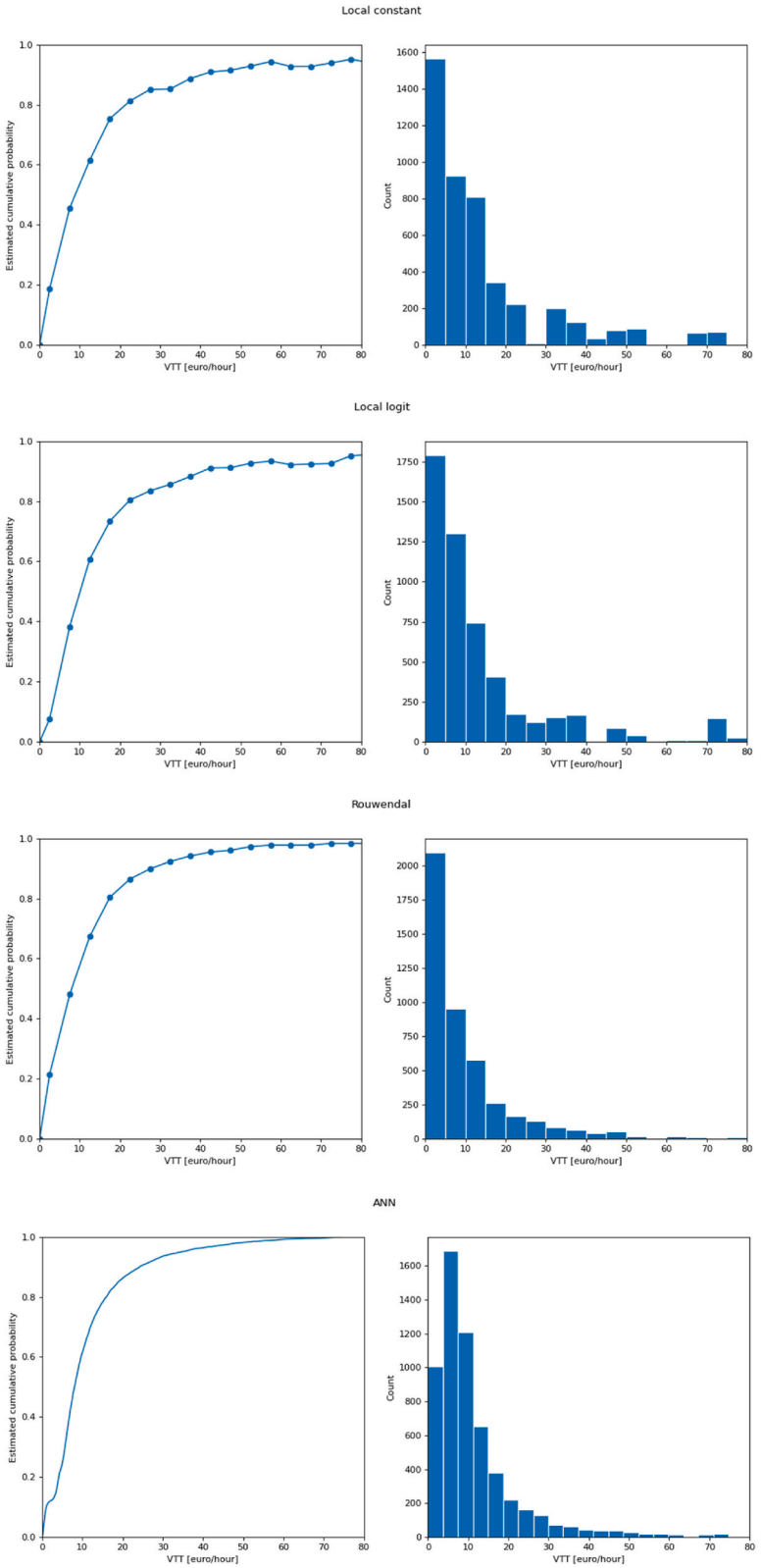

Fig. 3. Recovered VTT distributions using local constant (top), Local logit (second), Rouwendal (third) and ANN-based VTT model (bottom).

Fig. 3 presents empirical CDFs (left) and histograms (right) obtained from the four models we used to demonstrate NP4VTT. The upper row shows the results of the Local constant model; the second row shows the Local logit model; the third row shows the Rouwendal model; the lower row shows the ANN-based VTT model. In these plots, the *x*-axis shows the VTT [€/hr.]; the *y*-axis depicts the estimate of the CDF (left) and the count (right).

Being able to produce and compare the results of multiple nonparametric models provides a profound understanding of the VTT distribution. But, it is important to interpret the results considering the sort of nonparametric model. For instance, we see that the Rouwendal predicts a considerably thinner tail than the Local constant and Local logit models. In the Rouwendal model, the thickness of the tail is a result of the assumption that respondents have a fixed probability of making decisions that are inconsistent with their underlying VTT. This probability is not a function of the BVTT. In this model, the thickness on the very end of the tail is thus not the result of observations for the fast and expensive alternative at the very end of the tail. Furthermore, the estimation process of the ANN-based model is enriched by repeated estimations, data expansion and random shuffling. In contrast, the Local constant model and Local logit model do not assume any underlying process regarding inconsistent choices, they do not consider any panel structure, and they do not enrich the data, strictly limiting themselves to get an estimate of the CDF in a given set of VTT points.

5. Conclusion

This paper introduces NP4VTT, a new Python package to estimate nonparametric models for inference of the VTT distribution. This package includes the following nonparametric models: Local constant, Local logit, Rouwendal, ANN-based VTT model, and Logistic regression. The modular set-up of NP4VTT allows to incorporate other nonparametric models in the future. We hope this package lowers the burden for researchers of using these powerful models to analyse the shape of the VTT distribution.

From a substantive point of view, our results highlight the sensitivity of the recovered shape of the VTT distribution. While the advantage of nonparametric models is that the analyst does not need to assume the shape of the distribution, they do unequivocally produce the same shape of the VTT. The differences between the recovered distributions of the nonparametric models have various causes, such as whether the model accounts for a panel structure and how the model deals with stochasticity. We believe future research must dig further into the robustness of the VTT shape recovery, using both parametric and nonparametric approaches. We hope our Python package facilitates in this effort and encourages the development of new nonparametric models.

Authors' statement

José Ignacio Hernández: Writing – Original draft; Writing – Review & Editing; Software – Programming, software development; Project administration. **Sander van Cranenburgh:** Conceptualisation; Methodology; Writing – Original draft; Writing – Review & Editing, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

We would like to thank Askill H. Halse for sharing the Norwegian 2009 VTT data with us and allowing us to make these data publicly available for demonstrating this software. José Ignacio Hernandez would like to thank the Netherlands Organisation for Scientific Research (NWO) for funding his PhD (NWO Responsible Innovation grant - 313-99-333) and João Pizani Flor for his valuable support and knowledge during the development of this software.

Appendix A. Supplementary data

Supplementary data to this article can be found online at https://doi.org/10.1016/j.jocm.2023.100427.

References

Bierlaire, M., 2020. A Short Introduction to PandasBiogeme. École Polytechnique Fédérale de Lausanne.

Börjesson, M., Eliasson, J., 2014. Experiences from the Swedish value of time study. Transport. Res. Pol. Pract. 59, 144–158. https://doi.org/10.1016/j.tra.2013.10.022.

Cameron, T.A., James, M.D., 1987. Efficient estimation methods for 'closed-ended' contingent valuation surveys. Rev. Econ. Stat. 69 (2), 269–276. https://doi.org/10.2307/1927234.

Carson, R.T., Mitchell, R.C., Hanemann, M., Kopp, R.J., Presser, S., Ruud, P.A., 2003. Contingent valuation and lost passive use: damages from the Exxon Valdez oil spill. Environ. Resour. Econ. 25, 257–286. https://doi.org/10.1023/A:1024486702104.

Fan, J., Heckman, N.E., Wand, M.P., 1995. Local polynomial kernel regression for generalized linear models and quasi-likelihood functions. J. Am. Stat. Assoc. 90 (429), 141–150. https://doi.org/10.1080/01621459.1995.10476496.

Fosgerau, M., 2006. Investigating the distribution of the value of travel time savings. Transp. Res. Part B Methodol. 40 (8), 688–707. https://doi.org/10.1016/j.trb.2005.09.007.
Fosgerau, M., 2007. Using nonparametrics to specify a model to measure the value of travel time. Transport. Res. Pol. Pract. 41 (9), 842–856. https://doi.org/10.1016/j.tra.2006.10.004.
Fosgerau, M., Hjorth, K., Lyk-Jensen, S.V., 2007. The Danish Value of Time Study. Kgs. Lyngby, Denmark.
Hess, S., Daly, A., Börjesson, M., 2020. A critical appraisal of the use of simple time-money trade-offs for appraisal value of travel time measures. Transportation 47 (3), 1541–1570. https://doi.org/10.1007/s11116-020-10097-w.
Hess, S., Palma, D., 2019. Apollo: a flexible, powerful and customisable freeware package for choice model estimation and application. Journal of Choice Modelling 32, 100170. https://doi.org/10.1016/j.jocm.2019.100170.
Kouwenhoven, M., de Jong, G.C., Koster, P., van den Berg, V.A.C., Verhoef, E.T., Bates, J., Warffemius, P.M.J., 2014. New values of time and reliability in passenger transport in The Netherlands. Res. Transport. Econ. 47, 37–49. https://doi.org/10.1016/j.retrec.2014.09.017.
Ojeda-Cabral, M., Batley, R., Hess, S., 2016. The value of travel time: random utility versus random valuation. Transportmetrica: Transport. Sci. 12 (3), 230–248. https://doi.org/10.1080/23249935.2015.1125398.
Ojeda-Cabral, M., Chorus, C.G., 2016. Value of travel time changes: theory and simulation to understand the connection between Random Valuation and Random Utility methods. Transport Pol. 48, 139–145. https://doi.org/10.1016/j.tranpol.2016.03.006.
Ramjerdi, F., Flügel, S., Samstad, H., Killi, M., 2010. Value of Time, Safety and Environment in Passenger Transport–Time. TØI Report B, p. 1053.
Rouwendal, J., de Blaeij, A., Rietveld, P., Verhoef, E., 2010. The information content of a stated choice experiment: a new method and its application to the value of a statistical life. Transp. Res. Part B Methodol. 44 (1), 136–151. https://doi.org/10.1016/j.trb.2009.04.006.
Small, K.A., 2012. Valuation of travel time. Economics of Transportation 1 (1), 2–14. https://doi.org/10.1016/j.ecotra.2012.09.002.
StataCorp, L.P., 2005. Stata Base Reference Manual. StataCorp LLC, College Station.
van Cranenburgh, S., Kouwenhoven, M., 2021. An artificial neural network based method to uncover the value-of-travel-time distribution. Transportation 48 (5), 2545–2583. https://doi.org/10.1007/s11116-020-10139-3.