READ ME:

This file consists of three read me files, each necessary to obtain the results of my thesis called "Simulating Haemoglobin Concentrations for MISCAN-Colon Using Black-box Machine Learning as a Step Towards Personalised Colorectal Cancer Screening".

The code should be ran, and the read me files should be read, in order of (1) 'miscan_simulation', (2) 'data preprocessing and multiple imputation by chained equations', and (3) 'code to run the models'. All files necessary are provided in this ZIP-file. If you do find something is missing, or if you have any questions about the code, please email me at nykilsdonk@gmail.com.

READ ME: MISCAN SIMULATION

We provide an overview of the name, contents, function, location of each file. All files necessary to run the simulation run for my master thesis are denoted by "THESIS_", all other files serve as prefaces to these files.

Name	Contents	Function	Origin
Original files/ Cohorts_*.csv	CSV file with name, birth year, start year (year of first participation), number of (estimated) invitees, and status	Original file	Hilliene Vandermeer – Health Counsel Analyses – Population effect
Original files/ Cohorts_*.xlsx	Excel file with name, birth year, start year, number of (estimated) invitees, and status for cohorts 1 to 50 (our simulation period)	Preface to THESIS_cohort _size_*.py [includes both table version, comma delimited version, and calculations]	Yoëlle Kilsdonk
THESIS _cohort_ size_*.py	Python script containing precalculated fraction per cohort	Create cohort sizes. The number of invitees is used to determine the fraction of people that should be simulated per cohort in the final run (later on).	Yoëlle Kilsdonk
Original files/ Strategies.csv	CVS file with strategies per cohort with information on age at screening, type of test, chances of participating; chances of participating given they've participated before (defaults to 1 for first invitation), chances of participating given they haven't participated before (defaults to 1 for first invitation), chances of participating in diagnostic colonoscopy (surveillance)	Original file	Hilliene Vandermeer – Health Counsel Analyses – Population effect
Strategies.xls	Excel file with strategies per cohort with information on age at	Preface to THESIS _strategies.csv [includes both	Yoëlle Kilsdonk

	screening, type of test, chances of participating; chances of participating given they've participated before, chances of participating given they haven't participated before, chances of participating in diagnostic colonoscopy (surveillance) for cohorts that started and ended between 2014-2020 (our simulation period)	table version, comma delimited version]	
THESIS_ strategies.csv	CSV file containing the same as above	Input for THESIS_cohort _strategy.py	Yoëlle Kilsdonk
THESIS _cohort_ strategy.py	Python script containing methods to create and fill screening strategy based on cohort information	Input for strategy in CRC screening process in THESIS_MISCAN _simulation.py	Yoëlle Kilsdonk[1]
THESIS _cohort_ Birthyear.py	Python script containing year of individuals in each cohort.	Input to create birth table in birth process in THESIS_MISCAN _simulation.py	Yoëlle Kilsdonk (from Cohorts_*.csv)
THESIS _CRC _data.py	Python script containing data for the CRC process calibrated on NL observations.	Input for CRC process in THESIS_MISCAN _simulation.py	Hilliene Vandermeer – Health Counsel Analyses – Population effect
THESIS _CRC _screening_ data_*.py	Python script containing data for the CRC screening process calibrated on NL observations.	Input for tests in CRC screening process in THESIS_MISCAN _simulation.py	Yoëlle Kilsdonk[2]

[1] Edited from Danica van den Berg – Master Thesis – MISCAN with Hbsim. Changed all keys (green values) in def create_screening_strategy, and changed participation_surveillance key to participation_diag for compatibility with THESIS_strategies.csv file and changes in THESIS_crc_screening2 to account for problem with participation for diagnostic colonoscopy (see teams interaction with Hilliene).

[2] Edited from Hilliene Vandermeer – Health Counsel Analyses – Population effect. Remove adjusted tests and cutoff sensitivity.

THESIS _crc_ screening2.py	Python script as alternative to original panmodel.processes .crc_screening file	CRC screening module	Yoëlle Kilsdonk[3]

To run the simulation, run the "THESIS_MISCAN_simulation.py" in Python. Ensure that all files described above are in the proper working directory. After running the population simulation, you are left with two files named "individual_results_2014_2020_female.csv" and "individual_results_2014_2020_male.csv". These files report information on the cohort number, an ID per cohort, the age at event time, and a tag. The variables *universe* and *element* contain no information in this case.

The tags report when an individual was (first) invited, and if they participated in the test, it also reports the current stage of screen diagnosed cancer. It is also reported when clinical cancer is diagnosed (cancer diagnosis as cause of symptoms). In case a test result is positive, additional tags such as "crc_screening_triage_*" are shown, these are not relevant to our analysis. Finally, the tags also show when someone dies (from other causes or from CRC).

To convert this output to a data set suitable for the Multiple Imputation by Chained Equations (MICE) data set, we wrote the "THESIS_MISCAN_simulation_data_set" file. This file produces two data sets: "MISCAN_simulation_run_female" and "MISCAN_simulation_run_male", which contain an ID number, age at event time, conclusion of the FIT, current stage of cancer, and sex. These two data sets are then used in the input for the MICE algorithm, which is detailed below.

[3] Edited from Public Health – Screening –Pan model 2.0 – panmodel – processes – crc_screening. Changes to log current state and to fix participation of diagnostic colonoscopy. All changes are denoted by ***EDITED*** in the code.

READ ME: DATA PREPROCESSING AND MULTIPLE IMPUTATION BY CHAINED EQUATIONS

We provide an overview of the name, contents, and required input for each file. All files necessary to run the data preprocessing are prefaced by "data_processing_". First run the MISCAN simulation. Then, source the files in this folder in increasing order from 1 to 3, followed by "data_processing_1 – Threshold 15.R" and "data_imputing.R". The final "Data_MICE_imputed" is used as input for the remainder of analyses in this thesis.

Name	Contents	Input
Data_ processing_1.R	This file restructures the original data set from a horizontal structure (per id) to a vertical structure (per round). We make some additional (minor) changes for, i.a., anonymization and storage optimisation.	Dutch colorectal cancer population screening data[1]
Data_ processing_2.R	This file filters out all individuals who did not participate in consecutive rounds (with exception of those who participate only once). It also creates a lagged dependent variable, and the FIT variable which denotes the sequence number of the current FIT, and accounts for aberrant observations.	Data_processing_1.R[4]
Data_ processing_3.R	We construct all additional variables in this file (minimum and maximum)	Data_processing_2.R[1]
Data_ processing_3 _data	Final data set[1]	

Data_ processing_1 – Threshold 15.R	This file restructures the 15-threshold data set from a horizontal structure (per id) to a vertical structure (per round). All individuals without known current stage are deleted, and we make some minor changes for compatibility.	Dutch colorectal cancer population screening data 2014 with threshold 15[1]
Data_ processing_1 _data_15	Final data set[1]	

Data_ Imputing_ final.R	This file contains the code for the Multiple Imputation via Chained Equations method and some minor	Data_processing_3_data, Data_processing_1_data_15, MISCAN_simulation_run_female[5],

[4] We cannot include these data sets due to privacy reasons, so these files are only included in this overview for completeness and cannot be reproduced. For any questions, please inquire me via email.

[5] We've enclosed three versions of this file containing 1 million, 1.5 million, and 2 million individuals respectively. For the final analysis we chose to use the 2 million file, as the imputed data set using this file most closely

	changes to input data sets to ensure compatibility with the method. It also includes code for some descriptive statistics on stage.	MISCAN_simulation_run_male
Data_MICE_imputed	Final data set[1]	

resembled the MISCAN simulation. The code automatically fills the original RIVM data set with these corresponding imputed values.

READ ME: CODE TO RUN THE MODELS

We provide an overview of the name, functions, and descriptions of each file. To run main.py it is required to provide the path where the data is stored (as final_imputed_dataset.csv, see "READ ME: Data preprocessing and Multiple Imputation by Chained Equations"). I've attached a requirements.txt file in this ZIP-file which lists all required packages for your python interpreter to run this analysis.

Name	Function (callable)	Description
main.py	__init__(Path = "V:/UserData/079915/ Thesis -- Data", Tune = 0, Final = 3, nr_folds = 4, Normalize = False, currentrun = 'all', tweedie = False, tweedie_rho = 1.6)	This file first calls 'DataClass.py' to create the necessary folds. Then, based on the value of 'currentrun', 'tune' and 'final', this file initiates tuning of hyperparameters using train and validation folds, and predicting on test folds.
DataClass.py	__init__(Path = None, normalization=False, folds=5, seed=0)	This file is used to create folds. Specifically, we first delete left-over aberrant observations based on age. We then modify the data set to only include one observation per individual and create a stratified 70/30 train/test split. We then create an extra test set containing 10,000 last observations of individuals, using that 70% train set. Lastly, we create nr_folds folds for cross validation using the 70% train set minus the 10,000 observations for the extra test set. This results in one test set containing only new individuals, one test set containing only observations for existing individuals, and nr_folds sets of train/validation folds.
	load_data(path = None)	This function loads the dataset from a pre-specified path as y, X_gaus, X_bin.
	normalize(X, normalization)	This function performs normalization on the continuous variables in the regressors.
utils.py	__init__(model)	This file is initiated only when 'final' in 'main.py' is equal to three, and is made with the intention of retrieving quick stats on the models.
	writecsv(path, suffix='')	This function writes the predictions of the model to a csv file on your device, along with the actual haemoglobin

		values. Suffix can be used to create a suffix to the file name if desired.
	perc_correctlyspec(threshold = 47)	This function calculates the percentage correctly specified based on a certain threshold, i.e., how many predicted values are below (above) the threshold when the true values are also below (above) the threshold relative to the total number of predicted observations.
	perc_deviation(threshold = 5)	This function calculates the percentage of predictions which are within a [true – threshold; true + threshold] interval of the true value.
merfadjusted.py / merfadjusted-ANN.py[6]	__init__(fixed_effects_model=None, gll_early_ stop_threshold=None, max_iterations=20,)	This file is an adjusted version of open source MERF code, which allows for much quicker calculations and requires less memory[7]. This is the core class to instantiate, train, and predict using a mixed effects random forest model. It roughly adheres to the sklearn estimator API. Note that the user must pass in an already instantiated fixed-effects model that adheres to the sklearn regression estimator API, i.e. must have a fit() and predict() method defined.
	predict(X = np.ndarray, Z: np.ndarray, clusters: pd.Series)	This function makes predictions using trained mixed-effects models. For known clusters the trained random effect correction is applied. For unknown clusters the pure fixed effect estimate is used.
	fit(X: np.ndarray, Z: np.ndarray, clusters: pd.Series, y: np.ndarray, X_val: np.ndarray = None, Z_val: np.ndarray = None, clusters_val: pd.Series = None, y_val: np.ndarray = None,)	This function fits the ME model using Expectation-Maximization algorithm.
	get_bhat_history_df	This function retrieves a list of dataframes for the b_hat_history into a multi-indexed dataframe.

[6] The only difference between 'merfadjusted.py' and 'merfadjustedANN.py' is the way the model is loaded in due to 'issues' with Keras. Their functionality remains the same.

[7] User warning: my modified version (as it stands now) takes Z as input, but regardless of what input is given, it always uses a vector of ones as random effects features.

ANN.py	__init__(dataset, tune=False, normalization='none', params={})	If tune is true, a Keras neural network is built and its hyperparameters are tuned using the mean RMSE over nr_folds folds. All negative predictions are casted to be equal to zero. If tune is false, a Keras neural network is built with either the parameters from the tuned model, or user-specified parameters (depending on the input in 'main.py'), which in turn is used to make predictions on both test sets.
MeANN.py	__init__(dataset, tune=False, normalization='none', params={})	If tune is true, a Keras neural network is built and is used as input for the fixed-effects estimation in the mixed-effects model. Its hyperparameters are tuned using the RMSE over one fold (the first to be specific). All negative predictions are casted to be equal to zero. Since training this model requires more memory than provided by the computing facilities at EMC, this file is programmed to write its outcomes to excel in real-time, such that the best model outcome is chosen after the program terminates due to memory error. If tune is false, a Keras neural network is built with either the parameters from the tuned model, or user-specified parameters (depending on the input in 'main.py'), which in turn is used in a mixed-effects model to make predictions on both test sets.
XtremeGBoost.py	__init__(dataset, tune=False, normalization='none', params={}, tweedie=False, p=None)	If tune is true, a XGBoost model is built and its hyperparameters are tuned using either the mean RMSE of mean Tweedie loss over nr_folds folds depending on whether Tweedie = False or True. All negative predictions are casted to be equal to zero. If tune is false, a XGBoost model is built with either the parameters from the tuned model, or user-specified parameters (depending on the input in 'main.py'), which in turn is used to make predictions on both test sets.
MeXtremeGBoost.py	__init__(dataset, tune=False, normalization='none', params={})	If tune is true, a XGBoost model is built and is used as input for the fixed-effects estimation in the mixed-effects model. Its hyperparameters are tuned using the RMSE over one fold (the first to be specific). All negative predictions are casted to be equal to zero. If tune is false, a XGBoost model is built with either the parameters from the tuned model, or user-specified parameters (depending on the input in 'main.py'), which in turn is used in a

	mixed-effects model to make predictions on both test sets.