A new way of commuting: Understanding consumer preferences in the emerging shared micromobility market

December 4 - 2025

[redacted]

Aiden de Haan [redacted]

Andreas Hotvedt [redacted]

Hy-Cang Lu [redacted]

Tobias Navntoft [redacted]

Introduction and selection of a product/service

Shared mobility services that offer one-time rentals of various vehicles, such as regular bikes, E-Mopeds, and E-bikes, have become increasingly common in urban travel across Europe. As European countries are favoring electric vehicles, and battery technology is developing and becoming cost-competitive, this novel form of rental mobility option has become a larger part of the product mix in the short-form transport market (micromobility market). This growth is expected to be sustained long-term as it meets previously unmet consumer needs, e.g., when private transport is unavailable, and public transport does not meet location needs or is unreliable. In the Netherlands, the largest shared mobility brands include Lime, CHECK, and Felyx, with offerings of e-mopeds and e-bikes, a sustainable alternative to the traditional gasoline mopeds that are common in the Dutch market (Montes, Geržinic, Veeneman, van Oort, & Hoogendoorn, 2023).

Shared micromobility is a novel and not well-understood market, this is because consumer needs vary significantly in different subregions within each city. For example, offering e-bikes to a region that is well-served by trams/metros might not be a profitable venture, compared to regions that have fewer alternatives. In addition to geographic factors, several demographic factors play a role, e.g., older/family-oriented neighborhoods are more likely to have cars or other private options readily available (Fu, van Lierop, & Ettema, 2024).

Studies show that 80% of individuals who typically use the shared-mobility options, such as Lime and CHECK, tend to be within the age range of 21-40, and approximately 71% hold some level of higher education, e.g., a bachelor's degree or an applied science degree (Reck & Axhausen, 2021). The demographic of this age group aligns more closely with the primary uses of shared micromobility, which is the daily commute to/from work or education (Reck & Axhausen, 2021; Montes, Geržinic, Veeneman, van Oort, & Hoogendoorn, 2023).

As a result, the challenge is understanding the segment is essential for ensuring profitable ventures for firms in the shared micromobility industry. For example, which aspects of shared micromobility do these users value, and to what extent? Is their demand mainly driven by convenience, sustainability, price, travel time, or some combination of these factors?

To investigate this, our study uses Discrete Choice Experiments (DCEs) to analyze the preferences of people primarily aged 18-40 in the Netherlands, as these individuals are likely to have interacted with shared micromobility options. The focus is to analyze how important price, travel time, and walking distance to the vehicle are, as well as how sustainable the vehicle they

use is, in terms of CO2 emissions over the vehicle's lifetime. Furthermore, we investigate whether these preferences vary based on the income of the respondents and their self-reported sustainability focus. Identifying whether there is heterogeneity in the target segment allows for more specific consumer targeting and a better match of product offering. This will provide insights for potential/existing shared micromobility firms when deciding their profit offering.

Attributes and levels: Describe how attributes and levels were derived

Price attribute: Price levels were derived from real shared mobility prices in the Netherlands. For example, based on CHECK's pricing of €0.35 per minute plus a €1 unlock fee, a 3 km trip typically costs €2.80-€4.50 depending on vehicle type and traffic. To reflect realistic variation across modalities and traffic, we selected a price range of €2-€5 (see Table 1).

Sustainability attribute: The sustainability attribute was defined as the lifecycle carbon footprint of each shared mobility option *relative to a standard gasoline moped*. We chose this relative framing for two main reasons. First, anchoring sustainability to a familiar reference point makes the survey more intuitive for respondents, as most users have a basic sense of how polluting a gas moped is and would not derive any meaning from raw emission values. Second, estimating absolute CO2 emissions for electric vehicles is complex as it depends on factors such as the electricity mix, battery and vehicle production, and supply chains. Thus, expressing sustainability in relative terms provides a more accurate way to measure how sustainable modalities are compared to each other (see Table 1).

Walking distance to the vehicle: Unlike public transport stops, shared mobility vehicles can be located anywhere within the service area, and availability fluctuates throughout the day. Users are often not in an ideal location when they start looking for a vehicle. For example, they may be on the edge of the service area or in a high-demand area with few vehicles nearby. For this reason, walking distance levels were chosen to represent realistic suboptimal conditions, where users may need to walk a small to medium-long distance to find an available vehicle. By looking at the distance of shared mobility hubs to various central stations, we found that distances range from 100-500 meters. These walking distance levels allow respondents to evaluate situations that closely resemble typical shared mobility use (See table 1).

Travel time: The travel time levels were chosen to reflect realistic variations for a 3 km urban trip. In the best-case scenario, such as taking a fast moped or a car on uncongested streets,

the trip can be completed in around 5 minutes. In contrast, under heavier traffic or with slower modalities, the same 3 km can take up to 15 minutes. By capturing this range between optimal and congested conditions, the selected travel time levels represent the variability users commonly face when using shared mobility services (See table 1).

Table 1 Attributes and Levels

Attribute	**Low**	**Medium**	**High**
Sustainability Level	0% CO2	25% CO2	50% CO2
Price	€2	€3.5	€5
Distance to Vehicle	100	250	500
Travel Time	5	10	15

Methodology

A Discrete Choice Experiment (DCE) is appropriate for this research because it allows us to estimate the relative importance that users assign to specific attributes of shared mobility options. By modelling choices as a function of these attributes, we can quantify how each one affects utility and user preferences. This provides a structured and empirical way to understand the role of sustainability relative to price, convenience, and speed in the adoption of shared mobility services.

Each choice task contained two unlabeled mobility options: Option A and Option B, defined solely by their attributes (see Figure 1). Respondents were required to make a forced choice between the two alternatives, with no opt-out option provided. This is because adding an opt-out choice would require a bigger sample size to obtain significant results, which, given the scope of the survey collection, was not feasible.

Furthermore, we chose a non-full factorial design because it allows us to independently estimate the effects of each attribute while avoiding the need to present respondents with every combination, preserving statistical power and interpretability with far fewer choice profiles. To obtain the combinations of the experimental design, we used a D-efficient design, which resulted in 24 different choice tasks[1]. 12 random tasks were shown to each participant to reduce the risk of cognitive fatigue respondents might occur if they had to answer 24 choice tasks. A fully

[1] Statistical identification of parameters: 8/(2-1)*3=24

randomized design was chosen instead of blocking because the expected sample size of responses, up to about 120 respondents, would not sufficiently mitigate between-block differences. Randomizing individual questions, therefore, reduces the risk of systematic bias and provides a more balanced representation of respondent characteristics across the survey.

Before the final design, a pilot study was first conducted to receive estimates on the priors of the various attributes that would then be used to make a more efficient design for the final survey. Due to a smaller sample size of the pilot study, only the size of some coefficients was used (e.g., price) while other coefficients were less significant but gave an indication of the direction of the prior (e.g., CO2 emissions). Furthermore, we collected feedback on the design of the survey and iterated our survey design to arrive at the following choice task visualization (Figure 1).

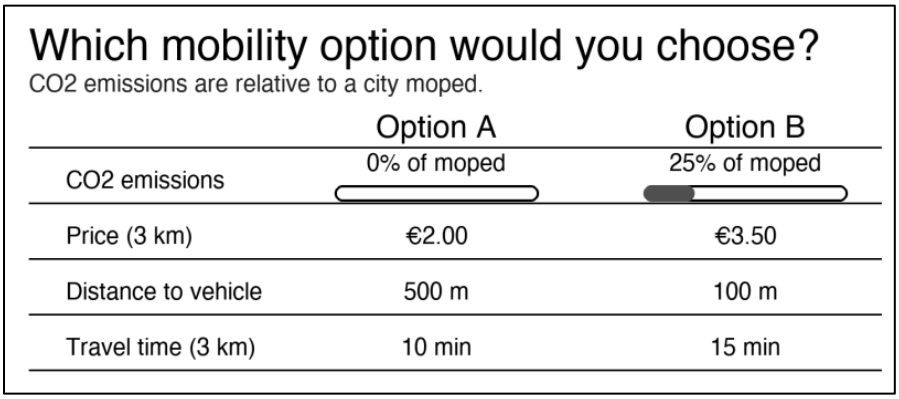


Figure 1 Example Choice Task Shown to Respondents

Before showing the choice tasks, the respondents were asked to imagine the following hypothetical scenario: *"In this survey, you will be shown a hypothetical situation: You are 3 km away from your destination and must choose between different shared rental mobility options. Each option varies on four attributes: 1) CO2 Emissions, 2) Price for the full 3km trip, 3) Walking distance to the vehicle, and 4) Travel time.*[2]

The experimental design primarily identifies the main effects of each attribute by varying the attribute values in choice tasks. relative to competing attributes. Additionally, the survey incorporates respondent characteristics such as age, education, gender, and income. For example, someone with a high income might be less price sensitive. Large cross-national surveys find that people with a higher level of education tend to acknowledge that climate change is happening and might therefore care more about sustainability. By incorporating these respondent characteristics,

[2] See appendix 2 for the full design.

the design allows for the estimation of interaction effects between demographic variables and the sustainability attribute. These interactions help assess how factors such as education level shape individuals' preferences and their relative openness toward more sustainable mobility options.

Survey respondent demographics and descriptive stats

The survey included 122 respondents, most of whom were young adults: nearly half (49%) were aged 18–24, followed by 28% aged 25–34. The gender distribution was predominantly male (64%), with females representing 35% of the sample. Educational attainment was split mainly between bachelor's degree holders (38%) and those with a high-school education (37%). Income levels were broadly dispersed, though the largest groups reported earnings below €25,000 (26%) or between €25,000–49,999 (25%). When it came to sustainability, 39.5% of people stated they were "Mostly Concerned" with sustainability, 25.8% stated they were neither or not concerned, and 19.4% said they were mostly unconcerned, whilst only 3.2% said they were very concerned. Overall, the sample consisted mostly of younger participants with varying educational backgrounds, income levels, and a 50/50 attitude towards sustainability.

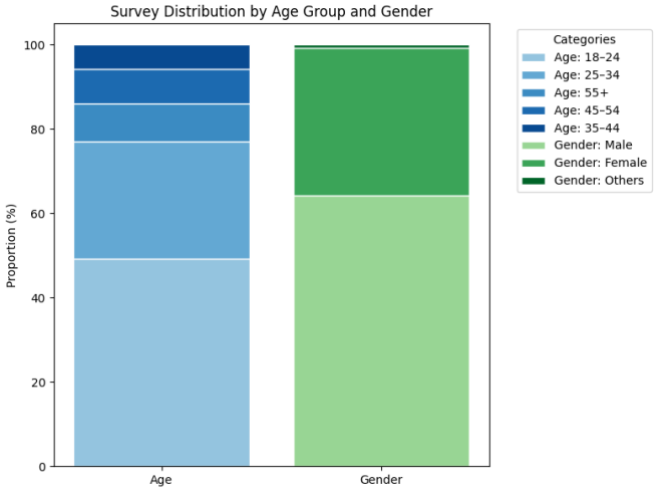


Figure 2 Survey Age and Gender Distribution

All demographic variables were converted into binary dummy variables by splitting the data. Age was split into below/above age 25. Income was split into below/above 50k. Gender was converted into Male/Female, as the "Other" category only had one response and would thus not yield any significant evaluation.

We estimated several discrete choice models to examine respondents' decision patterns. As a starting point, we employed a standard multinomial logit (MNL) specification using only the attributes included in the choice experiment. Because all attributes are categorical, reference levels were defined to avoid multicollinearity: for each attribute, the lowest or most preferable level served as the baseline. Specifically, the reference categories were No CO_2, €2, 100 meters, and 5 minutes. Another MNL model was run using continuous variables since the categorical version of the MNL yielded almost linear results in the effects.

All other levels were coded relative to these baselines. The resulting utility function for alternative i (where $i = 1, 2$) was defined as follows:

$$U_i = \beta_{C02,\,m} \cdot CO2_{m,i} + \beta_{C02,\,h} \cdot CO2_{h,i} + \beta_{price,\,35} \cdot price_{3.5,i} + \beta_{Dist,\,250} \cdot Dist_{250,i} + \beta_{Dist,\,500} \cdot Dist_{500,i} + \beta_{Time,\,10} \cdot Time_{10,i} + \beta_{Time,\,15} \cdot Time_{15,i} + \varepsilon_i$$

The Multinomial Logit with Interactions added the interaction of income with price and walking distance with age and sustainability perception with Co2 choice.

$$U_i = \beta_{C02,\,4x} \cdot CO2_{4x,i} + \beta_{C02,\,2x} \cdot CO2_{2x,i} + \beta_{price,\,3.5} \cdot price_{3.5,i} + \beta_{price,\,5} \cdot price_{5,i} + \beta_{Dist,\,250} \cdot Dist_{250,i} + \beta_{Dist,\,500} \cdot Dist_{500,i} + \beta_{Time,\,10} \cdot Time_{10,i} + \beta_{Time,\,15} \cdot Time_{15,i} + \beta_{Dist*Age} \cdot Dist * Age + \beta_{Income*Price} \cdot Income * Price + \beta_{Sustainability*Co2} \cdot Sustainability * Co2 + \varepsilon_i$$

Moreover, we estimated a Latent Class Multinomial Logit Model to identify potential subgroups within the shared mobility sample whose preferences differ systematically. The latent class approach assumes that the population is composed of a finite number of unobserved classes, each characterized by distinct preference structures. For this study, we specified two latent classes, motivated by the expectation that respondents may differ in how they value sustainability, price, distance to the vehicle, and travel time. A 2-segment model was run as it yielded more significant results.

In addition to the multinomial logit, we estimated a Mixed Logit Model to account for unobserved preference heterogeneity in shared-mobility choices. The utility specification followed the same structure as in the MNL model, but the attribute coefficients were treated as random rather than fixed. Each random parameter was expressed as:

$$U_i = \mu\beta + \sigma\beta + \varepsilon_i$$

Where the errors (ε_i) are normally distributed random variables with mean zero and unit variance. This formulation allows respondents to differ in how strongly they value changes in sustainability level, price, distance to the vehicle, and travel time.

Results

After estimating all the candidate models discussed previously, we selected the categorical MNL model with interaction effects as our final model. This selection was based on model fit and information criteria; these being the log-likelihood values (LL), the Akaike information criterion

(AIC), and the Schwarz information criterion (BIC). The values obtained by these models can be viewed in the appendix.

Our chosen model outperformed the alternative model options on every criterion (see Table 2), indicating that accounting for interaction effects greatly improves explanatory power. Therefore, we will discuss the results obtained in this model.

Table 2 Mixed Logit Performance Characteristics

Model	**LL**	**AIC**	**BIC**
MNL_continuous	-869.3	1748	1774
MNL_categorical	-866.38	1748.77	1790.95
MNL_interactions cat	-480.19	988.38	1054.45
Latent Class cat	-779.91	1603	1719
MixedLogit cat	-784.9	1601	1686
MNL_interactions cont	-855.23	1728	1775
Latent Class cont	-783.89	1599	1684
Mixed Logit cont	-725.11	1470	1522

Table 3 Multinomial Logit Model with interaction effects

Parameter	**Estimate**	**Std. error**	**P-value**
b_co2m	-0.028	0.138	0.841
b_co2h	-0.151	0.176	0.389
b_pricem	-1.033	0.214	0.000
b_priceh	-2.520	0.450	0.000
b_wdm	-0.199	0.131	0.130
b_wdh	-0.691	0.150	0.000
b_ttm	-0.509	0.110	0.000
b_tth	-1.122	0.189	0.000
b_co2m_green	-0.350	0.226	0.121
b_co2h_green	-0.669	0.272	0.015
b_pricem_wealthy	0.669	0.285	0.019
b_priceh_wealthy	1.229	0.562	0.029

b_wdm_ageold	-0.235	0.240	0.327
b_wdh_ageold	-0.504	0.292	0.085

Table 3 reports the parameter estimates. As expected, price, walking distance, and travel time all exert a negative effect on utility. Both medium and high price levels are highly significant and negative, indicating that respondents are strongly price sensitive. For walking distances, the coefficient for 500 meters is negative and statistically significant, while the effect of 250 meters is negative but not significant, suggesting that only relatively long walking distances substantially reduce utility. Likewise, both the 10-minute and 15-minute travel time levels significantly decrease utility, with the larger coefficient for 15 minutes indicating a stronger dislike of longer in-vehicle time. Interestingly, the C02 coefficient for both high and low emissions is negative, but not significant, suggesting that the model was not able to estimate how C02 impacts sustainability.

Small standard errors for Co2, travel time, and distance indicate that these effects would not vary much across repeated samples, so we can be relatively confident in their size and sign. Larger standard errors for the high price level and interaction effect terms imply more uncertainty around those effects, even though their signs align with expectations.

The interaction between sustainability and self-reported sustainability shows that only the high-emission level is statistically significant ($p = 0.015$), indicating that sustainability-aware individuals strongly dislike the most polluting options, while the medium level is not significant ($p = 0.121$). The income-price interactions are positive and significant at the 5% level ($p = 0.019$ and 0.029), meaning wealthier respondents are significantly less sensitive to both medium and high price increases. For age-distance, the coefficients are negative but not statistically significant at 5% (250 m: $p = 0.327$; 500 m: weakly significant at $p = 0.085$), suggesting older respondents may be more averse to long walking distances, but this effect is only weakly supported by the data.

Willingness to pay

Using the medium price coefficient in Table 5 as a monetary benchmark, the sustainability coefficients can be expressed as willingness to pay. For the average respondent, the utility loss from increasing emissions from 0% to 50% (–0.151) corresponds to a WTP of only about €0.15(0.151/1.033) to avoid that change. Among sustainability-oriented ("green") respondents, the relevant effect combines the main CO_2 coefficient and the interaction term (–0.151 – 0.669 = –

0.820), implying a much higher WTP of roughly €0.79 to avoid the high-emission option. This shows that while the typical respondent places a relatively small monetary value on emission reductions, environmentally conscious individuals are willing to pay several times more for the most sustainable alternative.

Marginal Utility Effects

Table 4 Marginal Effects of Multinomial Logit Model with interaction effects

Attribute	From	To	Alt DP
CO2	0% Co2	25% Co2	0.006945
CO2	0% Co2	50% Co2	0.03775
CO2_green	0% Co2	25% Co2	-0.0866
CO2_green	0% Co2	50% Co2	0.1593
Price	€2	€3.5	-0.2374
Price	€2	€5	-0.4255
Price_Rich	€2	€3.5	0.1612
Price_Rich	€2	€5	0.27355
Walking Distance	100m	250m	0.04948
Walking Distance	100m	500m	0.1662
Walking Distance_Age	100m	250m	0.05838
Walking Distance_Age	100m	500m	0.1232
Time Travel	5 min	10 min	0.1244
Time Travel	5 min	15 min	0.2543

**Only absolute values are reported*

Overall, the price attribute was found to have the biggest marginal effect. The highest price level was found to bring the most disutility to respondents. It is also observed that price is the single strongest driver in choice probability; moving from a price of 2 euros to 3.5 euros and from 2 euros to 5 euros reduces the probability of choosing an option by nearly 24 and 40 percentage points, respectively. Interestingly, respondents with higher income are substantially less price sensitive; the choice probabilities decrease by 16 and 27 percent, respectively. For CO2 emission levels, we obtain that higher emissions cause a noticeable drop in preference compared to moderate emissions. This effect is less obvious for the respondents who care more about sustainability.

Conclusion and Recommendation

Our study showed that interaction effects and relaxing the assumption that attributes have a linear effect on utility greatly improve model fit. It is therefore imperative that managers decide on their target segment.

Based on our findings, it is found that on average, people value the sustainability level of their shared mobility vehicle. It is thus beneficial for companies to perhaps express how green their vehicles really are. There are several simple features companies could add to their product to achieve this. We suggest a simple sticker on the vehicle with information on the CO2 emission level of the vehicle would be effective. This is a cheap way to inform the consumer about the sustainability level of a vehicle, leading to greater profits for the eco-friendlier options, as more consumers would pick these vehicles. In addition, this is further backed by the fact that the willingness to pay of more sustainability-aware individuals is substantially higher than that of non-sustainability-aware individuals.

Furthermore, the interaction effects, particularly of price with income and age with walking distance, suggest that placing mobility hubs in higher-income or older areas could justify premium pricing, as these groups appear less sensitive to price increases and more willing to pay for reduced access distance. Thus, highlighting a potential pool of customers that might normally revert to other mobility options.

Since we have chosen a categorical model, we cannot generalize our findings and infer the effect that an attribute would have when its value is outside of our attribute levels. We thus suggest that further research could include more levels of attributes. We also would have liked to include some other interaction effects. For example, how often people use shared mobility options and how this would impact their choice of a sustainable option. Furthermore, running multiple pilots would prove beneficial, especially when more interaction effects are included. Lastly, the lack of an opt-out option may have biased the estimates. Thus, a larger sample size including an opt-out alternative is recommended for more accurate results.

Taken together, our findings show that understanding preference heterogeneity is essential for designing a valued sustainable mobility product. By refining attribute specifications, incorporating more complex interactions, and expanding sample size, future work can build on these results to support data-driven decision-making in this sector.

Bibliography

Fu, X., van Lierop, D., & Ettema, D. (2024). Shared micromobility in multimodal travel: Evidence from three European cities. *The international journal of urban policy and planning*, https://doi.org/10.1016/j.cities.2024.105664.

Montes, A., Geržinic, N., Veeneman, W., van Oort, N., & Hoogendoorn, S. (2023). Shared micromobility and public transport integration - A mode choice study using stated preference data. *Research in Transportation Economics, 99*, 10.1016/j.retrec.2023.101302.

Reck, D. J., & Axhausen, K. W. (2021). Who uses shared micro-mobility services? Empirical evidence from Zurich, Switzerland. *Transportation research, 94*, https://doi.org/10.1016/j.trd.2021.102803.

Appendix

Appendix 1. Summary output tables of comprehensive analysis

Table 5 Estimates for Multinomial Logit model without interaction effects

Parameter	Estimate	Std. error	P-value
b_co2m	-0.188	0.084	0.026
b_co2h	-0.441	0.107	0.000
b_pricem	-0.754	0.108	0.000
b_priceh	-1.772	0.199	0.000
b_wdm	-0.183	0.094	0.052
b_wdh	-0.665	0.107	0.000
b_ttm	-0.603	0.090	0.000
b_tth	-1.208	0.137	0.000

Table 6 Log likelihood & Information criteria for MNL with no interaction effects

Statistic	Value
Log-Likelihood	-866.38
BIC	1790.95
AIC	1748.77

Table 7 basic MNL with continuous variables

Parameter	Estimate	Std. error	P-value
b_co2m	-0.200	0.084	0.017
b_co2h	-0.441	0.104	0.000
b_price	-0.592	0.065	0.000
b_wd	-0.002	0.000	0.000
b_tt	-0.122	0.014	0.000

Table 8 Log likelihood & Information criteria for basic continuous MNL

Statistic	Value
Log-Likelihood	-869.3
BIC	1774
AIC	1748

Table 9 Results for Multinomial Logit model with interaction effects

Parameter	Estimate	Std. error	P-value
b_co2m	-0.028	0.138	0.841
b_co2h	-0.151	0.176	0.389
b_pricem	-1.033	0.214	0.000
b_priceh	-2.520	0.450	0.000
b_wdm	-0.199	0.131	0.130
b_wdh	-0.691	0.150	0.000
b_ttm	-0.509	0.110	0.000
b_tth	-1.122	0.189	0.000
b_co2m_green	-0.350	0.226	0.121
b_co2h_green	-0.669	0.272	0.015
b_pricem_wealthy	0.669	0.285	0.019
b_priceh_wealthy	1.229	0.562	0.029
b_wdm_ageold	-0.235	0.240	0.327
b_wdh_ageold	-0.504	0.292	0.085

Table 10 Log likelihood & Information criteria for MNL with interaction effects

Statistic	Value
Log-Likelihood	-480.19
BIC	1054.45
AIC	988.38

Table 11 Estimates for Multinomial Logit model with interaction effects

Parameter	Estimate	Std. error	P-value
b_co2m	-0.028	0.138	0.841
b_co2h	-0.151	0.176	0.389
b_pricem	-1.033	0.214	0.000
b_priceh	-2.520	0.450	0.000
b_wdm	-0.199	0.131	0.130
b_wdh	-0.691	0.150	0.000
b_ttm	-0.509	0.110	0.000
b_tth	-1.122	0.189	0.000
b_co2m_green	-0.350	0.226	0.121
b_co2h_green	-0.669	0.272	0.015
b_pricem_wealthy	0.669	0.285	0.019
b_priceh_wealthy	1.229	0.562	0.029
b_wdm_ageold	-0.235	0.240	0.327
b_wdh_ageold	-0.504	0.292	0.085

Table 12 Results for Latent class model

Parameter	Estimate	Std. error	P-value
b_co2m_1	-0.430	0.265	0.105
b_co2h_1	-0.773	0.321	0.016
b_pricem_1	-0.449	0.228	0.049
b_priceh_1	-0.865	0.329	0.008
b_wdm_1	-0.655	0.444	0.140
b_wdh_1	-1.473	0.903	0.103
b_ttm_1	-1.256	0.786	0.110
b_tth_1	-2.498	1.643	0.128
b_co2m_2	-0.159	0.570	0.780
b_co2h_2	-0.751	0.391	0.055
b_pricem_2	-1.381	0.522	0.008
b_priceh_2	-3.357	1.769	0.058
b_wdm_2	-0.067	0.228	0.770
b_wdh_2	-0.513	0.191	0.007
b_ttm_2	-0.513	0.200	0.114
b_tth_2	-0.682	0.491	0.165
delta_1	-1.123	0.817	0.169
gamma_green_1	-0.115	0.926	0.901
gamma_ageold_1	0.353	0.915	0.699
gamma_gender_1	0.758	0.785	0.334
gamma_educated_1	0.602	0.528	0.254
gamma_wealthy_1	0.448	0.529	0.397
delta_2	0.0	0.0	0.0
gamma_green_2	0.0	0.0	0.0
gamma_ageold_2	0.0	0.0	0.0
gamma_gender_2	0.0	0.0	0.0
gamma_educated_2	0.0	0.0	0.0
gamma_wealthy_2	0.0	0.0	0.0

Table 13 Log likelihood & Information criteria for Latent class model

Statistic	**Value**
Log-Likelihood	-779.91
BIC	1719.83
AIC	1603.84

Table 14 Results for Mixed logit model

Parameter	Estimate	Std. error	P-value
mu_co2m	0.020	0.265	0.105
Sigma_co2m	-0.773	0.321	0.016
b_pricem_1	-0.449	0.228	0.049
b_priceh_1	-0.865	0.329	0.008
b_wdm_1	-0.655	0.444	0.140
b_wdh_1	-1.473	0.903	0.103
b_ttm_1	-1.256	0.786	0.110
b_tth_1	-2.498	1.643	0.128
b_co2m_2	-0.159	0.570	0.780
b_co2h_2	-0.751	0.391	0.055
b_pricem_2	-1.381	0.522	0.008
b_priceh_2	-3.357	1.769	0.058
b_wdm_2	-0.067	0.228	0.770
b_wdh_2	-0.513	0.191	0.007
b_ttm_2	-0.513	0.200	0.114
b_tth_2	-0.682	0.491	0.165
delta_1	-1.123	0.817	0.169
gamma_green_1	-0.115	0.926	0.901
gamma_ageold_1	0.353	0.915	0.699
gamma_gender_1	0.758	0.785	0.334
gamma_educated_1	0.602	0.528	0.254
gamma_wealthy_1	0.448	0.529	0.397
delta_2	0.0	0.0	0.0
gamma_green_2	0.0	0.0	0.0
gamma_ageold_2	0.0	0.0	0.0
gamma_gender_2	0.0	0.0	0.0
gamma_educated_2	0.0	0.0	0.0
gamma_wealthy_2	0.0	0.0	0.0

Table 15 Log likelihood & Information criteria for Mixed Logit model

Statistic	Value
Log-Likelihood	-784.9
BIC	1686
AIC	1601

Table 16 Continuous Latent Class model estimates

Parameter	Estimate	Std. error	P-value
b_co2m_1	-0.397	0.160	0.011
b_co2h_1	-0.657	0.266	0.013
b_price_1	-0.279	0.125	0.025
b_wd_1	-0.004	0.001	0.000
b_tt_1	-0.282	0.053	0.000
b_co2m_2	-0.255	0.160	0.111
b_co2h_2	-0.811	0.215	0.000
b_price_2	-1.002	0.155	0.000
b_wd_2	-0.001	0.000	0.001
b_tt_2	-0.065	0.019	0.001
delta_1	-1.284	0.727	0.078
g_green_1	-0.261	0.518	0.614
g_old_1	0.335	0.430	0.436
g_gender_1	0.793	0.485	0.102
g_educated_1	0.579	0.491	0.238
g_wealthy_1	0.400	0.438	0.361
delta_2	0.0	0.0	0.0
g_green_2	0.0	0.0	0.0
g_old_2	0.0	0.0	0.0
g_gender_2	0.0	0.0	0.0
g_educated_2	0.0	0.0	0.0
g_wealthy_2	0.0	0.0	0.0

Table 17 Log likelihood & Information criteria for Continuous Latent class model

Statistic	Value
Log-Likelihood	-783.89
BIC	1684
AIC	1599

Table 18 Continuous mixed Logit model estimates

Parameter	Estimate	Std. error	P-value
mu_co2m	-0.379	0.141	0.007
sigma_co2m	0.096	0.126	0.447
mu_co2h	-1.034	0.221	0.000
sigma_co2h	0.963	0.189	0.000
mu_price	-1.101	0.142	0.000
sigma_price	1.054	0.146	0.000
mu_wd	-0.003	0.001	0.000
sigma_wd	0.004	0.001	0.000
mu_tt	-0.237	0.032	0.000
sigma_tt	-0.226	0.028	0.000

Table 19 Log likelihood & Information criteria for Continuous Mixed Logit model

Statistic	Value
Log-Likelihood	-725.11
BIC	1522
AIC	1470

Table 20 Estimates for continuous Multinomial Logit model with interaction effects

Parameter	Estimate	Std. error	P-value
b_co2m	0.002	0.097	0.983
b_co2h	-0.131	0.122	0.284
b_price	-0.709	0.095	0.000
b_wd	-0.002	0.000	0.000
b_tt	-0.125	0.014	0.000
b_co2m_green	-0.450	0.166	0.007
b_co2h_green	-0.711	0.204	0.000
b_price_wealthy	0.237	0.135	0.079
b_wd_age	-0.000	0.001	0.748

Table 21 Log likelihood & Information criteria for Continuous MNL with interaction effects

Statistic	Value
Log-Likelihood	-855.23
BIC	1775
AIC	1728

Appendix 2. Survey design

Hi there,

Thank you for taking the time to participate in our survey. It should take about **3–5 minutes** to complete.

Please read the following carefully:

In this survey, you will be shown a hypothetical situation: You are 3 km away from your destination and **must choose between different shared rental mobility options**. Each option varies on four attributes:

- CO2 Emissions (Full lifecycle emissions of the vehicle compared to a city moped),
- Price for the full 3km trip,
- Walking distance to the vehicle,
- and Travel time (not including walking time).

For each choice, please select the option you would most likely choose in real life.

Which mobility option would you choose?

CO2 emissions are relative to a city moped.

	Option A	Option B
CO2 emissions	0% of moped	50% of moped
Price (3 km)	€2.00	€3.50
Distance to vehicle	500 m	250 m
Travel time (3 km)	15 min	10 min

◯ A

◯ B

Choice tasks:

1)

Which mobility option would you choose?

CO2 emissions are relative to a city moped.

	Option A	Option B
CO2 emissions	0% of moped	25% of moped
Price (3 km)	€2.00	€3.50
Distance to vehicle	500 m	100 m
Travel time (3 km)	10 min	15 min

2)

Which mobility option would you choose?

CO2 emissions are relative to a city moped.

	Option A	Option B
CO2 emissions	25% of moped	50% of moped
Price (3 km)	€5.00	€3.50
Distance to vehicle	500 m	250 m
Travel time (3 km)	10 min	5 min

3)

Which mobility option would you choose?

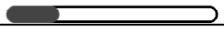
CO2 emissions are relative to a city moped.

	Option A	Option B
CO2 emissions	25% of moped	0% of moped
Price (3 km)	€2.00	€3.50
Distance to vehicle	100 m	500 m
Travel time (3 km)	15 min	10 min

4)

Which mobility option would you choose?

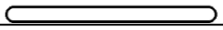
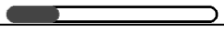
CO2 emissions are relative to a city moped.

	Option A	Option B
CO2 emissions	50% of moped 	25% of moped 
Price (3 km)	€3.50	€5.00
Distance to vehicle	500 m	100 m
Travel time (3 km)	15 min	5 min

5)

Which mobility option would you choose?

CO2 emissions are relative to a city moped.

	Option A	Option B
CO2 emissions	0% of moped 	25% of moped 
Price (3 km)	€5.00	€3.50
Distance to vehicle	100 m	250 m
Travel time (3 km)	15 min	10 min

6)

Which mobility option would you choose?

CO2 emissions are relative to a city moped.

	Option A	Option B
CO2 emissions	0% of moped 	25% of moped 
Price (3 km)	€2.00	€3.50
Distance to vehicle	250 m	100 m
Travel time (3 km)	15 min	5 min

7)

Which mobility option would you choose?

CO2 emissions are relative to a city moped.

	Option A	Option B
CO2 emissions	50% of moped	0% of moped
Price (3 km)	€2.00	€3.50
Distance to vehicle	500 m	250 m
Travel time (3 km)	15 min	5 min

8)

Which mobility option would you choose?

CO2 emissions are relative to a city moped.

	Option A	Option B
CO2 emissions	25% of moped	50% of moped
Price (3 km)	€2.00	€3.50
Distance to vehicle	500 m	250 m
Travel time (3 km)	10 min	5 min

9)

Which mobility option would you choose?

CO2 emissions are relative to a city moped.

	Option A	Option B
CO2 emissions	25% of moped	0% of moped
Price (3 km)	€3.50	€5.00
Distance to vehicle	500 m	100 m
Travel time (3 km)	15 min	5 min

10)

Which mobility option would you choose?

CO2 emissions are relative to a city moped.

	Option A	Option B
CO2 emissions	50% of moped	0% of moped
Price (3 km)	€5.00	€3.50
Distance to vehicle	100 m	500 m
Travel time (3 km)	5 min	10 min

11)

Which mobility option would you choose?

CO2 emissions are relative to a city moped.

	Option A	Option B
CO2 emissions	25% of moped	0% of moped
Price (3 km)	€2.00	€3.50
Distance to vehicle	250 m	500 m
Travel time (3 km)	10 min	15 min

12)

Which mobility option would you choose?

CO2 emissions are relative to a city moped.

	Option A	Option B
CO2 emissions	50% of moped	0% of moped
Price (3 km)	€5.00	€3.50
Distance to vehicle	500 m	250 m
Travel time (3 km)	5 min	15 min

13)

Which mobility option would you choose?

CO2 emissions are relative to a city moped.

	Option A	Option B
CO2 emissions	0% of moped	50% of moped
Price (3 km)	€5.00	€3.50
Distance to vehicle	100 m	500 m
Travel time (3 km)	15 min	5 min

14)

Which mobility option would you choose?

CO2 emissions are relative to a city moped.

	Option A	Option B
CO2 emissions	50% of moped	25% of moped
Price (3 km)	€3.50	€2.00
Distance to vehicle	100 m	250 m
Travel time (3 km)	10 min	15 min

15)

Which mobility option would you choose?

CO2 emissions are relative to a city moped.

	Option A	Option B
CO2 emissions	50% of moped	0% of moped
Price (3 km)	€3.50	€5.00
Distance to vehicle	100 m	500 m
Travel time (3 km)	15 min	5 min

16)

Which mobility option would you choose?

CO2 emissions are relative to a city moped.

	Option A	Option B
CO2 emissions	50% of moped	0% of moped
Price (3 km)	€2.00	€3.50
Distance to vehicle	500 m	250 m
Travel time (3 km)	5 min	10 min

17)

Which mobility option would you choose?

CO2 emissions are relative to a city moped.

	Option A	Option B
CO2 emissions	0% of moped	50% of moped
Price (3 km)	€3.50	€2.00
Distance to vehicle	100 m	250 m
Travel time (3 km)	5 min	15 min

18)

Which mobility option would you choose?

CO2 emissions are relative to a city moped.

	Option A	Option B
CO2 emissions	25% of moped	0% of moped
Price (3 km)	€3.50	€5.00
Distance to vehicle	500 m	100 m
Travel time (3 km)	5 min	10 min

19)

Which mobility option would you choose?

CO2 emissions are relative to a city moped.

	Option A	Option B
CO2 emissions	25% of moped	50% of moped
Price (3 km)	€3.50	€2.00
Distance to vehicle	100 m	250 m
Travel time (3 km)	15 min	10 min

20)

Which mobility option would you choose?

CO2 emissions are relative to a city moped.

	Option A	Option B
CO2 emissions	50% of moped	25% of moped
Price (3 km)	€5.00	€3.50
Distance to vehicle	250 m	100 m
Travel time (3 km)	5 min	10 min

21)

Which mobility option would you choose?

CO2 emissions are relative to a city moped.

	Option A	Option B
CO2 emissions	0% of moped	50% of moped
Price (3 km)	€2.00	€3.50
Distance to vehicle	500 m	250 m
Travel time (3 km)	15 min	10 min

22)

Which mobility option would you choose?

CO2 emissions are relative to a city moped.

	Option A	Option B
CO2 emissions	50% of moped	25% of moped
Price (3 km)	€3.50	€5.00
Distance to vehicle	100 m	250 m
Travel time (3 km)	10 min	5 min

23)

Which mobility option would you choose?

CO2 emissions are relative to a city moped.

	Option A	Option B
CO2 emissions	0% of moped	50% of moped
Price (3 km)	€3.50	€2.00
Distance to vehicle	500 m	100 m
Travel time (3 km)	5 min	10 min

24)

Which mobility option would you choose?

CO2 emissions are relative to a city moped.

	Option A	Option B
CO2 emissions	50% of moped	25% of moped
Price (3 km)	€3.50	€5.00
Distance to vehicle	100 m	250 m
Travel time (3 km)	5 min	10 min

Each respondent was shown 12 Choice tasks at random after which they were asked the following demographic questions:

How concerned are you about the environment/climate (for example reducing carbon emissions)?

- ◯ 1. Not at all concerned
- ◯ 2. Mostly unconcerned
- ◯ 3. Neither concerned nor unconcerned
- ◯ 4. Mostly concerned
- ◯ 5. Very concerned

What is your age?

Gender

- ◯ Female
- ◯ Male
- ◯ Others
- ◯ Prefer not to say

What is the highest level of education you have achieved?

- ◯ High school
- ◯ Applied sciences
- ◯ Bachelor's degree
- ◯ Master's degree
- ◯ PhD
- ◯ Other

What was your total household income before taxes during the past 12 months in Euros?

- ◯ Less than 25,000 Euros
- ◯ 25,000 - 49,999 Euros per year
- ◯ 50,000 - 99,999 Euros per year
- ◯ 100,000 - 199,999 Euros per year
- ◯ 200,000 Euros per year or more
- ◯ Prefer not to say

Appendix 3. Syntax used to generate the generate the various models

Code MNL basic with categorical variables only

```
### Step 1: Clear memory
rm(list = ls())

### Step 2: Set working directory for R initialization
setwd(dirname(rstudioapi::getActiveDocumentContext()$path))

### Step 3: Load Apollo library
library(apollo)
```

```
### Step 4: Initialize Apollo code
apollo_initialise()

### Step 5: Set core controls
###     ATTENTION: Your inputs must be enclosed in quotes like "this"
apollo_control = list(
  # USER ACTION: Specify model name
  ##    Note: Change the model name for every model that you run
  modelName       = "MNL_basic",
  # USER ACTION: Provide model description
  ##    Note: Change the model description to reflect the current model
  modelDescr      = "basic model",
  # USER ACTION: Specify the column with the respondent id
  indivID         = "peep_ID",
  # USER ACTION: Set path to the folder on your PC where the model results
will be stored
  ##    Note: Use the "outputs" folder that was created by the pre-
processing syntax
  outputDirectory ="outputs"
)

### Step 6: Load data
# Set path to the folder on your PC where the dataset is stored
path_dataset =
paste0(getwd(),sep=.Platform$file.sep,"data_with_dummies_final.csv")

# Load dataset into global environment
database = read.csv(path_dataset, header=TRUE)

### Step 7: Initialize all parameters that needs to be estimated in your
MNL model
# USER ACTION: The parameters for the attribute "Effectiveness" are
defined. Please,
#              complete the list with parameters that need to be
estimated. Provide
#              names for each parameter following by assigning a starting
value.
apollo_beta=c(b_co2m = 0,
              b_co2h = 0,
              b_pricem = 0,
              b_priceh = 0,
              b_wdm = 0,
              b_wdh = 0,
              b_ttm = 0,
              b_tth = 0)
              #'COMPLETE THE LIST HERE')

### Step 8: Define which parameters (as initialised above) should kept
fixed during estimation (in quotes); if none, keep empty
apollo_fixed = c()

### Step 9: Checkpoint for model inputs
apollo_inputs = apollo_validateInputs()
```

```
### Step 10: Define model and likelihood function
apollo_probabilities=function(apollo_beta, apollo_inputs,
functionality="estimate"){

  ### Attach dataset inputs and detach after function exit
  apollo_attach(apollo_beta, apollo_inputs)
  on.exit(apollo_detach(apollo_beta, apollo_inputs))

  ### Create list of choice probabilities P
  P = list()

  ### List of utility functions: these must use the same names as in
mnl_settings (see below), order is irrelevant
  V = list()
  # USER ACTION: Define utility function for alternative 1
  V[["ALT1"]] = b_co2m * Var12.1 + b_co2h * Var13.1 + b_pricem * Var22.1 +
b_priceh * Var23.1 +
    b_wdm * Var32.1 + b_wdh * Var33.1 + b_ttm * Var42.1 + b_tth * Var43.1

  # USER ACTION: Define utility function for alternative 2
  V[["ALT2"]] = b_co2m * Var12.2 + b_co2h * Var13.2 + b_pricem * Var22.2 +
b_priceh * Var23.2 +
    b_wdm * Var32.2 + b_wdh * Var33.2 + b_ttm * Var42.2 + b_tth * Var43.2

  ### Define settings for MNL model component
  mnl_settings = list(
    # USER ACTION: Attach utility functions to the alternatives in your
dataset
    alternatives  = c(ALT1=1, ALT2=2),
    # USER ACTION: Define which alternatives are "available" in each
choice task; in our study, all alternatives are "available"
    avail         = list(ALT1=1, ALT2=1),
    # USER ACTION: Specify the column containing the chosen alternative;
beware, no dummies are used (!)
    choiceVar     = choice,
    # USER ACTION: Attach list of utility functions
    utilities     = V
  )

  ### Compute choice probabilities using MNL model
  P[["model"]] = apollo_mnl(mnl_settings, functionality) #
functionality="estimate" as the parameters will be updated for estimating
the MNL model

  ### Take product across observations for same individual (i.e.,
considering the panel structure of the data)
  P = apollo_panelProd(P, apollo_inputs, functionality)

  ### Prepare and return outputs of function
  P = apollo_prepareProb(P, apollo_inputs, functionality)
  return(P)
}

### Step 11: Model estimation
```

```
model = apollo_estimate(apollo_beta, apollo_fixed, apollo_probabilities,
apollo_inputs)

### Step 12: Print model output with two-sided p-values
#### Note: if one-sided p-values are needed, set "printPVal=1" (p-values
are not reported if set to "0")
modelOutput_setting=list(printPVal=2)
apollo_modelOutput(model, modelOutput_setting)

### Save model output with two-sided p-values
apollo_saveOutput(model, modelOutput_setting)
```

Code MNL with interactions with categorical variables only

```
### Step 1: Clear memory
rm(list = ls())

### Step 2: Set working directory for R initialization
setwd(dirname(rstudioapi::getActiveDocumentContext()$path))

### Step 3: Load Apollo library
library(apollo)

### Step 4: Initialise Apollo code
apollo_initialise()

### Step 5: Set core controls
###     ATTENTION: Your inputs must be enclosed in quotes like "this"
apollo_control = list(
  # USER ACTION: Specify model name
  ##    Note: Change the model name for every model that you run
  modelName       = "MNL_I",
  # USER ACTION: Provide model description
  ##    Note: Change the model description to reflect the current model
  modelDescr      = "MNL with interaction effects",
  # USER ACTION: Specify the column with the respondent id
  indivID         = "peep_ID",
  # USER ACTION: Set path to the folder on your PC where the model results
will be stored
  ##    Note: Use the "outputs" folder that was created by the pre-
processing syntax
  outputDirectory = "outputs"
)

### Step 6: Load data
# Set path to the folder on your PC where the dataset is stored
path_dataset = paste0(getwd(),sep=.Platform$file.sep,"data_with_dummies-
2.csv")

# Load dataset into global environment
database = read.csv(path_dataset, header=TRUE)
```

```
### Step 7: Initialize all parameters that needs to be estimated in your
MNL model
# USER ACTION: The parameters for the attribute "Effectiveness" are
defined. Please,
#              complete the list with parameters that need to be
estimated. Provide
#              names for each parameter following by assigning a starting
value.
apollo_beta=c(b_co2m = 0,
              b_co2h = 0,
              b_pricem = 0,
              b_priceh = 0,
              b_wdm = 0,
              b_wdh = 0,
              b_ttm = 0,
              b_tth = 0,
              b_co2m_treehugger = 0,
              b_co2h_treehugger = 0,
              b_pricem_rich = 0,
              b_priceh_rich = 0,
              b_wdm_age = 0,
              b_wdh_age = 0)

### Step 8: Define which parameters (as initialized above) should kept
fixed during estimation (in quotes); if none, keep empty
apollo_fixed = c()

### Step 9: Checkpoint for model inputs
apollo_inputs = apollo_validateInputs()

### Step 10: Define model and likelihood function
apollo_probabilities=function(apollo_beta, apollo_inputs,
functionality="estimate"){

  ### Attach dataset inputs and detach after function exit
  apollo_attach(apollo_beta, apollo_inputs)
  on.exit(apollo_detach(apollo_beta, apollo_inputs))

  ### Create list of choice probabilities P
  P = list()

  ### List of utility functions: these must use the same names as in
mnl_settings (see below), order is irrelevant
  V = list()
  # USER ACTION: Define utility function for alternative 1
  V[["ALT1"]]  = (b_co2m * Var12.1 + b_co2h * Var13.1 + b_pricem * Var22.1
+ b_priceh * Var23.1 + b_wdm * Var32.1 + b_wdh * Var33.1 + b_ttm * Var42.1
+ b_tth * Var43.1 +
                  b_co2m_treehugger * d26 * Var12.1 + b_co2h_treehugger *
d26 * Var13.1 + b_pricem_rich * d30 * Var22.1 + b_priceh_rich * d30 *
Var23.1 +
                  b_wdm_age * d27 * Var32.1 + b_wdh_age * d27 * Var33.1)

  # USER ACTION: Define utility function for alternative 2
```

```
  V[["ALT2"]]  =(b_co2m * Var12.2 + b_co2h * Var13.2 + b_pricem * Var22.2
+ b_priceh * Var23.2 + b_wdm * Var32.2 + b_wdh * Var33.2 + b_ttm * Var42.2
+ b_tth * Var43.2 +
                   b_co2m_treehugger * d26 * Var12.2 + b_co2h_treehugger *
d26 * Var13.2 + b_pricem_rich * d30 * Var22.2 + b_priceh_rich * d30 *
Var23.2 +
                   b_wdm_age * d27 * Var32.2 + b_wdh_age * d27 * Var33.2)

  ### Define settings for MNL model component
  mnl_settings = list(
    # USER ACTION: Attach utility functions to the alternatives in your
dataset
    alternatives  = c(ALT1=1, ALT2=2),
    # USER ACTION: Define which alternatives are "available" in each
choice task; in our study, all alternatives are "available"
    avail         = list(ALT1=1, ALT2=1),
    # USER ACTION: Specify the column containing the chosen alternative;
beware, no dummies are used (!)
    choiceVar     = choice,
    # USER ACTION: Attach list of utility functions
    utilities     = V
  )

  ### Compute choice probabilities using MNL model
  P[["model"]] = apollo_mnl(mnl_settings, functionality) #
functionality="estimate" as the parameters will be updated for estimating
the MNL model

  ### Take product across observations for same individual (i.e.,
considering the panel structure of the data)
  P = apollo_panelProd(P, apollo_inputs, functionality)

  ### Prepare and return outputs of function
  P = apollo_prepareProb(P, apollo_inputs, functionality)
  return(P)
}

### Step 11: Model estimation
model = apollo_estimate(apollo_beta, apollo_fixed, apollo_probabilities,
apollo_inputs)

### Step 12: Print model output with two-sided p-values
#### Note: if one-sided p-values are needed, set "printPVal=1" (p-values
are not reported if set to "0")
modelOutput_setting=list(printPVal=2)
apollo_modelOutput(model, modelOutput_setting)

### Save model output with two-sided p-values
apollo_saveOutput(model, modelOutput_setting)
```

Code Latent Class with categorical variables only

```
### Step 1: Clear memory
```

```
rm(list = ls())

### Step 2: Set working directory for R initialization
setwd(dirname(rstudioapi::getActiveDocumentContext()$path))

### Step 3: Load Apollo library
library(apollo)

### Step 4: Initialise Apollo code
apollo_initialise()

### Step 5: Set core controls
###     ATTENTION: Your inputs must be enclosed in quotes like "this"
apollo_control = list(
  # USER ACTION: Specify model name
  ##    Note: Change the model name for every model that you run
  modelName       = "lc_model",
  # USER ACTION: Provide model description
  ##    Note: Change the model description to reflect the current model
  modelDescr      = "this estimates lc",
  # USER ACTION: Specify the column with the respondent id
  indivID         = "peep_ID",
  # Define number of cores used during estimation (used to speed up
estimation time)
  nCores          = 5,
  # Define seed used for any random number generation
  seed            = 100,
  # USER ACTION: Set path to the folder on your PC where the model results
will be stored
  ##    Note: Use the "outputs" folder that was created by the pre-
processing syntax
  outputDirectory =
paste(getwd(),'outputs',"practicum_lcmodel_3class_covar",sep=.Platform$fil
e.sep)
)

### Step 6: Load data
# Set path to the folder on your PC where the dataset is stored
path_data =
paste(getwd(),"data_with_dummies_final.csv",sep=.Platform$file.sep)
# Load dataset into global environment
database = read.csv(path_data, header=TRUE)

### Step 7: Initialise all parameters that needs to be estimated in your
MNL model
# USER ACTION: Define the (1) class-specific and (2) class membership
parameters
#              followed by assigning a starting value. The class-specific
#              alternative specific constant for the opt-out option and
the
#              constants for the class membership models are already
defined.
#              Please, complete the list with the parameters that are
missing.
```

```
#              Provide names for each parameter following by assigning a
#              starting value.
apollo_beta=c(# Class 1
  b_co2m_1 = -0.1,
  b_co2h_1 = 0,
  b_pricem_1 = 0,
  b_priceh_1 = 0,
  b_wdm_1 = 0,
  b_wdh_1 = 0,
  b_ttm_1 = 0,
  b_tth_1 = 0,

  # Class 2
  b_co2m_2 = 0.1,
  b_co2h_2 = 0,
  b_pricem_2 = 0,
  b_priceh_2 = 0,
  b_wdm_2 = 0,
  b_wdh_2 = 0,
  b_ttm_2 = 0,
  b_tth_2 = 0,

  # Class membership - class 1
  delta_1        = -0.1,
  g_treehugger_1 = 0,
  g_ageold_1 = 0,
  g_gender_1 = 0,
  g_educated_1 = 0,
  g_rich_1 = 0,
  # Class membership - class 2
  delta_2        = 0,
  g_treehugger_2 = 0,
  g_ageold_2 = 0,
  g_gender_2 = 0,
  g_educated_2 = 0,
  g_rich_2 = 0)


### Step 8
##  USER ACTION: Complete the list with parameters (as initialised above)
that
##               should be kept fixed during estimation (in quotes)
apollo_fixed = c("delta_2", "g_treehugger_2", "g_ageold_2", "g_gender_2",
"g_educated_2", "g_rich_2")

### Step 9: Define class membership model
apollo_lcPars=function(apollo_beta, apollo_inputs){
  lcpars = list()
  ## USER ACTION: Complete the empty lists by specifying the missing
class-specific parameters
  ##              which are needed for the class-specific utility
functions

  lcpars[["b_co2m"]] = list(b_co2m_1, b_co2m_2)
```

```
  lcpars[["b_co2h"]] = list(b_co2h_1, b_co2h_2)
  lcpars[["b_pricem"]] = list(b_pricem_1, b_pricem_2)
  lcpars[["b_priceh"]] = list(b_priceh_1, b_priceh_2)
  lcpars[["b_wdm"]] = list(b_wdm_1, b_wdm_2)
  lcpars[["b_wdh"]] = list(b_wdh_1, b_wdh_2)
  lcpars[["b_ttm"]] = list(b_ttm_1, b_ttm_2)
  lcpars[["b_tth"]] = list(b_tth_1, b_tth_2)


  ## List of class-membership functions:
  ##  These must use the same names as in classAlloc_settings (see below),
order is irrelevant
  V=list()
  # USER ACTION: Define class-membership function for class 1
  V[["class_1"]] = delta_1 + g_treehugger_1 * d26 + g_ageold_1 * d27 +
g_gender_1 * d28 + g_educated_1 * d29 + g_rich_1 * d30

  # USER ACTION: Define class-membership functions for class 2
  V[["class_2"]] = delta_2 + g_treehugger_2 * d26 + g_ageold_2 * d27 +
g_gender_2 * d28+ g_educated_2 * d29 + g_rich_2 * d30


  ## Define settings for class-membership model
  classAlloc_settings = list(
    # USER ACTION: Attach class-membership functions to the respective
classes
    classes      = c(class_1=1, class_2=2),
    # USER ACTION: Define which classes are "available" in our study, all
classes are "available"
    avail        = 1,
    # USER ACTION: Attach list of class-membership functions
    utilities    = V
  )

  lcpars[["pi_values"]] = apollo_classAlloc(classAlloc_settings)
  return(lcpars)
}

### Step 10: Checkpoint for model inputs
apollo_inputs = apollo_validateInputs()

### Step 11: Define model and likelihood function
apollo_probabilities=function(apollo_beta, apollo_inputs,
functionality="estimate"){

  ### Attach inputs and detach after function exit
  apollo_attach(apollo_beta, apollo_inputs)
  on.exit(apollo_detach(apollo_beta, apollo_inputs))

  ### Create list of choice probabilities P
  P = list()

  ### Define settings for MNL model component that are generic across
classes
```

```
  mnl_settings = list(
    # USER ACTION: Attach utility functions to the alternatives in your
dataset
    alternatives  = c(ALT1=1, ALT2=2),
    # USER ACTION: Define which alternatives are "available" in each
choice task; in our study, all alternatives are "available"
    avail         = list(ALT1=1, ALT2=1),
    # USER ACTION: Specify the column containing the chosen alternative;
beware, no dummies are used (!)
    choiceVar     = choice
  )

  ### List of utility functions for each latent class: these must use the
same names as in mnl_settings (see above), order is irrelevant
  # USER ACTION: Set number of latent classes you are estimating in the
model
  ##       Note: You can call class-specific parameters by
NAME_PARAM[[s]]; see example in ALT3 for the class-specific
  ##             alternative specific constant
  for(s in 1:2){
    V=list()
    # USER ACTION: Define utility function for alternative 1 for class "s"
    V[["ALT1"]] = b_co2m[[s]] * Var12.1 + b_co2h[[s]] * Var13.1 +
b_pricem[[s]] * Var22.1 + b_priceh[[s]] * Var23.1 +
      b_ttm[[s]] * Var42.1 + b_tth[[s]] * Var43.1 + b_wdm[[s]] * Var32.1 +
b_wdh[[s]] * Var33.1

    # USER ACTION: Define utility function for alternative 2 for class "s"
    V[["ALT2"]] = b_co2m[[s]] * Var12.2 + b_co2h[[s]] * Var13.2 +
b_pricem[[s]] * Var22.2 + b_priceh[[s]] * Var23.2 +
      b_ttm[[s]] * Var42.2 + b_tth[[s]] * Var43.2 + b_wdm[[s]] * Var32.2 +
b_wdh[[s]] * Var33.2


    mnl_settings$utilities = V
    mnl_settings$componentName = paste0("Class_",s)

    ### Compute within-class choice probabilities using MNL model
    P[[paste0("Class_",s)]] = apollo_mnl(mnl_settings, functionality)

    ### Take product across observations for same individual (i.e.,
considering the panel structure of the data)
    P[[paste0("Class_",s)]] = apollo_panelProd(P[[paste0("Class_",s)]],
apollo_inputs ,functionality)
  }

  ### Compute latent class model probabilities
  lc_settings   = list(inClassProb = P, classProb=pi_values)
  P[["model"]] = apollo_lc(lc_settings, apollo_inputs, functionality)

  ### Prepare and return outputs of function
  P = apollo_prepareProb(P, apollo_inputs, functionality)
  return(P)
}
```

```
### Step 12: Searching for starting value (recommended to ensure model
convergence!)
apollo_beta = apollo_searchStart(apollo_beta,
                                 apollo_fixed,
                                 apollo_probabilities,
                                 apollo_inputs,
                                 searchStart_settings=list(nCandidates=2))

### Step 13: Model estimation
model = apollo_estimate(apollo_beta, apollo_fixed, apollo_probabilities,
apollo_inputs)

### Step 14: Print model output with two-sided p-values
### Note: if one-sided p-values are needed, set "printPVal=1" (p-values
are not reported if set to "0")
modelOutput_setting=list(printPVal=2)
apollo_modelOutput(model, modelOutput_setting)

### Save model output with two-sided p-values
apollo_saveOutput(model, modelOutput_setting)
```

Code Mixed Logit with categorical variables only

```
### Step 1: Clear memory
rm(list = ls())

### Step 2: Set working directory for R initialization
setwd(dirname(rstudioapi::getActiveDocumentContext()$path))

### Step 3: Load Apollo library
library(apollo)

### Step 4: Initialise Apollo code
apollo_initialise()

### Step 5: Set core controls
###     ATTENTION: Your inputs must be enclosed in quotes like "this"
apollo_control = list(
  # USER ACTION: Specify model name
  ##    Note: Change the model name for every model that you run
  modelName       = "mixedLogitModel",
  # USER ACTION: Provide model description
  ##    Note: Change the model description to reflect the current model
  modelDescr      = "mixing",
  # USER ACTION: Specify the column with the respondent id
  indivID         = "peep_ID",
  # USER ACTION: Set logical variable to activate estimation of random
parameters
  mixing          = TRUE,
  # Define number of cores used during estimation (used to speed up
estimation time)
  nCores          = 5,
```

```
  # USER ACTION: Set path to the folder on your PC where the model results
will be stored
  ##    Note: Use the "outputs" folder that was created by the pre-
processing syntax
  outputDirectory = "outputs"
)

### Step 6: Load data
# Set path to directory on your PC where the dataset is stored
path_data =
paste0(getwd(),sep=.Platform$file.sep,"data_with_dummies_final.csv")
# Load dataset into global environment
database = read.csv(path_data, header=TRUE)
database <- database[!is.na(database$choice), ]

### Step 7: Initialise all parameters that needs to be estimated in your
Mixed Logit model
# USER ACTION: Define the (1) mu parameters that estimate the sample mean,
and
#                          (2) sigma parameters that estimate the sample
distribution
#              Please, complete the list with the parameters that are
missing.
#              Provide names for each parameter following by assigning a
#              starting value.
apollo_beta=c(
  mu_co2m = 0,
  sigma_co2m = 0,
  mu_co2h = 0,
  sigma_co2h = 0,
  mu_pricem = 0,
  sigma_pricem = 0,
  mu_priceh = 0,
  sigma_priceh = 0,
  mu_wdm = 0,
  sigma_wdm = 0,
  mu_wdh = 0,
  sigma_wdh = 0,
  mu_ttm = 0,
  sigma_ttm = 0,
  mu_tth = 0,
  sigma_tth = 0)
### Step 8:
##  USER ACTION: Complete the list with parameters (as initialised above)
that
##               should be kept fixed during estimation (in quotes); if
none, keep empty
apollo_fixed = c()

### Step 9: Set parameters for generating draws
# USER ACTION: Define the number of one random variable for each sigma in
apollo_beta
# Use the command line interNormDraws
```

```
apollo_draws = list(
  interDrawsType = "mlhs",
  interNDraws    = 200,
  interUnifDraws = c(),
  interNormDraws = c("inter_1","inter_2","inter_3","inter_4","inter_5",
"inter_6", "inter_7", "inter_8"),
  intraDrawsType = "mlhs",
  intraNDraws    = 0,
  intraUnifDraws = c(),
  intraNormDraws = c()
)

### Step 10: Create random parameters
# USER ACTION: Write every random coefficient function
# If necessary check the lecture slides
apollo_randCoeff = function(apollo_beta, apollo_inputs){
  randcoeff = list()

  randcoeff[["b_co2m"]] = mu_co2m + sigma_co2m * inter_1
  randcoeff[["b_co2h"]] = mu_co2h + sigma_co2h * inter_2
  randcoeff[["b_pricem"]] = mu_pricem + sigma_pricem* inter_3
  randcoeff[["b_priceh"]] = mu_priceh + sigma_priceh* inter_4
  randcoeff[["b_wdm"]] = mu_wdm + sigma_wdm* inter_5
  randcoeff[["b_wdh"]] = mu_wdh + sigma_wdh* inter_6
  randcoeff[["b_ttm"]] = mu_ttm + sigma_ttm* inter_7
  randcoeff[["b_tth"]] = mu_tth + sigma_tth* inter_8

  return(randcoeff)
}

### Step 11: Checkpoint for model inputs
apollo_inputs = apollo_validateInputs()

### Step 12: Define model and likelihood function
apollo_probabilities=function(apollo_beta, apollo_inputs,
functionality="estimate"){

  ### Attach dataset inputs and detach after function exit
  apollo_attach(apollo_beta, apollo_inputs)
  on.exit(apollo_detach(apollo_beta, apollo_inputs))

  ### Create list of choice probabilities P
  P = list()

  ### List of utility functions: these must use the same names as in
mnl_settings (see below), order is irrelevant
  V = list()

  # USER ACTION: Define utility function for alternative 1
  # Code "effectiveness" and "risk false negative" attributes as numerical
variables
  V[["ALT1"]]  = (b_co2m * Var12.1 + b_co2h * Var13.1 + b_pricem * Var22.1
+ b_priceh * Var23.1 + b_wdm * Var32.1
                  + b_wdh * Var33.1 + b_ttm * Var42.1 + b_tth * Var43.1 )
```

```
  # USER ACTION: Define utility function for alternative 2
  # Code "effectiveness" and "risk false negative" attributes as numerical
variables
  V[["ALT2"]]  = (b_co2m * Var12.2 + b_co2h * Var13.2 + b_pricem * Var22.2
+ b_priceh * Var23.2 + b_wdm * Var32.2
                  + b_wdh * Var33.2 + b_ttm * Var42.2 + b_tth * Var43.2 )


  ### Define settings for MNL model component
  mnl_settings = list(
    # USER ACTION: Attach utility function to the choice alternative in
your dataset
    alternatives  = c(ALT1=1, ALT2=2),
    # USER ACTION: Define which alternatives are "available" in each
choice task
    #              In our study, all alternatives are "available"
    avail         = 1,
    # USER ACTION: Specify the column containing the chosen alternative
    choiceVar     = choice,
    # USER ACTION: Attach list of utility functions
    utilities     = V
  )

  ### Compute choice probabilities using MNL model
  #### functionality="estimate" as the parameters will be updated for
estimating the MNL model
  P[["model"]] = apollo_mnl(mnl_settings, functionality)

  ### Take product across observations for same individual
  ### (i.e., considering the panel structure of the data)
  P = apollo_panelProd(P, apollo_inputs, functionality)

  ## Average across inter-individual draws
  P = apollo_avgInterDraws(P, apollo_inputs, functionality)

  ### Prepare and return outputs of function
  P = apollo_prepareProb(P, apollo_inputs, functionality)
  return(P)
}

### Step 13: Model estimation
model = apollo_estimate(apollo_beta, apollo_fixed, apollo_probabilities,
apollo_inputs)

### Step 14: Print model output with two-sided p-values
#### Note: if one-sided p-values are needed, set "printPVal=1" (p-values
are not reported if set to "0")
modelOutput_setting=list(printPVal=2)
apollo_modelOutput(model, modelOutput_setting)

### Save model output with two-sided p-values
apollo_saveOutput(model, modelOutput_setting)
```

```
### Step 15: Estimate individual coefficients conditional on choice
sequence
conditionals = apollo_conditionals(model,
                                   apollo_probabilities,
                                   apollo_inputs)

# Set path to directory on your PC where the conditionals will be stored
path_cond =
paste0(apollo_control$outputDirectory,sep=.Platform$file.sep,"conditionals
.RDS")
### Save conditionals
saveRDS(conditionals, file = path_cond)
```

Code MNL basic with continuous variables

```
### Step 1: Clear memory
rm(list = ls())

### Step 2: Set working directory for R initialization
setwd(dirname(rstudioapi::getActiveDocumentContext()$path))

### Step 3: Load Apollo library
library(apollo)

### Step 4: Initialize Apollo code
apollo_initialise()

### Step 5: Set core controls
###     ATTENTION: Your inputs must be enclosed in quotes like "this"
apollo_control = list(
  # USER ACTION: Specify model name
  ##    Note: Change the model name for every model that you run
  modelName       = "MNL_basic",
  # USER ACTION: Provide model description
  ##    Note: Change the model description to reflect the current model
  modelDescr      = "basic model",
  # USER ACTION: Specify the column with the respondent id
  indivID         = "peep_ID",
  # USER ACTION: Set path to the folder on your PC where the model results
will be stored
  ##    Note: Use the "outputs" folder that was created by the pre-
processing syntax
  outputDirectory ="outputs"
)

### Step 6: Load data
# Set path to the folder on your PC where the dataset is stored
path_dataset = paste0(getwd(),sep=.Platform$file.sep,"data_with_dummies-
2.csv")

# Load dataset into global environment
```

```
database = read.csv(path_dataset, header=TRUE)

### Step 7: Initialize all parameters that needs to be estimated in your
MNL model
# USER ACTION: The parameters for the attribute "Effectiveness" are
defined. Please,
#              complete the list with parameters that need to be
estimated. Provide
#              names for each parameter following by assigning a starting
value.
apollo_beta=c(b_co2m = 0,
              b_co2h = 0,
              b_price = 0,
              b_wd = 0,
              b_tt = 0)
              #'COMPLETE THE LIST HERE')

### Step 8: Define which parameters (as initialised above) should kept
fixed during estimation (in quotes); if none, keep empty
apollo_fixed = c()

### Step 9: Checkpoint for model inputs
apollo_inputs = apollo_validateInputs()

### Step 10: Define model and likelihood function
apollo_probabilities=function(apollo_beta, apollo_inputs,
functionality="estimate"){

  ### Attach dataset inputs and detach after function exit
  apollo_attach(apollo_beta, apollo_inputs)
  on.exit(apollo_detach(apollo_beta, apollo_inputs))

  ### Create list of choice probabilities P
  P = list()

  ### List of utility functions: these must use the same names as in
mnl_settings (see below), order is irrelevant
  V = list()
  # USER ACTION: Define utility function for alternative 1
  V[["ALT1"]] = b_co2m * Var12.1 + b_co2h * Var13.1 + b_price * price1 +
b_wd * wd1 + b_tt * tt1

  # USER ACTION: Define utility function for alternative 2
  V[["ALT2"]] = b_co2m * Var12.2 + b_co2h * Var13.2 + b_price * price2 +
b_wd * wd2 + b_tt * tt2

  ### Define settings for MNL model component
  mnl_settings = list(
    # USER ACTION: Attach utility functions to the alternatives in your
dataset
    alternatives  = c(ALT1=1, ALT2=2),
    # USER ACTION: Define which alternatives are "available" in each
choice task; in our study, all alternatives are "available"
    avail         = list(ALT1=1, ALT2=1),
```

```
    # USER ACTION: Specify the column containing the chosen alternative;
beware, no dummies are used (!)
    choiceVar     = choice,
    # USER ACTION: Attach list of utility functions
    utilities     = V
  )

  ### Compute choice probabilities using MNL model
  P[["model"]] = apollo_mnl(mnl_settings, functionality) #
functionality="estimate" as the parameters will be updated for estimating
the MNL model

  ### Take product across observations for same individual (i.e.,
considering the panel structure of the data)
  P = apollo_panelProd(P, apollo_inputs, functionality)

  ### Prepare and return outputs of function
  P = apollo_prepareProb(P, apollo_inputs, functionality)
  return(P)
}

### Step 11: Model estimation
model = apollo_estimate(apollo_beta, apollo_fixed, apollo_probabilities,
apollo_inputs)

### Step 12: Print model output with two-sided p-values
#### Note: if one-sided p-values are needed, set "printPVal=1" (p-values
are not reported if set to "0")
modelOutput_setting=list(printPVal=2)
apollo_modelOutput(model, modelOutput_setting)

### Save model output with two-sided p-values
apollo_saveOutput(model, modelOutput_setting)
```

Code MNL with interactions with continuous variables

```
### Step 1: Clear memory
rm(list = ls())

### Step 2: Set working directory for R initialization
setwd(dirname(rstudioapi::getActiveDocumentContext()$path))

### Step 3: Load Apollo library
library(apollo)

### Step 4: Initialise Apollo code
apollo_initialise()

### Step 5: Set core controls
###     ATTENTION: Your inputs must be enclosed in quotes like "this"
apollo_control = list(
  # USER ACTION: Specify model name
```

```
  ##    Note: Change the model name for every model that you run
  modelName       = "MNL_I",
  # USER ACTION: Provide model description
  ##    Note: Change the model description to reflect the current model
  modelDescr      = "MNL with interaction effects",
  # USER ACTION: Specify the column with the respondent id
  indivID         = "peep_ID",
  # USER ACTION: Set path to the folder on your PC where the model results
will be stored
  ##    Note: Use the "outputs" folder that was created by the pre-
processing syntax
  outputDirectory = "outputs"
)

### Step 6: Load data
# Set path to the folder on your PC where the dataset is stored
path_dataset =
paste0(getwd(),sep=.Platform$file.sep,"data_with_dummies_final.csv")

# Load dataset into global environment
database = read.csv(path_dataset, header=TRUE)

### Step 7: Initialize all parameters that needs to be estimated in your
MNL model
# USER ACTION: The parameters for the attribute "Effectiveness" are
defined. Please,
#              complete the list with parameters that need to be
estimated. Provide
#              names for each parameter following by assigning a starting
value.
apollo_beta=c(b_co2m = 0,
              b_co2h = 0,
              b_price = 0,
              b_wd = 0,
              b_tt = 0,
              b_co2m_treehugger = 0,
              b_co2h_treehugger = 0,
              b_price_rich = 0,
              b_wd_age = 0)

### Step 8: Define which parameters (as initialized above) should kept
fixed during estimation (in quotes); if none, keep empty
apollo_fixed = c()

### Step 9: Checkpoint for model inputs
apollo_inputs = apollo_validateInputs()

### Step 10: Define model and likelihood function
apollo_probabilities=function(apollo_beta, apollo_inputs,
functionality="estimate"){

  ### Attach dataset inputs and detach after function exit
  apollo_attach(apollo_beta, apollo_inputs)
  on.exit(apollo_detach(apollo_beta, apollo_inputs))
```

```
  ### Create list of choice probabilities P
  P = list()

  ### List of utility functions: these must use the same names as in
mnl_settings (see below), order is irrelevant
  V = list()
  # USER ACTION: Define utility function for alternative 1
  V[["ALT1"]]  = (b_co2m * Var12.1 + b_co2h * Var13.1 + b_price * price1 +
b_wd * wd1 + b_tt * tt1 +
                  b_co2m_treehugger * d26 * Var12.1 + b_co2h_treehugger *
d26 * Var13.1 + b_price_rich * d30 * price1 +
                  b_wd_age * d27 * wd1)

  # USER ACTION: Define utility function for alternative 2
  V[["ALT2"]]  =(b_co2m * Var12.2 + b_co2h * Var13.2 + b_price * price2 +
b_wd * wd2 + b_tt * tt2 +
                   b_co2m_treehugger * d26 * Var12.2 + b_co2h_treehugger *
d26 * Var13.2 + b_price_rich * d30 * price2 +
                   b_wd_age * d27 * wd2)

  ### Define settings for MNL model component
  mnl_settings = list(
    # USER ACTION: Attach utility functions to the alternatives in your
dataset
    alternatives  = c(ALT1=1, ALT2=2),
    # USER ACTION: Define which alternatives are "available" in each
choice task; in our study, all alternatives are "available"
    avail         = list(ALT1=1, ALT2=1),
    # USER ACTION: Specify the column containing the chosen alternative;
beware, no dummies are used (!)
    choiceVar     = choice,
    # USER ACTION: Attach list of utility functions
    utilities     = V
  )

  ### Compute choice probabilities using MNL model
  P[["model"]] = apollo_mnl(mnl_settings, functionality) #
functionality="estimate" as the parameters will be updated for estimating
the MNL model

  ### Take product across observations for same individual (i.e.,
considering the panel structure of the data)
  P = apollo_panelProd(P, apollo_inputs, functionality)

  ### Prepare and return outputs of function
  P = apollo_prepareProb(P, apollo_inputs, functionality)
  return(P)
}

### Step 11: Model estimation
model = apollo_estimate(apollo_beta, apollo_fixed, apollo_probabilities,
apollo_inputs)
```

```
### Step 12: Print model output with two-sided p-values
#### Note: if one-sided p-values are needed, set "printPVal=1" (p-values
are not reported if set to "0")
modelOutput_setting=list(printPVal=2)
apollo_modelOutput(model, modelOutput_setting)

### Save model output with two-sided p-values
apollo_saveOutput(model, modelOutput_setting)
```

Code Latent Class with continuous variables

```
### Step 1: Clear memory
rm(list = ls())

### Step 2: Set working directory for R initialization
setwd(dirname(rstudioapi::getActiveDocumentContext()$path))

### Step 3: Load Apollo library
library(apollo)

### Step 4: Initialise Apollo code
apollo_initialise()

### Step 5: Set core controls
###     ATTENTION: Your inputs must be enclosed in quotes like "this"
apollo_control = list(
  # USER ACTION: Specify model name
  ##    Note: Change the model name for every model that you run
  modelName       = "first_lc",
  # USER ACTION: Provide model description
  ##    Note: Change the model description to reflect the current model
  modelDescr      = "this estimates lc",
  # USER ACTION: Specify the column with the respondent id
  indivID         = "peep_ID",
  # Define number of cores used during estimation (used to speed up
estimation time)
  nCores          = 5,
  # Define seed used for any random number generation
  seed            = 100,
  # USER ACTION: Set path to the folder on your PC where the model results
will be stored
  ##    Note: Use the "outputs" folder that was created by the pre-
processing syntax
  outputDirectory =
paste(getwd(),'outputs',"practicum_lcmodel_3class_covar",sep=.Platform$fil
e.sep)
)

### Step 6: Load data
# Set path to the folder on your PC where the dataset is stored
path_data =
paste(getwd(),"data_with_dummies_final.csv",sep=.Platform$file.sep)
```

```
# Load dataset into global environment
database = read.csv(path_data, header=TRUE)

### Step 7: Initialise all parameters that needs to be estimated in your
MNL model
# USER ACTION: Define the (1) class-specific and (2) class membership
parameters
#              followed by assigning a starting value. The class-specific
#              alternative specific constant for the opt-out option and
the
#              constants for the class membership models are already
defined.
#              Please, complete the list with the parameters that are
missing.
#              Provide names for each parameter following by assigning a
#              starting value.
apollo_beta=c(# Class 1
  b_co2m_1 = -0.1,
  b_co2h_1 = 0,
  b_price_1 = 0,
  b_wd_1 = 0,
  b_tt_1 = 0,

  # Class 2
  b_co2m_2 = 0,
  b_co2h_2 = 0,
  b_price_2 = 0,
  b_wd_2 = 0,
  b_tt_2 = 0,


  # Class membership - class 1
  delta_1        = -0.1,
  g_treehugger_1 = 0,
  g_ageold_1 = 0,
  g_gender_1 = 0,
  g_educated_1 = 0,
  g_rich_1 = 0,
  # Class membership - class 2
  delta_2        = 0,
  g_treehugger_2 = 0,
  g_ageold_2 = 0,
  g_gender_2 = 0,
  g_educated_2 = 0,
  g_rich_2 = 0)


### Step 8
##  USER ACTION: Complete the list with parameters (as initialised above)
that
##               should be kept fixed during estimation (in quotes)
apollo_fixed = c("delta_2", "g_treehugger_2", "g_ageold_2", "g_gender_2",
"g_educated_2", "g_rich_2")
```

```
### Step 9: Define class membership model
apollo_lcPars=function(apollo_beta, apollo_inputs){
  lcpars = list()
  ## USER ACTION: Complete the empty lists by specifying the missing
class-specific parameters
  ##              which are needed for the class-specific utility
functions

  lcpars[["b_co2m"]] = list(b_co2m_1, b_co2m_2)
  lcpars[["b_co2h"]] = list(b_co2h_1, b_co2h_2)
  lcpars[["b_price"]] = list(b_price_1, b_price_2)
  lcpars[["b_wd"]] = list(b_wd_1, b_wd_2)
  lcpars[["b_tt"]] = list(b_tt_1, b_tt_2)


  ## List of class-membership functions:
  ##  These must use the same names as in classAlloc_settings (see below),
order is irrelevant
  V=list()
  # USER ACTION: Define class-membership function for class 1
  V[["class_1"]] = delta_1 + g_treehugger_1 * d26 + g_ageold_1 * d27 +
g_gender_1 * d28 + g_educated_1 * d29 + g_rich_1 * d30

  # USER ACTION: Define class-membership functions for class 2
  V[["class_2"]] = delta_2 + g_treehugger_2 * d26 + g_ageold_2 * d27 +
g_gender_2 * d28+ g_educated_2 * d29 + g_rich_2 * d30


  ## Define settings for class-membership model
  classAlloc_settings = list(
    # USER ACTION: Attach class-membership functions to the respective
classes
    classes      = c(class_1=1, class_2=2),
    # USER ACTION: Define which classes are "available" in our study, all
classes are "available"
    avail        = 1,
    # USER ACTION: Attach list of class-membership functions
    utilities    = V
  )

  lcpars[["pi_values"]] = apollo_classAlloc(classAlloc_settings)
  return(lcpars)
}

### Step 10: Checkpoint for model inputs
apollo_inputs = apollo_validateInputs()

### Step 11: Define model and likelihood function
apollo_probabilities=function(apollo_beta, apollo_inputs,
functionality="estimate"){

  ### Attach inputs and detach after function exit
  apollo_attach(apollo_beta, apollo_inputs)
  on.exit(apollo_detach(apollo_beta, apollo_inputs))
```

```
  ### Create list of choice probabilities P
  P = list()

  ### Define settings for MNL model component that are generic across
classes
  mnl_settings = list(
    # USER ACTION: Attach utility functions to the alternatives in your
dataset
    alternatives  = c(ALT1=1, ALT2=2),
    # USER ACTION: Define which alternatives are "available" in each
choice task; in our study, all alternatives are "available"
    avail         = list(ALT1=1, ALT2=1),
    # USER ACTION: Specify the column containing the chosen alternative;
beware, no dummies are used (!)
    choiceVar     = choice
  )

  ### List of utility functions for each latent class: these must use the
same names as in mnl_settings (see above), order is irrelevant
  # USER ACTION: Set number of latent classes you are estimating in the
model
  ##       Note: You can call class-specific parameters by
NAME_PARAM[[s]]; see example in ALT3 for the class-specific
  ##             alternative specific constant
  for(s in 1:2){
    V=list()
    # USER ACTION: Define utility function for alternative 1 for class "s"
    V[["ALT1"]] = b_co2m[[s]] * Var12.1 + b_co2h[[s]] * Var13.1 +
b_price[[s]] * price1 + b_tt[[s]] * tt1 + b_wd[[s]] * wd1

      # USER ACTION: Define utility function for alternative 2 for class
"s"
      V[["ALT2"]] = b_co2m[[s]] * Var12.2 + b_co2h[[s]] * Var13.2 +
b_price[[s]] * price2 + b_tt[[s]] * tt2 + b_wd[[s]] * wd2


    mnl_settings$utilities = V
    mnl_settings$componentName = paste0("Class_",s)

    ### Compute within-class choice probabilities using MNL model
    P[[paste0("Class_",s)]] = apollo_mnl(mnl_settings, functionality)

    ### Take product across observations for same individual (i.e.,
considering the panel structure of the data)
    P[[paste0("Class_",s)]] = apollo_panelProd(P[[paste0("Class_",s)]],
apollo_inputs ,functionality)
  }

  ### Compute latent class model probabilities
  lc_settings   = list(inClassProb = P, classProb=pi_values)
  P[["model"]] = apollo_lc(lc_settings, apollo_inputs, functionality)

  ### Prepare and return outputs of function
```

```
  P = apollo_prepareProb(P, apollo_inputs, functionality)
  return(P)
}

### Step 12: Searching for starting value (recommended to ensure model
convergence!)
apollo_beta = apollo_searchStart(apollo_beta,
                                 apollo_fixed,
                                 apollo_probabilities,
                                 apollo_inputs,
                                 searchStart_settings=list(nCandidates=2))

### Step 13: Model estimation
model = apollo_estimate(apollo_beta, apollo_fixed, apollo_probabilities,
apollo_inputs)

### Step 14: Print model output with two-sided p-values
### Note: if one-sided p-values are needed, set "printPVal=1" (p-values
are not reported if set to "0")
modelOutput_setting=list(printPVal=2)
apollo_modelOutput(model, modelOutput_setting)

### Save model output with two-sided p-values
apollo_saveOutput(model, modelOutput_setting)
```

Code Mixed Logit with continuous variables

```
### Step 1: Clear memory
rm(list = ls())

### Step 2: Set working directory for R initialization
setwd(dirname(rstudioapi::getActiveDocumentContext()$path))

### Step 3: Load Apollo library
library(apollo)

### Step 4: Initialise Apollo code
apollo_initialise()

### Step 5: Set core controls
###     ATTENTION: Your inputs must be enclosed in quotes like "this"
apollo_control = list(
  # USER ACTION: Specify model name
  ##    Note: Change the model name for every model that you run
  modelName       = "mixedLogitModel",
  # USER ACTION: Provide model description
  ##    Note: Change the model description to reflect the current model
  modelDescr      = "mixing",
  # USER ACTION: Specify the column with the respondent id
  indivID         = "peep_ID",
  # USER ACTION: Set logical variable to activate estimation of random
parameters
```

```
  mixing          = TRUE,
  # Define number of cores used during estimation (used to speed up
estimation time)
  nCores          = 5,
  # USER ACTION: Set path to the folder on your PC where the model results
will be stored
  ##    Note: Use the "outputs" folder that was created by the pre-
processing syntax
  outputDirectory = "outputs"
)

### Step 6: Load data
# Set path to directory on your PC where the dataset is stored
path_data =
paste0(getwd(),sep=.Platform$file.sep,"data_with_dummies_final.csv")
# Load dataset into global environment
database = read.csv(path_data, header=TRUE)

### Step 7: Initialise all parameters that needs to be estimated in your
Mixed Logit model
# USER ACTION: Define the (1) mu parameters that estimate the sample mean,
and
#                          (2) sigma parameters that estimate the sample
distribution
#              Please, complete the list with the parameters that are
missing.
#              Provide names for each parameter following by assigning a
#              starting value.
apollo_beta=c(
  mu_co2m = 0,
  sigma_co2m = 0,
  mu_co2h = 0,
  sigma_co2h = 0,
  mu_price = 0,
  sigma_price = 0,
  mu_wd = 0,
  sigma_wd = 0,
  mu_tt = 0,
  sigma_tt = 0)

### Step 8:
##  USER ACTION: Complete the list with parameters (as initialised above)
that
##               should be kept fixed during estimation (in quotes); if
none, keep empty
apollo_fixed = c()

### Step 9: Set parameters for generating draws
# USER ACTION: Define the number of one random variable for each sigma in
apollo_beta
# Use the command line interNormDraws

apollo_draws = list(
  interDrawsType = "mlhs",
```

```
  interNDraws    = 200,
  interUnifDraws = c(),
  interNormDraws = c("inter_1","inter_2","inter_3","inter_4","inter_5"),
  intraDrawsType = "mlhs",
  intraNDraws    = 0,
  intraUnifDraws = c(),
  intraNormDraws = c()
)

### Step 10: Create random parameters
# USER ACTION: Write every random coefficient function
# If necessary check the lecture slides
apollo_randCoeff = function(apollo_beta, apollo_inputs){
  randcoeff = list()

  randcoeff[["b_co2m"]] = mu_co2m + sigma_co2m * inter_1
  randcoeff[["b_co2h"]] = mu_co2h + sigma_co2h * inter_2
  randcoeff[["b_price"]] = mu_price + sigma_price* inter_3
  randcoeff[["b_wd"]] = mu_wd + sigma_wd* inter_4
  randcoeff[["b_tt"]] = mu_tt + sigma_tt* inter_5

  return(randcoeff)
}

### Step 11: Checkpoint for model inputs
apollo_inputs = apollo_validateInputs()

### Step 12: Define model and likelihood function
apollo_probabilities=function(apollo_beta, apollo_inputs,
functionality="estimate"){

  ### Attach dataset inputs and detach after function exit
  apollo_attach(apollo_beta, apollo_inputs)
  on.exit(apollo_detach(apollo_beta, apollo_inputs))

  ### Create list of choice probabilities P
  P = list()

  ### List of utility functions: these must use the same names as in
mnl_settings (see below), order is irrelevant
  V = list()

  # USER ACTION: Define utility function for alternative 1
  # Code "effectiveness" and "risk false negative" attributes as numerical
variables
  V[["ALT1"]]  = b_co2m * Var12.1 + b_co2h * Var13.1 + b_price * price1 +
b_wd * wd1 + b_tt * tt1

  # USER ACTION: Define utility function for alternative 2
  # Code "effectiveness" and "risk false negative" attributes as numerical
variables
  V[["ALT2"]]  = b_co2m * Var12.2 + b_co2h * Var13.2 + b_price * price2 +
b_wd * wd2 + b_tt * tt2
```

```
  ### Define settings for MNL model component
  mnl_settings = list(
    # USER ACTION: Attach utility function to the choice alternative in
your dataset
    alternatives  = c(ALT1=1, ALT2=2),
    # USER ACTION: Define which alternatives are "available" in each
choice task
    #              In our study, all alternatives are "available"
    avail         = 1,
    # USER ACTION: Specify the column containing the chosen alternative
    choiceVar     = choice,
    # USER ACTION: Attach list of utility functions
    utilities     = V
  )

  ### Compute choice probabilities using MNL model
  #### functionality="estimate" as the parameters will be updated for
estimating the MNL model
  P[["model"]] = apollo_mnl(mnl_settings, functionality)

  ### Take product across observations for same individual
  ### (i.e., considering the panel structure of the data)
  P = apollo_panelProd(P, apollo_inputs, functionality)

  ## Average across inter-individual draws
  P = apollo_avgInterDraws(P, apollo_inputs, functionality)

  ### Prepare and return outputs of function
  P = apollo_prepareProb(P, apollo_inputs, functionality)
  return(P)
}

### Step 13: Model estimation
model = apollo_estimate(apollo_beta, apollo_fixed, apollo_probabilities,
apollo_inputs)

### Step 14: Print model output with two-sided p-values
#### Note: if one-sided p-values are needed, set "printPVal=1" (p-values
are not reported if set to "0")
modelOutput_setting=list(printPVal=2)
apollo_modelOutput(model, modelOutput_setting)

### Save model output with two-sided p-values
apollo_saveOutput(model, modelOutput_setting)

### Step 15: Estimate individual coefficients conditional on choice
sequence
conditionals = apollo_conditionals(model,
                                   apollo_probabilities,
                                   apollo_inputs)

# Set path to directory on your PC where the conditionals will be stored
```

```
path_cond =
paste0(apollo_control$outputDirectory,sep=.Platform$file.sep,"conditionals
.RDS")
### Save conditionals
saveRDS(conditionals, file = path_cond)
```