UNIVERSITY OF CALIFORNIA

Los Angeles

Imputation is a Hyperparameter:
Imputation Deep Learning Model Selection and Evaluation on Large Clinical Datasets

A dissertation submitted in partial satisfaction

of the requirements for the degree

Doctor of Philosophy in Computer Science

by

Davina Jasmine Zamanzadeh

2023

ABSTRACT OF THE DISSERTATION

Imputation is a Hyperparameter:
Imputation Deep Learning Model Selection and Evaluation on Large Clinical Datasets

by

Davina Jasmine Zamanzadeh
Doctor of Philosophy in Computer Science
University of California, Los Angeles, 2023
Professor Majid Sarrafzadeh, Chair

Many real-world datasets suffer from missing data, which can introduce uncertainty into ensuing analyses. To address missing data, researchers have been developing, analyzing, and comparing statistical and machine learning techniques for missing data estimation or imputation. In this context, we built an original framework, Autopopulus, and performed novel analyses that explored predictive pipelines using flexible autoencoder-led imputation. Our work examines autoencoder-led imputation with a deeper regard for the taxonomy of missingness scenarios and mixed feature data of large real-world clinical datasets. In this dissertation we quantify, in a direct manner, the extent to which different methods of imputation affect downstream tasks, and therefore provide rationale for how to choose a solution for a particular dataset and task. We illuminate important decision-making points when assembling a data processing pipeline that handles missing data, while our framework itself allows researchers to apply and compare solutions directly in a unified way for any large dataset. We find that there are different imputation traits under a more granular classification of missingness scenarios, and that trends between imputation performance

superiority and predictive performance superiority do not align. Based on our exploration, we believe that the characterization of missingness in the literature must be expanded and that imputing accurately is not always necessary for predicting accurately. We are just beginning to have a clearer view of just how wide the gap is in our understanding and classification of missingness and have hope that this new information will lead to progress in comprehending both the unknown and the unknowable.

The dissertation of Davina Jasmine Zamanzadeh is approved.

Yizhou Sun

Guy Van den Broeck

Alex Anh-Tuan Bui

Majid Sarrafzadeh, Committee Chair

University of California, Los Angeles

2023

Data can lead to information and information can lead to knowledge and knowledge can lead to power. But sometimes the data are .

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

ACEI angiotensin-converting enzymes inhibitor 39

AI artificial intelligence 1

ARB angiotensin receptor inhibitor 39

ASHA automatic sweep of hyperparameters with asynchronous hyberband scheduling 33, 53

BCE Binary Cross Entropy 19, 20, 33, 34, 52

BRITS bidirectional recurrent imputation for time series 27, 28, 110

CE Cross Entropy 52

CKD chronic kidney disease ix, 37–39, 59, 70, 73

CPU central processing unit 105

CRRT continuous renal replacement therapy viii, x, xi, 4, 82–99, 101, 110, 113

CS Cedars-Sinai 83

CURE-CKD Center for Kidney Disease Research, Education and Hope vii–xi, 3, 4, 37–39, 58, 60, 61, 66, 70, 75–79, 81, 82, 84, 89–92, 97–99, 101, 110, 113

CW column-wise ix, 53–55, 63, 65, 66, 76–78, 98

DAE denoising autoencoder 1, 2, 18, 26, 33, 36, 61, 90, 91

DM diabetes 39

DVAE denoising variational autoencoder 61

eGFR estimated glomerular filtration rate ix, 37–39

EHR electronic health records vii, 1, 2, 4, 5, 7–9, 23–25, 30, 31, 38, 82–84, 101

EW element-wise 52–55, 63, 65, 66, 77, 98

FFNN feed forward neural network 17

FN False Negative 13, 14

FP False Positive 13, 14

FPR False Positive Rate 15

GAIN generative adversarial imputation network 25

GAN generative adversarial network 25, 28

GPU graphical processing unit 16, 105

GRU gated recurrent unit 27

GRU-D gated recurrent unit with decay 27

HTN hypertension 39

ICD International Classification of Diseases 8, 93

KL Divergence Kullback-Leibler Divergence 20, 26, 33, 52

KNN k-nearest neighbors 11, 36, 43, 62, 65, 68, 92, 100, 110

LGBM light gradient-boosting machine 57, 63, 64, 68, 70, 90–92, 99

LR logistic regression 92

LSTM long short-term memory ix, x, 23, 24, 27, 109, 110

MAAPE Mean Arctangent Absolute Percent Error ix, 16, 35, 40, 41, 52, 54, 55, 63, 65, 66, 77, 78, 98

MAE Mean Absolute Error 16

MAPE Mean Absolute Percent Error 16

MAR missing at random 10–12, 24, 31, 32, 35, 40, 43, 44, 46, 48, 60, 61, 68, 99, 110–113

MCAR missing completely at random 10, 11, 24, 26, 31, 35, 40, 43, 44, 46, 48, 60, 61, 67, 68, 99, 110–113

MDL minimum description length 34, 45, 50, 57

MICE multiple imputation by chained equations 11, 36, 40, 43, 44, 62

MIDA Multiple Imputation using Denoising Autoencoders 25, 36, 43

ML machine learning 1, 30, 101, 102

MNAR missing not at random 4, 10–12, 24, 32, 35, 40, 43, 44, 46, 48, 60, 61, 65–68, 99, 110–113

MRI magnetic resonance imaging 5

MSE Mean Squared Error 15, 16, 33

NaN not a number 9, 31, 32, 106

PPV Positive Predictive Value 14, 15

PR-AUC Precision-Recall - Area Under the Curve 15, 36, 37, 40, 43, 44, 68, 70, 90, 91, 93, 95, 97, 99

PSJH Providence St. Joseph Health 39, 59, 73

ReLU rectified linear unit 18, 19, 61

RF random forest 63, 64, 68, 90, 92

RKFD rapid kidney function decline 37–39, 59, 61, 70, 73

RL reinforcement learning 1

RMSE Root Mean Squared Error 16, 35, 40, 41, 54, 63, 65, 66, 98

RNN recurrent neural network vi, ix, 22, 23, 109

ROC-AUC Receiving Operator Characteristic - Area Under the Curve 15, 36, 40, 43, 44, 68, 70, 90, 93, 95, 97, 99

SAITS self-attention-based imputation for time series 28, 110

TN True Negative 13

TNR True Negative Rate 14

TP True Positive 13

TPE tree-structured parzen estimator 93

TPR True Positive Rate 14, 15

uACR urine albumin-creatinine ratio 39

UCLA University of California, Los Angeles 39, 59, 73, 83–85, 87

uPCR urine protein-creatinine ratio 39

VAE variational autoencoder 2, 18–20, 24, 26, 28, 33, 36, 40, 44, 61, 66

XGB extreme gradient-boosted decision tree 92, 93

ACKNOWLEDGMENTS

This work was funded by the UCLA CTSI grant (UL1 TR001881), CKD grant (R01 MD014712), T32 training grant (T32 EB016640), and NIH grants (TL1 DK132768 and U2C DK129496).

I thank my advisor Dr. Majid Sarrafzadeh for seeing my potential and providing me resources throughout my academic career. I appreciate the unlimited freedom I had to pursue the projects that interested me and the encouragement to follow my own passion project when I first started as a PhD student. I thank my mentor Dr. Alex Bui for providing me technical guidance and pushing me to be the best researcher I could be. I do not know where I would be without you. Thank you for pulling me onto interesting and rewarding projects and for helping me find funding throughout most of my PhD. I am eternally grateful to have been under your wing and part of your lab, and made long-lasting connections with you and through you. I thank my clinical mentors Drs. Susanne Nicholas and Ira Kurtz. Your willingness to dive into and learn a new topic and technical savvy impressed me. I am so grateful for the time you invested in me in discussing clinical aspects for my curiosity, or helping me with presentations. Thank you to my committee members Drs. Yizhou Sun and Guy Van den Broeck for being a part of my journey.

I thank all those that I have worked with over the years who mentored me, listened to me, and supported me: Dr. Panayiotis Petousis, Drs. Kenrik Duru and Keith Norris on behalf of the CURE CKD team, and the Medical Imaging and Informatics Group. Thank you Panayiotis for being a mentor and peer to me throughout my entire PhD, for helping me pick my thesis topic, and for always mirroring to me my positive qualities when I felt down and unsure about my research. Thank you Dr. Anders Garlid for also being there for me from the start, I appreciated all of our chats and how you always invested time for me no matter the topic. Thank you to my lab-cousins Henry Zheng, Samir Akre, Al Rahrooh, and Jeffrey Feng for being my labmates from another lab, for collaborating with me on

projects, and for volunteering your time to help me with practice presentations and paper revisions. Thanks to my labmates as well, Ghazaal Ershadi, Shayan Fazeli, Tyler Davis, and Anaelia Ovalle for collaborating with me on projects and running the occassional game of Call of Duty. Thank you Clifford Kravit for all your technical help with AWS, you always moved quickly for me and I am very grateful for that. I thank the KUH-ART committee for welcoming me as an intersectional addition to the fellowship group. I am grateful for the connections I made, shout-out to Meagan Jenkins and Kevyn Hart. Thank you Meagan for accompanying me to Chicago and providing amazing feedback on my work.

I also thank my family and friends for their infinite patience, pride, and encouragement. I thank my parents for supporting me in my endeavor, and encouraging me to be a life-long learner. Thank you to my dad, Dr. Behzad Zamanzadeh, who laughed when I told him I realized I had made a mistake that resulted in three months of experiments going down the drain and that I had to restart. You set the example for me to be interested in learning, to enjoy my PhD, and that sometimes you can only laugh when you are down on your luck. Also, thank you for telling me to take a computer science class when I had an empty summer in high school. Thank you to my mom, for being the honorary Dr. in a family of PhDs, and for expressing interest in my work even though you did not always understand what I was talking about. Thank you for bragging about me and making sure I had no anxieties about having a roof over my head, my health, and having the financial support to live very comfortably. Both of you helped ensure I had as smooth process through graduate school as possible and I am very lucky and grateful. Thank you to my sister, Dr. Nicole Zamanzadeh, who did so much for me and was my biggest cheerleader. Also shout-out for helping me name my framework *Autopopulus*. You were my role model as a child, throughout your own PhD, and consequently still a role model to me now. Thank you to my brother-in-law Jarrett Gorlick for telling me not to do my PhD. I learned from you to proceed with intentionality, which helped me many times during my PhD. I appreciate the skepticism and sense of choice you helped me understand to hold true for myself throughout the process. I thank my uncle

Ramin Nekoukar for encouraging me to be a scientist my whole life and always engaging with curiousity for my work.

I thank this important quote: "Suficiente! Ahora hablara. Corre, corre, corre, que nadie te pueda alcanzar, no me podras atrapar, soy el hombre de jengibre! *Eres un monstruo!* El único monstruo aquí eres tu! Tu, y esos personajes de cuentos de hadas que arruinan mi mundo perfecto. Ahora dime, donde estan los otros? *Cerdo!* He tratado de ser paciente con ustedes pero mi paciencia a llegado a su limite. Dime! O te arrancare. *No! Mis botones, no! Mo mis botones de gomita.* Entonces, cuentame! Quién los oculta? *De acuerdo, te lo cuento. ¿Tú conoces a pin pon?* ¿A pin pon? *Si pin pon.* Si, es un muñeco muy guapo y de cartón. *Si, se lava su carita con agua y con jabón.* ¿Con agua y con jabón? *Si, se lava la carita!* Se lava la carita con agua y con jabón."

Thank you to my PhD-brother, Eli Jaffe, who has made it with me from start to finish, only defending five days apart. You have become my lifelong friend and have been a strong pillar of support and understanding throughout this process. I valued our support-group sessions with Paul Lou, and being menaces to our professors when we took the same classes. Thank you to my best friend and cousin Ashley Selki for being my partner in crime and always being interested in updates for my PhD and reminding me to be proud of my work. I thank all my not-yet-mentioned cousins, my tribe, all the Selkis and the Zamanzadehs. Thank you to my best friend Danielle Robinson for also being there to laugh with me, complain with me, and encourage me to get through the downturns. Thanks to my friend Bijan Semnani for being the first to star Autopopulus on Github. Thank you to all my friends who allowed me to have fun and celebrated wins with me! The list includes, but is not limited to: Ariana Sedighpour, Kayvon Benyamin, Dr. Kevin Hakimi, Andre Askarinam, Rachel Kamran, Ariel Dankner, Dr. Brenda Asilnejad, Yasmine Mzayek, Maggie Marazita, Andrea Alcaraz, Iliana De Hoyos, Andy Williams, Michelle Rosen, Ashley Hannani, Jasmine Gass, Daniel Raban, Andrea Alcaraz, Chris Fiore, Jessie Kahle, Nima Gadhimi, and Morgan Trezek. I thank (and blame!) my previous mentors, ex-labmates, and lifelong friends from UCSB, Dr. Kyle

Dewey and Mika Gavrilov for kickstarting my academic career. Thank you Mika for pushing me to be better, lending me an ear, and reminding me of my strengths. Also shoutout to my Overwatch crew for winning and losing games with me, and more importantly, keeping me sane when I worked late nights and odd hours.

VITA

2022–2023	National Institutes of Health Kidney, Urologic, and Hematologic-Advanced Research TL1 Training Grant; University of California, Los Angeles.
2021	Master of Science in Computer Science; Department of Computer Science, University of California, Los Angeles.
2018–2020	T32 National Institutes of Health Training Grant; University of California, Los Angeles.
2018–2019	Graduate Dean's Scholar Award; Department of Computer Science, University of California, Los Angeles.
2018	Bachelor of Science in Computer Science; Summa Cum Laude; Department of Computer Science, University of California, Santa Barbara.
2017–2018	Undergraduate researcher for the Programming Languages Lab; Department of Computer Science, University of California, Santa Barbara.

PUBLICATIONS AND PRESENTATIONS

Zamanzadeh, D., Feng, J., Petousis, P., Vepa, A., Bui, A., Kurtz, I. "Improving continuous renal replacement therapy outcome predictions with machine learning" *Manuscript in preparation for Nature Portfolio.*

Zamanzadeh, D., Bui, A., Sarrafzadeh, M. "Autoencoders for Imputation: A Closer Look at Mixed Feature Imputation Under Different Missingness Mechanisms." IEEE Transactions on Neural Networks and Learning Systems. *In Submission.*

Schouten, R., **Zamanzadeh, D.**, Singh, P. "pyampute: a Python library for data amputation." *21st Python in Science Conference*, 2022

Wong, M.S., Wells, M., **Zamanzadeh, D.**, Akre, S., Pevnick, J.M., Bui, A.A.T., and Gregory, K.D. "Applying Automated Machine Learning to Predict Mode of Delivery Using Ongoing Intrapartum Data in Laboring Patients." *American Journal of Perinatology*, December 2022.

Zamanzadeh, D.J., Petousis, P., Davis, T.A., Nicholas, S.B., Norris, K.C., Tuttle, K.R., Bui, A.A.T., and Sarrafzadeh, M. "Autopopulus: A Novel Framework for Autoencoder Imputation on Large Clinical Datasets." *Annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE Engineering in Medicine and Biology Society. Annual International Conference*, **2021**:2303–2309, November 2021

Fazeli, S., **Zamanzadeh, D.**, Ovalle, A., Nguyen, T., Gee, G., and Sarrafzadeh, M. "COVID-19 and Big Data: Multi-faceted Analysis for Spatio-temporal Understanding of the Pandemic with Social Media Conversations.", April 2021.

Zamanzadeh, D., Petousis, P., Davis, T.A., Garlid, A., Wang, X., Norris, K.C., Duru, K., Tuttle, K.R., Bui, A.A.T., Nicholas, S.B. "Using autoencoders for imputing missing data in estimated glomerular filtration rate (eGFR) decline trajectories of patients with chronic kidney disease (CKD)". *American Society of Nephrology (ASN) Kidney Week*, 2020.

Hosseini, A., **Zamanzadeh, D.**, Valencia, L., Habre, R., Bui, A.A.T., and Sarrafzadeh, M. "Domain Adaptation in Children Activity Recognition." In *2019 41st Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pp. 1725–1728, 2019.

CHAPTER 1

Introduction

The widespread adoption of electronic health records (EHR) has ushered in a new age of data-driven medicine. Researchers are beginning to explore artificial intelligence (AI)-based methods (e.g., machine learning (ML); reinforcement learning (RL)) to provide new insights from the growing number of patient records. However, the variation in healthcare delivery complicates the analysis of such datasets, especially resulting issues like missing data. Many real-world datasets suffer from missing data, which can introduce uncertainty into ensuing analyses. Furthermore, mathematical models cannot intrinsically handle missing data or do not manage missing data well.

Among the possible approaches to handling missing data, the most advisable and most used one is generally imputation, or the estimation of missing values. However, in the broader view of data processing pipelines, imputation is a step that is either glossed over or handled in only one specific way with much scrutiny. Of the growing body of methodologies to handle imputation, deep learning models such as autoencoders have gained popularity. Autoencoders are good candidates for imputation due to their ability to impute on big data (such as EHR datasets) as a whole at once, quickly learn multi-variate and nonlinear relationships between the data, and operate on mixed features—both continuous and categorical. However, despite the appeal of autoencoders for imputation, there are no theoretical proofs on their behavior. Most autoencoder-led imputation methods developed only deal with particulars: they define a specific arrangement of an autoencoder (e.g., denoising autoencoder (DAE)) on a particular dataset with only a singular type of data (i.e., only continuous data).

However, real-world data, and particularly EHR data, are never so confined; variables are a combination of continuous and categorical and there may be a mix of longitudinal and static variables. For example, a patient may have multiple systolic blood pressure measurements (a continuous longitudinal variable), or multiple risk level assessments (a categorical longitudinal variable), but also a single age (a continuous static variable) and ethnicity (a categorical static variable).

Autoencoders encompass a wide range of models; their constraints are that the network must be symmetric, and the output should closely resemble the input without memorizing it. When evaluating autoencoders as a class of imputation methods on tabular data, the same way we consider the different configurations of the variables in the data, we must consider different configurations of autoencoders such as DAEs or variational autoencoders (VAEs). We may look at particular types to compare to each other, but we may also look at their similarities to draw a picture of autoencoders as an umbrella class of networks. We would also like to efficiently compare different autoencoder methods, in addition to assisting in identifying the best technique for the given dataset and task at hand. We provide a series of flexible frameworks that enable us to:

- Highlight the variability of imputation performance across models and datasets, and thus treating imputation as a hyperparameter to tune.

  To many, imputation is a fixed step in the data wrangling pipeline. However, the line between imputation and estimation is thin. If we view imputation as prediction of missing values, rather than just a menial data-processing step, it would make sense that the process for choosing a model would need tuning the same way one might choose a model for a downstream predictive task. Most works in the literature do not view imputation this way. We investigate and show that the choice of imputation model depends on the dataset, task, and downstream model for that task, and ultimately the choice needs to be tuned to each scenario.

- Establish an empirical profile of autoencoders (across their spectrum of configurations) that work on all types of static data for imputation via a unified training and evaluation pipeline.

  While there is no theoretical understanding established of autoencoders as imputation models, we begin to understand them by empirically evaluating and comparing different versions of them. To our knowledge, no other work has approached autoencoders as a whole for imputation to characterize them. Rather, most works develop a particular autoencoder model and pit it against other methods in a competition, or compare autoencoders widely for other tasks. Additionally, we begin to establish which characteristics of datasets are useful for imputation model selection. We focus on static clinical datasets, as opposed to longitudinal ones, which would require an equal, if not greater, amount of analysis and effort compared to this work. However, we enable future work to expand this analysis into the time domain.

- Enable researchers and applied practitioners to easily compare and choose a deep learning imputation model that best suits their dataset.

  If there are many options for autoencoders that need to be evaluated widely and tuned, we ultimately need to select the model that best lends itself to the current task. The frameworks developed in this work then assist in autoencoder model selection that best suits the data and task by making evaluating and comparing them easy.

The dissertation is organized as follows. In Chapter 2, we provide terms and concepts necessary in order to understand our work. We discuss concepts about data, imputation, and machine and deep learning, before summarizing the current state of imputation as it relates to our context. In Chapter 3, we dive into why we created *Autopopulus*, our first attempt at building the framework and its pipelines, demonstrate it on the Center for Kidney Disease Research, Education and Hope (CURE-CKD) dataset, and highlight what we learned through this process. We review our experimental pipeline which first evaluated the

imputation performance of various imputation methods under controlled but simple missingness scenarios, and then separately evaluated the predictive performance after training and applying an imputation model on the entire semi-observed dataset. Using what we learned from our first iteration of Autopopulus, we created a fully mature form of the framework. In Chapter 4, we highlight why we needed to make changes to Autopopulus, what changes we made, and demonstrate our framework on an updated version of the CURE-CKD dataset. This time, we consider more parameters of the imputation task and we look at those parameters with more granularity and scrutiny. We review our experimental pipeline which now split the missing not at random (MNAR) scenario into three different ones: a non-recoverable MNAR mechanism, and two recoverable ones using latent features drawn from a categorical and continuous distribution, respectively. We also explore different feature mappings so that the data contained mixed features (continuous categorical), purely continuous features, or purely categorical features. This time, we primarily analyzed predictive performance under each missingness scenario, rather than on the entire dataset as-is. To bridge the gap, we would evaluate predictive performance of the best imputation models on the remaining semi-observed subset of data. In Chapter 5 we apply Autopopulus to a raw real-world electronic health records (EHR) dataset, the continuous renal replacement therapy (CRRT) dataset, in contrast to the curated EHR-derived CURE-CKD dataset. Combining our analyses on both datasets, we draw a profile of autoencoders. Finally, in Chapter 6 we share what we learned from our journey in imputation and how we best believe one could continue to expand on this work.

CHAPTER 2

Background

With the ever-growing availability and capability to capture, store, and compute on data, data-driven analysis, and specifically machine learning, has become a prominent tool and school of thought in science. Machine learning techniques are tools that are meaningless in-and-of themselves, as they rely on *data*. Consequently, the field and sub-fields of machine learning largely exists in a multidisciplinary or intersectional context, depending on the domains of data in which they dwell. While our methodology can be applied to a wide variety of domains, we evaluated our work on two tabular clinical datasets. Clinical data can present in many forms: tabular data such as electronic health records (EHR) or sensor data from wearables, imaging such as magnetic resonance imaging (MRI), free text such as doctor's notes, or even audio recordings such as those from an ultrasound. Our datasets in particular are EHR or EHR-derived data, and therefore our methodology revolves primarily around tabular data and missingness in tabular data. Below we will dive into the structure of our data, what missingness looks like in those data, relevant machine learning and deep learning concepts and techniques, followed by relevant imputation methods.

2.1 Data

2.1.1 Tabular Data

Tabular data contains entries (referred to as *observations*, *samples*, or *examples*) as rows with various *features*, or equivalently, *variables* as the columns. The variables can be of various

types. We typically classify features based on their ranges, or the set of possible values they can take.

- *Categorical* features, sometimes referred to as *qualitative* or *nominal* features, can take on a discrete finite set of values. They include:
  - *Binary*: features that can take on only two values, usually $\{0, 1\}$, or $\{no, yes\}$ semantically. An example of a binary feature is whether a patient is an adult or not.
  - *Ordinal*: features that can take on a discrete, finite set of values that are ordered. An example of an ordinal feature is a risk level that could be low, medium, or high.
  - *Multicategorical*: features that can take on 3 or more discrete values that have no relative order. An example of a multicategorical feature is a patient's ethnicity.
- *Continuous* features, sometimes referred to as *quantitative* features, can take on an infinite set of values, usually numerical. Some examples of continuous feature are a patient's age or height.

Datasets can be classified in many ways, we enumerate the most relevant ones to this work here:

- *Mixed*: data contain features that are both categorical and continuous. They are sometimes also referred to as *heterogeneous* data, which is overloaded to also refer to *multimodal* data, which include data from multiple modalities such as images, text, audio, etc. For our purposes, we exclusively refer to homogeneous data as those that contain categorical and continuous features.
- *Longitudinal*: data where features are sampled or measured multiple times, at or not at regular intervals. Longitudinal data are equivalently referred to *time-series* or *sequential* data. Longitudinal data may also contain either continuous or categorical features.

We may refer to features themselves as longitudinal as shorthand. An example of longitudinal data is multiple measurements of a patient's blood pressure. *Variable-length* data arise when a given feature is measured a different number of times per example.

2.1.2 Electronic Health Records (EHR)

Clinical and health data are collected in what is called electronic health records (EHR). The EHR behave as a digital patient chart including patient's medical history, diagnoses, medications, treatments, allergies, labs, etc. While data can be collected in any form (e.g., images, text, audio), a large portion of EHR consists of tabular data.

Structure Typically, the EHR may contain a single row per patient or a single row per event in a hospital, where an event may be, but is not limited to, a diagnosis, procedure, or prescription. Features may be continuous (a real value such as age, or weight), or categorical (a discrete value such as sex or ethnicity). They could also be repeated measurements, which are considered series or longitudinal data, like repeated blood pressure measurements. We differentiate between longitudinal data and those that are only observed or measured once (e.g., sex) by referring to the latter as static data. EHR data may consist of both static and longitudinal data, such as a table with patient demographic information and other tables for labs that may be taken multiple times at different points in time. Longitudinal data, or sequential data may be variable per sample or per patient, which is commonly referred to as variable length or unequal length sequences. For example, in a study that draws labs from patients at 3, 6, and 12 months, some patients may only participate at one time point, or only have 2 of 3 measurements.

Although the EHR has modernized healthcare and enabled a new era of medical analytics, analyzing the consequent data is almost always a complex process. Datasets tend to be afflicted with various issues such as missing data or highly correlated features. To properly assess and combat these challenges, it is necessary to understand how EHR are generated

and why we may see such characteristics (e.g., missing data).

Compared to the history of healthcare systems globally, the EHR is a relatively new concept. Namely, in the United States, the EHR had approximately a 20.8% adoption rate across office-based physicians in 2004 [Hea19]. Due to former President Bush's and former President Obama's initiatives in 2004 and 2009 respectively, the EHR adoption rate amongst office-based physicians expanded to 85.9% by 2017 [Ath11], and adoption amongst non-federal acute care hospitals rose from 12.2% in 2009 to 83.8% by 2015 [Doe]. However, the rapid adoption of the EHR was just one battle in the overall effort towards modernizing healthcare. Adjusting to the EHR systems (e.g., converting physical records for millions of patients, training healthcare providers to use the EHR systems) in the early years of adoption led to the corresponding data from those time periods to be less reliable and more error-prone. Learning EHR systems and constant documentation required of healthcare providers has proven to be a continued burden [CKH18]. This may lead to errors or gaps in data as medical professionals struggle to juggle their role as care providers and additionally to record everything that transpires.

In addition to the recentness of the EHR, issues in analysis arise given variations in healthcare delivery and particularly in data capture. Although there are efforts towards standardization for medical care (e.g., sliding-scale protocol for regular insulin use [RHJ04]) and EHR structuring/coding (e.g., International Classification of Diseases (ICD) codes), there is no one way to provide care for a patient, and no one way to document their history and current processes of care. Data themselves may physically look different in datasets across healthcare institutions or even within a single institution, for example, due to usage of different encodings (e.g., ICD-9 vs ICD-10), different standards for what information is required to be documented, or different software for recording information (e.g., manual vs. automatically parsed dictation) [CKH18]. Data distributions may shift within a dataset due to a constant flux of healthcare providers (physicians, nurses, technicians, etc.) on a day-to-day basis. Some physicians may be more or less strict about putting a patient on a certain

type of care, or different hospitals may impose a more or less strict protocol on particular procedures.

2.1.2.1 Nephrology

The particular domain of medicine in which we analyze EHR is nephrology. *Nephrology* is the study of the kidneys. Normally, most people have two kidneys, and they are recognizable by their bean-like shape. The kidneys are the chemists of the body: they work to keep all the chemicals in your blood balanced by filtering the blood for waste, excess fluids, and regulate compounds such as potassium.

2.2 Missing Data

Missing data plague most datasets across multiple domains and can occur for a wide variety of reasons: human error, machine error, random chance, etc. Missing data introduces uncertainty that affects any downstream tasks on those datasets, whether it be statistical analysis or prediction. Additionally, many mathematical models cannot handle missing data well, if at all. Missing data are stored or referred to as not a number (NaN) so that they may be treated differently computationally.

One way missing data are commonly dealt with is by removing the rows or features with missing data. Dropping observations with missing data poses issues such as: reducing the dataset size significantly or introducing bias by limiting the dataset to observations with features that are well-populated but may not be randomly distributed. For example, in an EHR dataset, sicker patients require more tests and visits than healthier patients and are less likely to have missing values, so dropping patients with missing values might bias the dataset towards these less healthy patients. These biases can be pernicious, particularly for data disadvantaged populations (e.g., lower income patients miss appointments more often due to lack of access to care, transportation, etc.) [GC20]. Alternatively, removing features

with missing data is not always tenable, as those features might be important predictive factors.

Another common approach to handling missing data is substituting missing values with estimates of the missing values, which is known as *imputing*. While imputation seems more appealing than dropping potentially useful information, using the wrong imputation method may degrade the quality of prediction. To properly impute missing data, it is important that the method impose a reasonable assumption about the missingness mechanism for the data. The literature follows the missing mechanism paradigm of [Rub76] that divides all cases of missingness into three categories: missing completely at random (MCAR), missing at random (MAR), and missing not at random (MNAR). Data are MCAR when the probability of an observation missing for a particular variable is unrelated to any other variable, observed or unobserved, meaning, there is no systematic difference between samples that are missing data and those that are not [MSW18]. For instance, equipment fails for a day and the values are lost. Data are MAR when the probability of data missing for a particular variable is related to one or more observed variables in the dataset. For instance, in the United States, patients under the age of 21 years may be less likely to report alcohol use to their care providers. In this scenario, age is observed, which explains a trend over the missing values for alcohol use. Data are MNAR when the probability of data missing for a particular variable is related to either the unobserved value itself or another unobserved variable. For example, a patient refuses testing due to religious reasons, but religion is not recorded. In this scenario, the patient's religion explains the missing test results, but the religion itself it unobserved. Another example of MNAR may be that the equipment does not register values over 100. In this scenario, the true value of the missing entry itself (being some value over 100) explains the missing data.

Though we maintain that is important to impute according to the mechanism under which the missingness occurs, it is not a simple task. One issue is that real-world datasets are complicated and do not follow a single missingness mechanism. Features may be missing due

to the influence of a combination of other features, or certain features may be missing under different mechanisms. Even with making the simplifying assumption that a dataset is missing data under only one particular mechanism, another complication arises with distinguishing between different mechanisms within a given dataset. Currently, it is only possible to test for MCAR in a dataset using Little's MCAR test [Lit88]. It is not possible to distinguish between data MAR and MNAR.

Additionally, the understanding of missingness in the domain of data analysis while not nascent is not necessarily robust. There is only one paradigm available of understanding missingness in data, which to our knowledge has not been challenged until recently. More researchers are beginning to expand on and reconsider this existing paradigm of reasoning about missingness. In particular, Mohan et al. demonstrated that there is a partition of the space of MNAR scenarios that have distinct missingness characteristics [MP21]. There are certain MNAR scenarios that are *recoverable*, where the missing value can be estimated consistently from observed data, while others are not recoverable and there are no algorithms that can provide that guarantee.

2.3 Imputation

Many imputation techniques exist and are frequently used. The simplest forms of imputation are those such as mean or mode, (stochastic) regression, k-nearest neighbors (KNN), and carry forward/backward imputation. While these are simple, fast, and easy to implement, they underestimate standard error by reducing variability and ultimately produce biased estimates [End10].

Another approach involves maximum likelihood and/or multiple imputation, such as multiple imputation by chained equations (MICE). Multiple imputation involves repeatedly imputing the same values multiple times to account for the uncertainty in single point estimates for missing values. While these methods are guaranteed to produce unbiased estimates

under MAR, they are computationally expensive and time-consuming and will produce biased estimates under MNAR. Existing methods that can model data MNAR in addition to MAR, such as pattern mixture models [FHB17], rely on Monte Carlo methods that are computationally slow.

2.4 Machine Learning

Machine learning is the process of using data and a learning algorithm to train a model to generalize to unseen data on a given task. Common tasks include, but are not limited to, (binary or multi-class) classification, regression, or clustering. Towards this end, models undergo two phases in their lifetime: 1) training or learning, and 2) inference. During training, the model parameters change to adjust its performance on its given task. During inference, data are simply passed to the model for its output, and the model does not change.

There are many types of learning algorithms. At a high level, learning algorithms are categorized depending on when the learning takes place, and how it might (or might not) use labels. With *offline* algorithms, learning takes place before the execution of the task. In contrast, with an *online* algorithm, learning takes place as a task progresses. *Supervised* learning algorithms provide labels in addition to the data, where the labels are the *ground truth*, or the correct answers for each sample. *Semi-supervised* learning algorithms provide only some labels, while the rest are unknown. *Reinforcement* learning algorithms provided occasional and delayed information. Lastly, *unsupervised* learning algorithms provide no direct labels.

There are also different types of models. A *discriminative* model simply learns to produce an output based on existing data as input in order to make a prediction or estimation. On the other hand, a *generative* model can produce outputs that do not rely on existing data as an input. Both require data to train, but the latter can generate new data.

2.4.1 Evaluation of Performance

What makes a model "good" largely depends on the context of a given task and dataset. Many metrics for evaluating models exist that capture different notions of "performance". To produce a robust evaluation of a given model, in addition to proper data handling, it is important to choose appropriate metrics that can capture performance from different angles. Keep in mind, the metrics used to evaluate a model may not be the same metrics used to aid in training the model.

Binary classification. Binary classifiers aim to discriminate between examples in the *positive* class and *negative* class. By way of illustration, a binary classifier might classify (or discriminate between) patients *with* cancer and patients who do not have cancer. The choice for which class is the *positive* class is entirely context-dependent. For example, with cancer prediction, it may make more sense for a patient with cancer to be a positive example. Binary classifiers output either a 1 or 0, or *yes* or *no* for the positive class. Therefore, the classifier can, for a given positive instance, predict *yes* correctly (True Positive (TP)), or *no* incorrectly (False Negative (FN) or Type II Error). Similarly, for a given negative instance, the classifier can predict *no* correctly (True Negative (TN)), or *yes* incorrectly (False Positive (FP) or Type I Error). A Type I Error (FP) can be thought of as a false alarm, while a Type II Error (FN) can be thought of as failing to alert. We can evaluate binary classification tasks using the following metrics and highlight their strengths, where the abbreviated names represent the *count*, and P_{est} and N_{est} are the counts of the positive class and negative class respectively that were predicted by the model:

- *Accuracy* measures how many samples (regardless of if they are positive or negative) were guessed correctly. It is useful if the dataset is *balanced* (almost equal number of positive and negative samples in the dataset). If a dataset is *imbalanced* (few positive examples), it is possible to cosmetically achieve a high accuracy if the model guesses

everything is negative. Accuracy can be computed as

$$\frac{TP + TN}{P + N}$$

- True Negative Rate (TNR) or *Specificity* measures, of the instances that were truly negative, how many did the model predict / believe to be negative. Formulated as a probability, Pr(Predict Negative | Is Negative). Specificity focuses on the negative class and penalizes FP. Specificity can be computed as

$$\frac{TN}{N} = \frac{TN}{TN + FP}$$

- True Positive Rate (TPR) or *Sensitivity* or *Recall* measures, of the instances that were truly positive, how many did the model predict to be positive. Formulated as a probability, Pr(Predict Positive | Is Positive). Recall focuses on the positive class and penalizes FN more heavily, compared to penalizing FP. This renders recall as a useful metric for imbalanced datasets. Recall can be computed as

$$\frac{TP}{P} = \frac{TP}{TP + FN}$$

- Positive Predictive Value (PPV) or *Precision* measures, of the instances the model predicted to be positive, how many were guessed correctly. Formulated as a probability, Pr(Is Positive | Predict Positive). Precision focuses on the positive class and penalizes FP more heavily. This renders precision as a useful metric for imbalanced datasets. Precision can be computed as

$$\frac{TP}{P_{est}} = \frac{TP}{TP + FP}$$

- *F1 Score* is the harmonic mean of precision and recall, which, compared to the arithmetic mean, punishes extreme values more. There is a variant that allows a hyperparameter to dictate how much more to weight recall over precision. F1 Score can be computed as

$$\frac{2 \cdot TP}{P + P_{est}}$$

- Receiving Operator Characteristic - Area Under the Curve (ROC-AUC) measures the area under the curve of plotting the ROC (plots TPR vs FPR) across different probability thresholds for the positive class. The ROC-AUC is equivalent to the probability of correctly ranking a positive example above a negative example.

- Precision-Recall - Area Under the Curve (PR-AUC) measures the area under the curve of plotting the precision vs. recall (plots PPV vs TPR) across different probability thresholds for the positive class.

- *Brier Score* measures the *calibration* of a model. When a model is well calibrated, you can interpret the predicted probability as a confidence level (e.g., 85% of samples with probability of 85% of being in the positive class should actually be positive). Brier score suffers a similar problem to accuracy on imbalanced datasets, where the model can inflate its score by naively predicting the majority class with 100% probability. Note though that it acts as a loss, so compared to the other metrics, a lower Brier score is better. Brier score, for N samples, predicted label $\hat{y}_i$, and true label y_i, can be computed as

$$\sum_{i=1}^{N}(Pr(\hat{y}_i) - y_i)^2$$

Regression. Regression models aim to predict continuous values (as opposed to the discrete *yes* or *no* of binary classifiers). For example, a regression model may predict a person's height, or the dollar value of a United States stock. In contrast to binary classifiers which can be "right" or "wrong", regression models are evaluated as a matter of *distance*, or how close the predicted value ($\hat{y}$) is to the true one (y).

- Mean Squared Error (MSE) computes the squared difference between the true and estimated label. The squared term penalizes larger aberrations from the true value more. Optimizing on squared values results in finding the mean error, which is sensitive

to outliers. MSE can be computed as

$$\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

- Root Mean Squared Error (RMSE) is a smoothly differentiable function that computes the sample standard deviation of the error terms. RMSE can be computed as

$$\sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}$$

- Mean Absolute Error (MAE) is a non-differentiable, but more interpretable error. Optimizing on absolute value results in finding the median error, which is more robust to outliers. MAE can be computed as

$$\frac{1}{N}\sum_{i=1}^{N} \mid y_i - \hat{y}_i \mid$$

- Mean Arctangent Absolute Percent Error (MAAPE) is an extension of the Mean Absolute Percent Error (MAPE), which is scale-independent and interpretable, that is less biased and well-behaved when values are zero or near zero [KK16a]. MAAPE can be computed as

$$\frac{1}{N}\sum_{i=1}^{N}\arctan\left(\left|\frac{y_i - \hat{y}_i}{y_i}\right|\right)$$

2.5 Deep Learning

Deep learning is a sub-branch of machine learning that has added representation learning (feature extraction and dimension reduction) in the process. It requires more data and computation power and focuses on network architectures with multiple layers. Deep learning has become popular for large datasets due to their ability to: scale to large sample sizes, harness the now-more-accessible graphical processing unit (GPU) for efficient computation, and formulate complex relations in the data such as non-linear dependencies or multivariate time-series data [GBC16].

2.5.1 Neural Networks

Neural networks form the basis of many deep learning architectures. They are modeled after the network of neurons of the human brain, which connect to each other in complex webs to share and pass information as computational units. In neural networks, a neuron is a computational unit that takes a set of inputs of N entries and D features, a weights matrix that maps all N samples from D dimensions to D' dimensions, a bias vector that behaves as class priors for all N samples, and outputs a set of scalars, or activations, for all N samples. The weights and biases are parameters that are learned by the network via an optimization procedure and loss function. Optimization procedures are algorithms that dictate how a network might learn or update its weights. A loss function is a penalty function that is used when the model is incorrect to drive weight updates.

2.5.1.1 Activation Functions

As is, feed forward neural networks (FFNNs) can only learn linear functions or boundaries. This is because each layer computes linear combinations of the input and weights, and a linear function composed with a linear function is still linear. Additionally, particularly with deep networks, the extra layers could be reduced to a single linear transform, rendering useless the utility of multiple layers being able to approximate more complex functions. Activation functions allow us to introduce non-linearity into neural networks. The only time linear-activation (or identity) functions are useful is if the output variable or target is a continuous real number, warranting use in the output layer.

Commonly used activation functions are:

- **Sigmoid activations**: convert any real number into a value between 0 and 1 (inclusive), which can be interpreted as probabilities. The mean of the activations will be close to 0.5. Aside from introducing non-linearity, the sigmoid activation function is useful for the output layer if a network's task is binary classification. The sigmoid

function, notated as $\sigma(\cdot)$ can be computed as

$$\sigma(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}$$

- **Rectified linear unit (ReLU) activations**: bottom-cap the values at 0. While this does not produce values that could be interpreted as probabilities, it is more likely to create sparse representations, less likely to suffer from the vanishing gradient problem, and is computationally faster. The ReLU function can be computed as

$$ReLU(x) = max(0, x)$$

2.5.2 Autoencoders

Autoencoders [Lea] are a type of neural network that are used to recreate the original dataset without memorizing the data, essentially aiming to learn an approximation of the identity function. Formally, an autoencoder $f(\cdot \mid W, b)$ tries to learn weights and biases W, b such that for an input x, $f(x) = \hat{x} \approx x$. Autoencoders, as depicted in Figure 2.1, are symmetric networks with multiple layers, composed of two halves: an *encoder* and a *decoder*. The encoder outputs a *latent representation*, or *code*, of the data. The latent representation is a "hidden", compressed representation of data, which is hopefully more meaningful or useful towards downstream tasks. The code is then passed to the decoder, which attempts to reconstruct the original input. Formally, with encoder $e(\cdot)$ and decoder $d(\cdot)$, the autoencoder computes $f(x) = d(e(x))$.

Autoencoders have evolved greatly since their inception, and their architecture can be adjusted to achieve different results. An autoencoder is *undercomplete* when the code is smaller than the input, acting as lossy compression, and is *overcomplete* when the code is larger than the input. A denoising autoencoder (DAE) [VLB08] is a type of autoencoder that partially corrupts (e.g., set to zero or adding noise) the inputs. In an imputation context, a DAE would treat missing values as noise. A variational autoencoder (VAE) [KW14] is a type of autoencoder whose encoder outputs two additional vectors: one of means and one

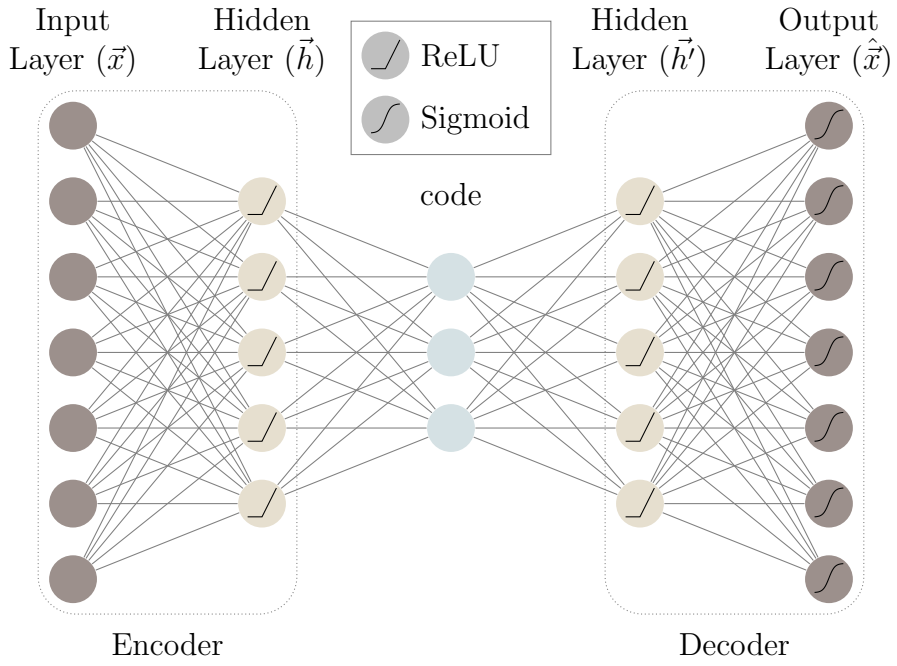

Figure 2.1: An example of an undercomplete autoencoder architecture that takes input data with D features; using rectified linear unit (ReLU) activations at each hidden layer, and a Sigmoid activation at the final layer.

of standard deviations, detailing a Normal distribution for each feature. Due to this nature, VAEs also require an additional loss term to penalize differences between the learned and true distribution, and they are considered generative models. For a deeper dive, refer to [Mic22].

Due to their nature, at their simplest, autoencoders do not require separate labels, instead, the data themselves behave as the labels. Therefore, many learning algorithms for autoencoders are unsupervised. To guide the training and update the weights, autoencoders require a *reconstruction loss* or *reconstruction error*. That is, a metric that measures how well the autoencoder recreated the original dataset. Commonly used metrics for reconstruction loss include:

- Binary Cross Entropy (BCE), also known as log-loss, measures accuracy of binary

classifiers that output a probability, or in the case of autoencoders: BCE loss, $J(\cdot)$ in Eq. 2.1, where $f(\cdot)$ is the autoencoder, $\vec{x}_i$ is one sample of D features or dimensions, $\hat{\vec{x}}$ is the reconstructed input, and N is the total number of samples, can be computed as:

$$J(\hat{\vec{x}}, \vec{x}) = -\frac{1}{N}\sum_{i=1}^{N}\left[\vec{x}_i \log(\hat{\vec{x}}_i) + (\vec{1} - \vec{x}_i)\log(\vec{1} - \hat{\vec{x}}_i)\right] \qquad \hat{\vec{x}}, \vec{x} \in X \in \mathbb{R}^{N \times D} \tag{2.1}$$

- Kullback-Leibler Divergence (KL Divergence), also known as relative entropy, measures the difference between two probability distributions [KL51]. Technically it is not a true distance metric since it is asymmetric, and instead it is referred to as a divergence, which is where it gets its name. KL Divergence is used as a component of the loss function for VAEs which computes the divergence between the multivariate normal distribution defined by the means and standard deviations learned in the model, and the multivariate standard normal distribution (all means of 0 and standard deviations 1). KL Divergence, $KL(\cdot \mid\mid \cdot)$ in Eq. 2.2, where $\mathcal{N}$ is the normal distribution, $\vec{x}_i$ is a sample from the distribution $\mathcal{N}(\vec{\mu}, \vec{\Sigma})$, $\vec{\mu}$ and $\vec{\Sigma}$ are from the means and standard deviations learned by the autoencoder, and $\vec{z}_i$ is a sample from the distribution $\mathcal{N}(\vec{0}, I)$:

$$KL(\mathcal{N}(\vec{\mu}, \vec{\Sigma}) \mid\mid \mathcal{N}(\vec{0}, I)) = \frac{1}{N}\sum_{i=1}^{N}\left[\vec{z}_i \cdot log\left(\frac{\vec{z}_i}{\vec{x}_i}\right)\right] \tag{2.2}$$

2.5.3 Regularization

Deep learning models as-is can learn very complex functions. However, too much complexity can be difficult to hone for particular tasks. Regularization techniques add constraints or operations that adjust or simplify the model in order to tailor the model's performance.

2.5.3.1 Batch Normalization

Batch normalization (Batch-norm) is a regularization technique that standardizes the output of a layer using the mean of and variance of that layer's outputs for the current batch for each feature, in order to combat the gradient explosion or vanishing problem in deep networks [IS15a]. Batch-norm can confer faster model convergence, higher accuracy, and allow the model to tolerate higher learning rates.

2.5.3.2 Dropout

Dropout temporarily deactivates or ignores neurons in input and/or hidden layers so that they do not contribute to learning model weights during training [SHK14]. By prevent complex interactions between neurons by simply disallowing some percentage of interaction, dropout may help force the network to learn better representations.

Batch-norm and Dropout There is literature that suggests that combining Batch-norm with dropout can cause problems for training deep learning models [LCH18]. Dropout changes the variance of a layer's outputs during training but not when evaluating, therefore causing a shift in neuron variance between training and evaluation. Batch-norm however uses the statistics learned during training and applies the same variance when evaluating, which has now changed due to dropout. There are some suggestions on particular order of operations that can assuage this issue.

2.5.3.3 Early Stopping

Sometimes models struggle with convergence and their performance may stray away from the local or global optima. Early stopping evaluates the model on a validation dataset after each epoch, and stops the training process if the performance of the model on the validation dataset starts to degrade.

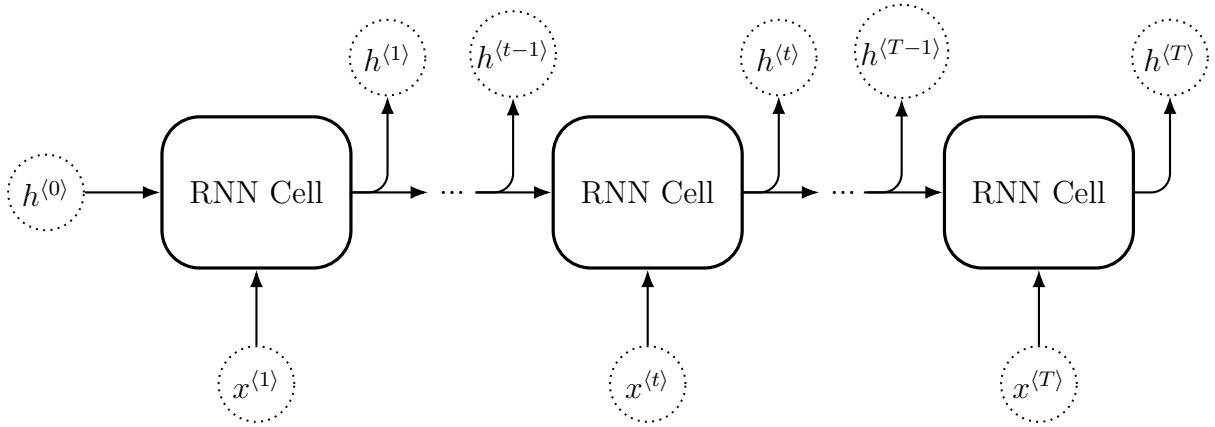

Figure 2.2: Depiction of multiple RNN cells rolled out over a time-series with T time points.

2.5.3.4 Batch Swapping

Batch swapping is a similar corruption-style regularization technique as dropout [Mue]. During training, values are swapped between rows within a batch at random for a random set of features.

2.5.4 RNNs

A recurrent neural network (RNN) is a type of neural network that can handle time series data of varying sequence lengths. Instead of a basic computational unit being a neuron, a basic computational unit of an RNN is an RNN Cell. Each cell takes two inputs at a time t: the previous hidden state $h^{\langle t-1 \rangle}$, and the input $x^{\langle t \rangle}$. What is special about the RNN cell is that it passes through some form of "memory" to each subsequent cell, therefore passing information along time points (Fig. 2.2).

There are many types of RNN cells. A shortcoming with vanilla RNN cells is that they tend to "forget" old information as new measurements for each time point come in, and can suffer from the vanishing or exploding gradient.

2.5.5 Long-Short Term Memory

Long short-term memory (LSTM) models [HS97] are a type of RNN that are designed to deal with the vanishing/exploding gradient problem. Each cell takes three inputs at a time t: the previous cell state $c^{\langle t-1 \rangle}$, the previous hidden state $h^{\langle t-1 \rangle}$, and the input $x^{\langle t \rangle}$. The cell depicted in Fig. 2.3 contains four gates that provide specific functionality to the cell, where $\frown$ represents concatenation: the forget gate $f^{\langle t \rangle}$ (Eq. 2.3), the input gate $i^{\langle t \rangle}$ (Eq. 2.4), the update cell state gate $\tilde{c}^{\langle t \rangle}$ (Eq. 2.5), and the output gate $o^{\langle t \rangle}$ (Eq. 2.6). Ultimately, the LSTM computes the new cell state $c^{\langle t \rangle}$ (Eq. 2.7) and hidden state $h^{\langle t \rangle}$ (2.8) for the current step, potentially passing them both to the next cell, and outputting $h^{\langle t \rangle}$. The cell state behaves like an information conveyor belt, with the gates interacting along the way to regulate (by removing, modifying, and adding) information. In this setup, the cell state behaves like long-term memory and the hidden state behaves like short-term memory, which is where the model gets its name.

$$f^{\langle t \rangle} = \sigma(W_f \cdot [h^{\langle t-1 \rangle} \frown x^{\langle t \rangle}] + b_f) \tag{2.3}$$

$$i^{\langle t \rangle} = \sigma(W_i \cdot [h^{\langle t-1 \rangle} \frown x^{\langle t \rangle}]) + b_i) \tag{2.4}$$

$$\tilde{c}^{\langle t \rangle} = \tanh(W_c \cdot [h^{\langle t-1 \rangle} \frown x^{\langle t \rangle}] + b_c) \tag{2.5}$$

$$o^{\langle t \rangle} = \sigma(W_o \cdot [h^{\langle t-1 \rangle} \frown x^{\langle t \rangle}] + b_o) \tag{2.6}$$

$$c^{\langle t \rangle} = f^{\langle t \rangle} \otimes c^{\langle t-1 \rangle} + i^{\langle t \rangle} \otimes \tilde{c}^{\langle t \rangle} \tag{2.7}$$

$$h^{\langle t \rangle} = o^{\langle t \rangle} \otimes \tanh(c^{\langle t \rangle}) \tag{2.8}$$

2.6 Related Work

We apply deep learning architectures, in particular, autoencoders, for imputation on EHR datasets in a static setting so that we may then carry out a downstream clinical decision-making task. In general, deep learning autoencoders were popularized for computer vision

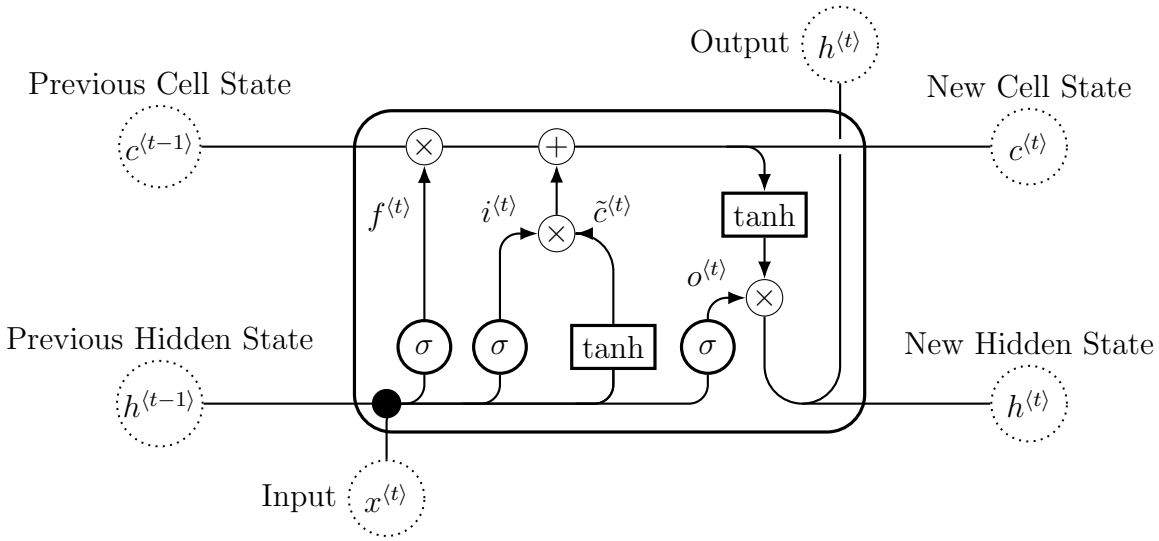

Figure 2.3: Depiction of a single LSTM cell, where $\times$ is element-wise product, $+$ is element-wise addition, σ is the sigmoid function, and $\bullet$ or $\frown$ is concatenation.

tasks on image data. However, these types of models have found success in other applications on tabular data. We explore existing work on deep learning, focusing on autoencoder methods, for imputation on static data. To enable other researchers to explore the longitudinal setting in future work, we include existing works that employ longitudinal imputation.

2.6.1 Static Imputation with Deep learning

Particular configurations of VAEs have seen success in imputation on data MNAR and MCAR in myriad domains including traffic forecasting [BVM19], synthetic and simulated milling circuit data MCAR [MKA18], and facial image data MAR [WNN19]. Unlike biomedical and EHR data, however, these datasets involve automated data collections systems and can be modeled with clear Gaussian distributions. Camino et al. [CHS19] explored the effectiveness of variational autoencoders across tabular data in different domains such as breast cancer data, credit card data, and optical character recognition data. For high dimensional data, Chen et al. [CS19] proposed sparse convolutional denoising autoencoders to impute yeast and human genotypic data. They leveraged the added complexity of con-

volutional layers and a sparse weight matrix to make imputation of high dimensional data tractable. Gondara et al. proposed MIDA, an approach that used an overcomplete denoising autoencoder with the assistance of simple imputation methods such as mean or mode imputation [GW18]. Beaulieu-Jones et al. proposed denoising autoencoders for the imputation of EHR data on patients diagnosed with Lou Gehrig's disease [BM16].

While the use of autoencoders for imputation has been previously explored, *the context in which any of these techniques may be optimal is unclear.* This observation prompted our development of an open framework that would enable the rapid implementation of different autoencoder imputation methods on a dataset, comparing imputation performance and the sensitivity of a given predictive task relative to inferred missing values [ZPD21].

Generative adversarial network (GAN) Aside from autoencoders, another popular deep learning model architecture for imputation is the generative adversarial network (GAN) [GPM14]. Generative adversarial networks are a type of neural network made of two parts: a generator and discriminator. The generator attempts to generate believable samples (that could be confused to be from the original dataset), and the discriminator attempts to distinguish between generated and real/original samples. The intuition behind the use of GANs for imputation is like that of autoencoders; while autoencoders attempt to learn to recreate the original dataset using relevant information from other samples and features, GANs attempt to *generate* samples that could believably be from the input domain. While autoencoders follow a simple reconstruction loss, GANs train with an additional game-theory-driven loss that comes from the discriminator. Also, while autoencoders produce a latent representation, not all of them are generative models. GANs are generative models and can produce new samples given a random vector of inputs. Yoon et al. proposed generative adversarial imputation network (GAIN), an imputation procedure using a modified GAN, where the discriminator aimed to distinguish which data was originally missing [YJS18].

2.6.1.1 Mixed Feature Imputation

Some researchers have begun to develop methodologies for particular autoencoder versions to address imputation on mixed feature datasets. Simple techniques added an extra layer to act as an embedding layer to the categorical feature space [Mue]. Valera et al. developed Bayesian non-parametric latent feature models with Gibbs sampling for imputation [VPL17]. They assigned a different likelihood and metric for the following feature types: real, positive real, count, categorical, and ordinal. Nazábal et al. expanded on Valera's work in applying a similar concept to VAEs such that each feature was represented by a different likelihood depending on the feature type when learning the parameterization [NOG20]. Nazábal's work inspired a slew of other work such as Gootjes-Dreesbach et al. who combined VAEs with modular Bayesian networks in order to generate simulated data and model missing data, though they did test on a real world dataset with missing values they did not control the missingness scenarios [GSS20]. Akrami et al. used β-divergence instead of KL Divergence for computing the reconstruction term for continuous and categorical features in a VAE [AAL20]. Ma et al. trained a VAE per feature and combined the latent representation from all of them into another VAE, though they only tested on MCAR [MTT20]. While VAEs are popular for this task as they can be modified to model different distributions depending on the feature type, others such as Li et al. passed the data through stacked denoising autoencoders (DAEs), and Hou et al. did not address the different feature spaces in their work [LMX18, Hou20].

2.6.1.2 Collection Frameworks

Understanding imputation methods as a whole and making techniques more easily available is a growing need. Projects like `jenga` provide collections of data corruption methods and a workflow for understanding how those corruptions, such as missing data, and various solutions to those corruptions affect downstream prediction [JAB21]. Projects like `autoimpute`

provide collections of statistical imputation methods [KBB19]. Some existing works compared different statistical and machine-learning-based imputation methods in a wide variety of domains, though they did not provide a framework for others to do so [JPR19, PC23, SS23]. There are other works that made a usable framework available, serving as a collection of deep learning imputation methods. Tilman et al. similarly implemented a framework to compare a wide variety of autoencoders toward applications in computer vision [Kro21]. Jarrett et al. developed `clairvoyance` [JYB21], a framework for end-to-end time series tasks such as imputation and classification, among others. `clairvoyance` implemented a series of deep learning techniques as well as some baseline methods such as interpolation or mean imputation. `PyPOTS` is a toolbox containing the implementation of specific deep learning techniques published in the literature that handle time series imputation and classification tasks [Du23]. Our work enabled the implementation and comparison of several existing methods in published work, was more generally about robustly comparing autoencoders for imputation rather than a collection of specific implementations, and only focused on imputation on static datasets.

2.6.2 Longitudinal Imputation with Deep Learning

Aside from basic extensions of commonly known imputation methods such as simple imputation to the longitudinal settings, most remaining techniques for multivariate longitudinal imputation involve deep learning. Che et al. proposed gated recurrent unit with decay (GRU-D) [CPC18], a longitudinal imputation technique based on an altered GRU cell for clinical data. GRU-D imputed values to the last observation, but decayed towards the mean over time by incorporating a temporal decay mechanism in the recurrent cell. GRU-D made assumptions specific to clinical data, e.g., the human body tends to want to stay in homeostasis. Cao et al. proposed bidirectional recurrent imputation for time series (BRITS) [CWL18], which expanded upon the temporal decay mechanism in the recurrent cells from GRU-D. Compared the GRU-D, BRITS instead used LSTM cells, trained the same model

on the imputation and prediction task jointly, was bidirectional, and imputed the value of a variable at time t using a weighted sum of a variable's "history" (the previous time step's hidden values h_{t-1}) versus the values of other features at time t. Du et al. proposed self-attention-based imputation for time series (SAITS) [DCL22]. Like BRITS, SAITS imputed missing values using temporal (or historical) data and feature information. However, SAITS achieved this by using two diagonally-masked self-attention blocks in a transformer model, foregoing the use of a recurrent network for imputation. SAITS separated the tasks of imputation and original data reconstruction that we considered the same task in this work, and jointly trained the model on these separate tasks.

2.6.3 Summary

Researchers are increasingly employing Autoencoders to handle imputation tasks. There are both particular configurations of autoencoders and collections of particular configurations in the literature. VAEs tend to work best for automatically collected data that tend to follow a Gaussian distribution (e.g., sensor data). While other specific implementations exist in the static setting and for mixed feature data, we are not aware of any collections of them, and many do not consider the consequences of imputation on a downstream task or across a wide range of missingness mechanisms. In contrast, our work served as a flexible autoencoder that could take on many forms rather than a collection of specialized methods, and evaluated all of them in a unified way on both the imputation task and downstream predictive task. The other popular deep learning architecture used for imputation was GANs. Similar to VAE, GANs are generative models, however, our work focused on understanding autoencoders as a whole for imputation rather than searching for the best class of deep learning architectures. Additionally, autoencoders did not involve creating an extra model that would later be discarded (i.e., the discriminator in the GAN). In addition to static data imputation, there are a growing number of deep learning methods to impute longitudinal data. However, most of these methods did not look at missingness mechanisms, since there is no established

paradigm for reasoning with missingness in the time dimension. We did not address this gap in the literature directly, but rather opened it up for discussion and direction for the future.

CHAPTER 3

Creating a Framework for Autoencoder-led Imputation

Recognizing that the adoption of EHRs had made patient data increasingly accessible, we were tasked with the development of various clinical decision support systems and data-driven models to help physicians. However, we would soon learn that missing data were pervasive in EHR-derived datasets, which introduced significant uncertainty, if not invalidating the development of a predictive model for these clinical decision-making tasks. We wanted to salvage the data we did have under information poor conditions to perform well on our predictive tasks, *without* compromising the integrity and reliability of analyses. We approached the topic of imputation with deep learning, and in particular, autoencoder models, based on the growing body of work demonstrating the power and promise of ML-based imputation methods. We believed autoencoders were fit for imputation over other methods due to their ability to rapidly impute on large datasets, such as EHR datasets, and quickly learn nonlinear relationships. However, despite the appeal of autoencoders for imputation, there were no theoretical proofs on their behavior.

We wanted to characterize how autoencoders would behave as imputation models, and to compare and choose between which model may be best for a particular dataset and task. To achieve this, we needed a framework that could train and evaluate a wide array of autoencoder configurations and could compare to other commonly used models for imputation. Additionally, we needed a framework that could control different missingness scenarios so that we could begin to define the characteristics of autoencoders under each of the missingness mechanisms and at what percent of missingness. We expanded upon prior literature

by taking a closer look at not only how accurately autoencoders could learn to impute an EHR dataset with missing values, but also how that imputation would affect downstream predictive performance.

As such, we developed an initial version of *Autopopulus* to empirically study the potential of autoencoders for imputation. To demonstrate Autopopulus, we used the Center for Kidney Disease Research, Education and Hope (CURE-CKD) dataset [NDA19, TAD19] to identify individuals at-risk for and with chronic kidney disease (CKD). We implemented several existing autoencoders for imputation, as well as our own approach for imputation that utilizes a completely discretized formulation of the data.

3.1 Building the Pipeline

With Autopopulus, we aimed to provide an extensible framework for developing, testing, and ultimately comparing different autoencoder imputation methods. Our first foray into building the pipeline was influenced by existing works that used autoencoders for imputation. We drew on the work of McCoy et al. [MKA18], Gondara et al. [GW18], and Beaulieu-Jones et al. [BM16] in addition to implementing a novel autoencoder imputation technique. We designed Autopopulus so it could be used in the same fashion as a Scikit-learn [PVG11] imputation model. This lent itself to easy and intuitive use for future datasets, as it interfaced well with widely used tools with minimal overhead. The initial form of the imputer training pipeline is shown in Fig. 3.1, and we describe key elements below.

Simple Amputation At the time of building the initial version of Autopopulus, `pyampute` (Section 4.2) had not been implemented yet. Instead, we implemented a simple amputation procedure. For data MCAR, we replaced a value with NaN uniformly at random for selected variables. For data MAR we created a cutoff, determined by the percentage of missingness specified, on an observed variable. If a given data entry fell above the cutoff, the value for the

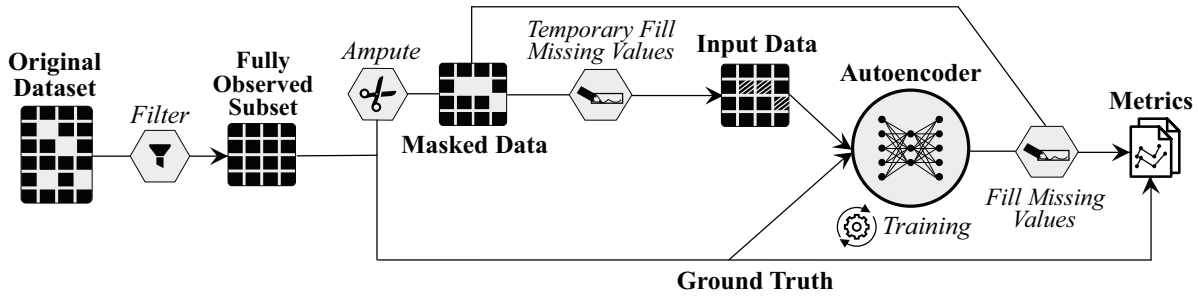

Figure 3.1: The initial pipeline for training the autoencoder. After training the autoencoder, its output was only used to fill originally missing values.

chosen missing variable would be NaN, simulating the scenario where the value was missing due to another observed variable. For data MNAR we created lower and upper cutoffs based on the variables that would be missing themselves (e.g., a historically normotensive patient opted to not have their blood pressure measured). If a patient fell inside the cutoff range, the value would be missing, simulating the scenario where the value was missing due to the value itself. Under MAR we could choose which observed features we wanted to control the missingness of each of the missing features.

Data processing The input to our model would be a dataset $\hat{X}$ and label X, which was the unaltered or true version. We standardized all continuous variables using a min-max scaler, and one-hot encoded all categorical variables. Autoencoders are not designed to handle missing values by default, so we first performed a "warm start" imputation: initializing the missing values in $\hat{X}$, and also X if the true dataset also had missing values, with some initial estimate. We could choose to fill in missing values with 0 or with per-batch simple imputation (mean imputation for continuous variables and mode for categorical variables), or data (only originally continuous variables) might be discretized with a uniform distribution imposed over missing values. Note if the latter was true, the ground truth was also discretized before being fed to the autoencoder, and the autoencoder output was undiscretized.

Autoencoder architecture As mentioned prior, autoencoders come in many flavors. Depending on the nature of the dataset or problem, we needed to specify a type of autoencoder. This initial form of Autopopulus supported common architectures such as DAE and VAE. Additionally, it supported undercomplete and overcomplete networks by allowing users to specify layer dimensions explicitly or by implicitly by the proportion of the original number of features.

Loss The reconstruction losses supported by Autopopulus were BCE, MSE, and a combination that applied BCE only to categorical variables and MSE to continuous variables. When the autoencoder was not a VAE, loss was purely reconstruction-based. When the autoencoder was a VAE, we added an extra KL Divergence error term in addition to the reconstruction loss. We could also choose the option to only evaluate the loss on originally non-missing data instead of all the data.

Training Unless specified, we tuned the number of layers and nodes per layer, learning rate, L2 penalty, maximum epochs, and early stopping patience on the validation set using an automatic sweep of hyperparameters with asynchronous hyberband scheduling (ASHA) [LJR18].

Output Once the loss had been calculated, the model output was further processed before computing imputation performance metrics. The same steps were employed when using the model for imputation after training. If the data had been discretized, we undiscretized the data, which only affected the originally continuous variables that had been discretized. We applied the sigmoid function only to the categorical variables. Lastly, all methods were only used to fill in missing values, so the original values were otherwise kept.

3.1.1 Adding New Imputation Methods

Alongside existing autoencoders used for imputation, we used Autopopulus to design a new technique for comparison, showing how the framework could be extended. In this method, the data were discretized (into one-hot features) and a uniform distribution was imposed across the discretized variables wherever a value was missing as a data preprocessing step. The goal of the autoencoder was to take the uncertainty, modeled by a uniform probability, for any given missing variable and learn to shift weight onto the discrete bin that most likely represented the true value. For example, if age was discretized into 10 bins and its value was missing, then each bin had a 10% probability of being the "true" bin. We replaced the missing value with this uniform probability. The data $\hat{X}$ were passed through the autoencoder model and the result was then compared to the label X according to a given loss function to train the model. We employed minimum description length (MDL) discretization via the Orange package to automatically create bins [FI93, DCE13]. MDL discretization is a supervised algorithm that recursively decides the split that minimizes entropy and minimum description length principles, as well as the labels assigned to each sample. We chose this discretization method to be able to employ Autopopulus on a wide array of datasets quickly and because it had been used successfully on clinical data in the literature [MPL13].

When discretizing, we chose a vanilla autoencoder with BCE loss for the reconstruction error and trained using the Adam optimizer [KB14]. We used BCE instead of mean-squared error because our inputs were either binary due to discretization, or continuous between 0 and 1 after imposing a uniform distribution across bins for a missing variable. Imposing a uniform probability across missing values maximized entropy for the missing value, which was then penalized by the BCE loss, forcing the autoencoder to focus on correcting for missing values.

To allow a more direct comparison to other imputation methods, we first "undiscretized"

the output. To undiscretize the data, we mapped the originally-continuous variables down to the mean of their most likely bin (the one with the highest score). For example, if age was discretized into 10 bins, and the autoencoder yielded the highest score for the range (50,60], then the patient's age would be estimated to be 55 years.

3.2 Enabling Experiments to Compare Imputation Methods

To create a profile for autoencoders as imputers, we needed to comprehensively evaluate them in a way that enabled fair comparison and explored relevant and useful aspects of what we expected to gain from incorporating an imputation step. An important aspect of imputer behavior was its estimation accuracy. However, imputation does not occur in a vacuum; most of the time imputation is not the end goal, but rather the end goal is some downstream task such as regression or classification. Aside from evaluating the imputation models themselves, we wanted to characterize and analyze any effects of imputation on downstream tasks. We developed an experimental pipeline with a consistent interface and unified training pipeline both on the immediate imputation task, and downstream predictive task.

We conducted two sets of experiments to answer two investigative questions: the ability for various imputation methods to create an accurate representation of the dataset (Fig. 3.1); and the impact of missingness and imputation on downstream prediction (Fig. 3.2).

Imputation Accuracy In our first set of experiments, we explored an imputation method's ability to create an accurate representation of the data and can control for performance given different levels of missingness and mechanisms. We filtered the dataset down to the fully observed subset of data and amputed according to different mechanisms and at different missingness rates. In this set of experiments, we tested the MAAPE and RMSE by amputing at both a low and high percentage of missingness (33% and 66%) across all three missingness mechanisms (MCAR, MAR, and MNAR).

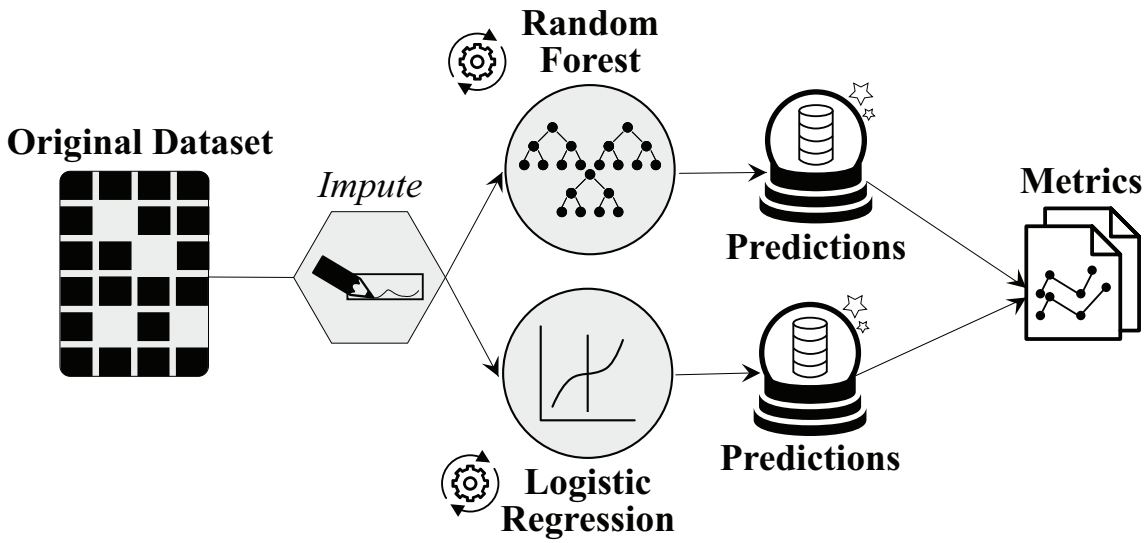

Figure 3.2: The data flow for the predictive task. Imputation was done with either baseline methods such as k-nearest neighbors (KNN), or an autoencoder-led method. All imputers were trained (or fitted) first and then used to impute the dataset.

We compared various types of autoencoders: DAE, VAE, and our new method based on a vanilla autoencoder which we labelled (*APnew*) (Section 3.1.1). We followed the configurations of the following works for their respective autoencoder types to tie our work to the literature. For the DAE we followed Beaulieu-Jones et al. [BM16] and MIDA [GW18], and for the VAE we followed McCoy et al. [MKA18]. Baseline methods for static imputation included simple, k-nearest neighbors (KNN), and MICE imputation. We refer to simple imputation as mean imputation for continuous variables and mode imputation for categorical variables.

Downstream Predictive Performance In the second set of experiments, we used the entire dataset "as is" to explore the effect of imputation on predictive performance via the Brier score (calibration), PR-AUC, and ROC-AUC for classifying rapid decline. We used a train, validation, test split of 60%, 20%, and 20% respectively both for training the autoencoder and for training the predictive models.

To explore whether linear or non-linear predictive models were better suited for the static predictive task, we trained both a logistic regression model and a random forest model on top of the autoencoder outputs. We trained the models to be sensitive to the positive class due to the dataset being highly imbalanced by taking advantage of class weights. Hyperparameters for the logistic regression and random forest models were tuned during validation automatically via Scikit-learn. Each predictive model was trained on 100 stratified bootstrapped samples. The logistic regression and random forest models were implemented via Scikit-learn [PVG11]. The 95% confidence intervals and means for each classification performance metric, such as PR-AUC, were produced from this bootstrapping process.

3.3 Use Case: CURE-CKD

We evaluated our framework using a clinical dataset, as motivated by a real-world clinical decision-making problem we had first set out to tackle. In particular, we first used Autopopulus on the CURE-CKD dataset, detailed below.

3.3.1 The CURE-CKD Dataset

Amongst its many goals, the Center for Kidney Disease Research, Education and Hope (CURE-CKD) project (and its subsequent Registry [NDA19, TAD19]) aims to use machine learning to assist with preventative care for patients with chronic kidney disease (CKD). CKD is marked by a gradual loss of kidney function. However, there are a subset of patients that experience a sudden drop in kidney function, which is called rapid kidney function decline (RKFD) or *rapid decline*. The prevailing method for testing for CKD and tracking disease progression is measuring the estimated glomerular filtration rate (eGFR) [Cla12]. RKFD is normally defined as an average of greater than or equal to 5 ml/min/1.73 m^2/year decline in eGFR, compared to an average of 1 ml/min/1.73 m^2/year for those diagnosed with CKD [Cha13]. For the purposes of our work, we defined RKFD as a 40% decline in eGFR

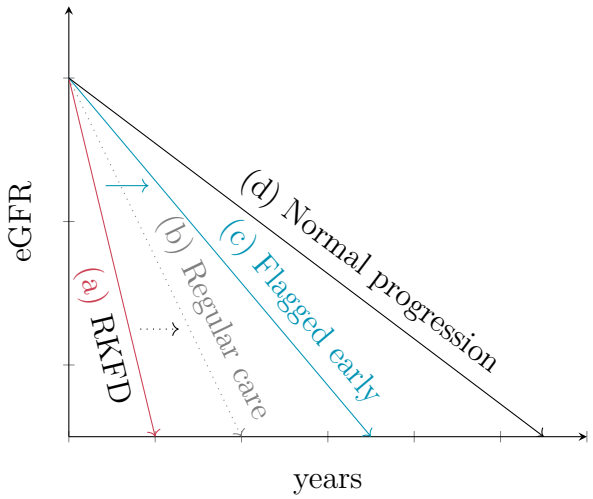

Figure 3.3: This graph visualizes simplified eGFR trajectories for a patient (a) with RKFD, (b) who might have experienced RKFD but received treatment without any early flagging, (c) who might have experienced RKFD but flagged and treated early, and (d) with normal CKD progression. With prediction and intervention we hope to push CKD patients towards a more normal rate of decline (solid arrow) compared to not predicting for RKFD and reacting with intervention at a later time (dotted arrow). Note, this is the goal, rather than a current observation. For more details refer to [KSR17].

over two years [LTS14].

We aimed to elucidate if it was possible to improve CKD patient outcomes, who might have RKFD to motivate future trials for intervention. Towards this aim, we built a model to predict which patients would experience RKFD two years after the entry period. Figure 3.3 depicts a simplified representation of how we hope or expect that future intervention could positively impact patients. In a future study, we would flag these patients and do preventative work to improve their eGFR trajectories and observe if their outcomes could be improved with intervention.

To train a model and make predictions, we used the CURE-CKD Registry. The CURE-CKD Registry comprised a comprehensive, real-world, longitudinal dataset of EHR-derived

data from two large healthcare systems (Providence St. Joseph Health (PSJH) and University of California, Los Angeles (UCLA) Health) over a period of 12 years (2006-2017). Patients entered and exited the registry that generated the dataset at different points over the 12-year period, resulting in different history lengths for each patient. The CURE-CKD dataset consisted of two cohorts: patients diagnosed with CKD or flagged as "at-risk" for CKD (diagnosed with diabetes (DM), hypertension (HTN), or pre-DM), based on diagnostic administrative codes, laboratory data, vital signs, and medications. We included patients over 18 diagnosed with CKD, and excluded patients that were "at-risk" in order to focus on rapid decline prevention in the CKD population. We defined study-entry as the date of the first serum creatinine measurement during the observation period for a patient, and the entry period as the 90 days following study-entry.

We used vitals and labs (eGFR, A1c, systolic blood pressure, and number of ambulatory and inpatient visits) in addition to demographic information (age, sex, ethnicity, and rurality status) and risk factor information (diagnosis of HTN, DM, or pre-DM, and use of ACEIs and ARBs) to predict RKFD. The registry reported these features as per-year aggregates. For example, systolic blood pressure was recorded as an average over the entry period and each following year, but as a sum for ambulatory visit count. All multicategorical features, such as patient race, were one-hot encoded. We dropped features missing more than 89% data: the entry period urine albumin-creatinine ratio (uACR) and urine protein-creatinine ratio (uPCR) means.

There were $4,067$ patients in the positive class (experiencing rapid decline), which made up 4.59% of the $88,560$ patients diagnosed with CKD. There were $9,062$ (10.23%) patients that were not missing any data. Notably, 74.5% of A1c entries were missing at study-entry and 71.9% at entry-period. 60% of systolic blood pressure entries are missing at study-entry and 60.5% at entry-period.

3.3.2 Imputation Performance

We reported metrics computed only where the data were originally missing for imputation performance. Values that were not originally missing were kept at the end of the pipeline, so we were largely interested in the performance on originally missing values only. Fig. 3.4 shows that different imputation methods created more accurate representations of the missing data under different missingness mechanisms, and those patterns remained similar across the amount of data missing. All models achieved lower error under MNAR compared to MAR and MCAR. Under MAR, the RMSE under 33% missing was minutely larger than 66% missing. Under MCAR and MAR, MICE achieved the lowest MAAPE. However, under data MNAR, the VAE achieved the lowest error. In general, under MNAR the autoencoder imputation models tended to achieve lower error than the baseline models, aside from APnew. APnew produced a large error across all missingness scenarios.

APnew uniquely discretized and then undiscretized the data. Therefore, only for this approach were we interested in the amount of times the autoencoder correctly guessed a bin for an originally continuous feature before undiscretizing. We evaluated this accuracy over the bins for missing values only in Table 3.1. We observed that the autoencoder was fairly accurate under MCAR and MNAR for both percentages of missingness, but struggled more with data MAR.

3.3.3 Predictive Performance

Fig. 3.5 shows that the logistic regression model was in general less calibrated than the random forest model, regardless of which imputation method was used. Though the difference was small, the autoencoder-based imputation methods tended to be slightly less calibrated than the baseline methods; however, APnew was the best calibrated autoencoder imputation method.

The imputation method used did not have a large impact on the PR-AUC and ROC-AUC

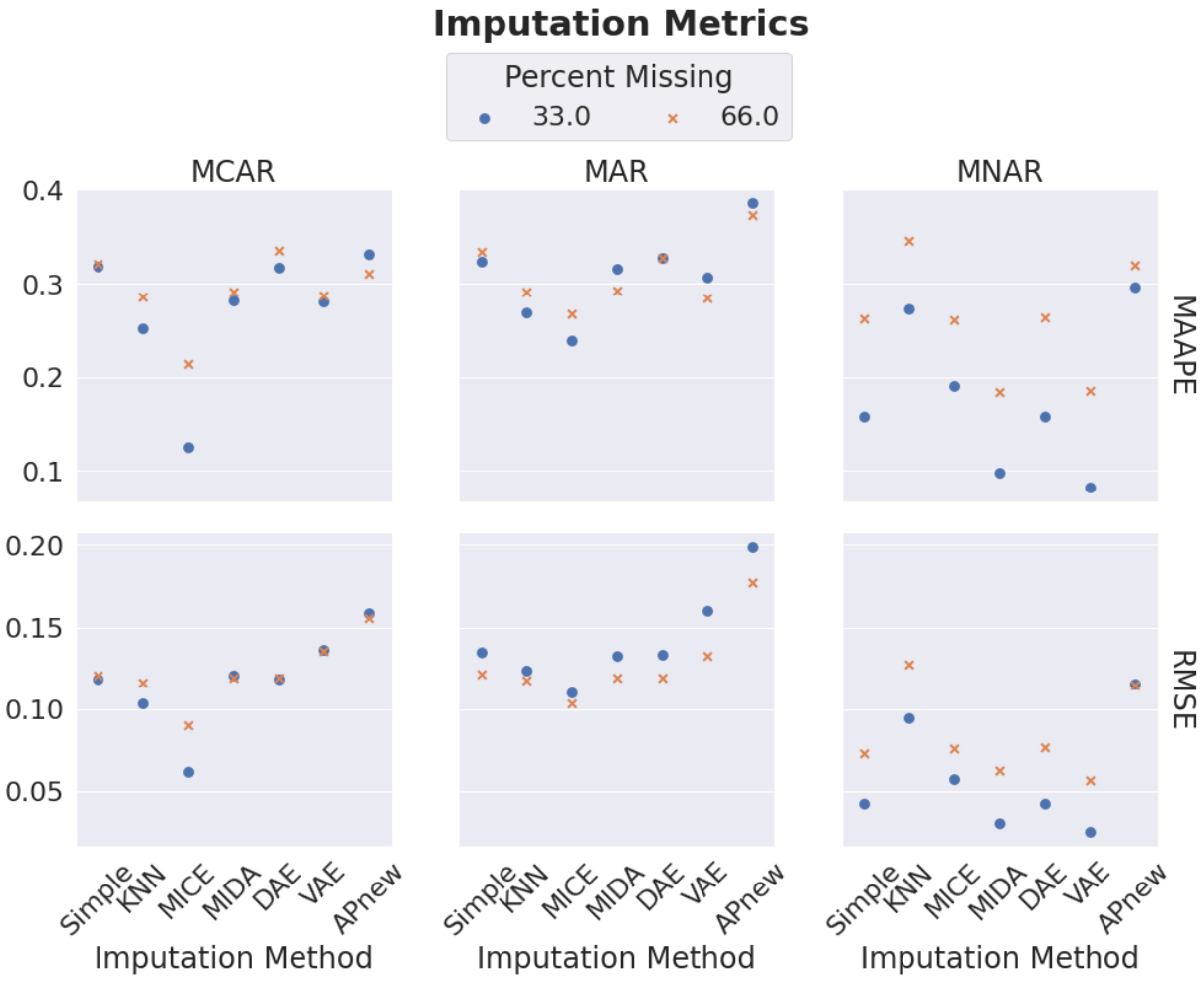

Figure 3.4: Imputation performance captured via MAAPE and RMSE for each imputation method across missingness scenarios, computed only on the missing values. Every column is a different missingness mechanism, and each row is a single imputation metric. The y-axis range differs between MAAPE and RMSE for a closer inspection of the performance under each metric. Note that MAAPE is a percentage reported as a decimal value, while RMSE is a non-negative decimal score.

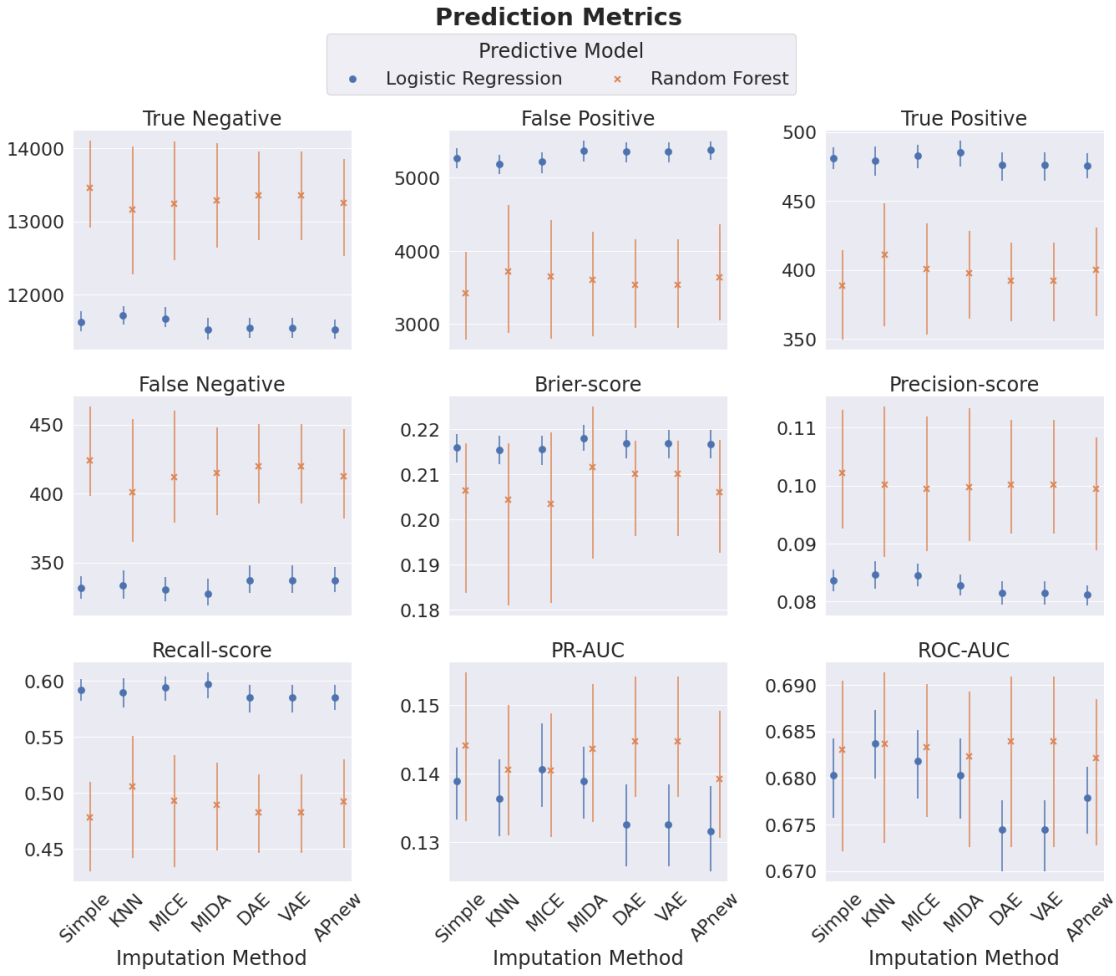

Figure 3.5: Average predictive performance for each imputation method on the entire dataset as-is for the predictive models across bootstraps. The true negatives, false positives, true positives, and false negatives are all reported as counts. The remaining metrics are percentages reported as decimal values. Note that the Brier score is in fact a loss, where smaller is better.

Table 3.1: Accuracy Over Bins (only `APnew`)

Mechanism	Percent	Accuracy Over Bins
MAR	33.0	0.626
	66.0	0.699
MCAR	33.0	0.804
	66.0	0.842
MNAR	33.0	1.000
	66.0	0.864

when using a random forest model. But in general, the baseline predictive methods produced better results when using a logistic regression. Markedly, the PR-AUC and ROC-AUC for MIDA almost exactly mirrored simple imputation no matter which predictive model was used, which called into question whether MIDA offered much advantage beyond simple imputation for the downstream task on this dataset. The remaining three autoencoder-based imputation methods performed similarly across both predictive models. The most calibrated predictive models were trained on data imputed with either MICE or APnew. Across all imputation methods, the logistic regression model offered tighter confidence intervals and better Recall, but overall poorer performance regarding calibration (Brier score), Precision, PR-AUC, and ROC-AUC. Do note that the Brier score acts as a loss, where a larger value means poorer calibration. MICE imputation produced a very similar PR-AUC, whether using a logistic regression or a random forest model for prediction. KNN imputation produced a very similar ROC-AUC, whether using a logistic regression or a random forest model for prediction. Another point of interest was the relatively poor PR-AUC across all imputation methods and predictive models. Upon closer inspection, we noticed that the precision was extremely poor while recall was more acceptable. Both the logistic regression and random forest struggled with false positives (i.e., Type I errors) on this dataset.

3.3.4 Takeaways

Autopopulus enabled us not only to easily implement multiple autoencoders with a consistent interface and unified training pipeline, but also allowed us to explore their performance on the same dataset and under different missingness scenarios, including MAR and MNAR. We could identify patterns across missingness mechanisms, on the hypothesis that each mechanism behaves differently. For example, when comparing imputation metrics across methods we saw better imputation performance on data MNAR over data MAR across imputation methods, even though MNAR is famously difficult to handle. In this case, this observation might have been due to the steps Autopopulus took to simulate MNAR, which could have been too simplistic compared to real-world series of events. Autopopulus also allowed us to easily test which imputation methods would perform best in different missingness scenarios. Per prior findings in the literature, under MCAR and MAR, MICE consistently performed best, though sometimes only marginally. However, under MNAR the VAE performed best. This finding may have been due to how we simulated MNAR by specifying lower- and upper-bound cutoff thresholds on the missing variables themselves. The encoder portion of a variational autoencoder produced means and standard deviations, defining a normal distribution for each input. This likely made it "easier" for the VAE to guess that the missing values were in the tails of the distributions, if its inferred means and standard deviations were accurate enough.

Through Autopopulus, we were also able to investigate how the different imputation methods might affect a predictive model's ability to handle class imbalance. Though we attempted to deal with class imbalance in the predictors by weighting samples, we did see the imbalance severely affecting performance. It is likely the ROC-AUC was significantly larger than the PR-AUC due to the large number of samples in the negative class (not experiencing rapid kidney decline). Overall, we saw that no one imputation method significantly helped remedy the missing data problem. Markedly, by comparing methods we identified the imputation method with the consistently largest errors, which turned out to be APnew. This

result was to be expected, as the bins produced by automated MDL discretization on this dataset were wide. We do note that the predictive performance was for the unknown missingness scenario of the original dataset, as we did not evaluate the predictive performance for each missingness scenario.

CHAPTER 4

Autopopulus

We previously built an initial form of *Autopopulus*, a novel open-source framework that enabled the design and evaluation of various autoencoder architectures for efficient imputation on large datasets. Autopopulus implemented a wide variety of types of autoencoder and demonstrated a workflow that helps users make an informed decision on an appropriate imputation method. We could use Autopopulus to identify not only which imputation methods could most accurately impute on a large clinical dataset, but to also identify the imputation methods that enabled downstream predictive models to achieve the best performance for a given task. However, we explored autoencoder-led imputation under three different missingness mechanisms, missing completely at random (MCAR), missing at random (MAR), and missing not at random (MNAR), with little regard for mixed feature data.

After reading the work by Mohan and applying Autopopulus to other datasets, we realized there were two key components missing from our framework and analysis that required a deeper dive: the taxonomy of missingness scenarios, and handling mixed feature data. We had previously implemented a very simple amputation scheme. However, this time we used `pyampute` in order to impute under more granular missingness scenarios. Additionally, our previous pipeline computed the loss separately based on feature type, but did not evaluate the metrics based on feature type. We realized to properly evaluate on mixed feature data, we needed to use different metric functions for the different feature types. We also previously considered discretization a "novel" technique for imputation that we called APnew in Chapter 3. With the new context of feature-type handling, discretization was not a new addition

to an otherwise undeviating pipeline, but rather a specific realization of a feature mapping component. Under that view, we understood that the mixed data could be transformed to have all continuous or all categorical features.

4.1 Updated Pipeline

In its mature form, Autopopulus consisted of a similar three-step pipeline: 1) the data pipeline, 2) the imputation pipeline, and 3) the prediction pipeline. The data processing portion of the Autopopulus workflow when investigating missing scenarios is illustrated in the top-most bolded square, Fig. 4.1, labeled *Data Pipeline*. The imputation portion of the Autopopulus workflow is illustrated in the bottom-most bolded square, Fig. 4.1, labeled *Impute Pipeline*. Diamond blocks represent operations or models. The unfilled diamond blocks with dotted lines represent operations that are applications of the operation block they are connected to, trained with incoming data to the sister block. The dark squares on all datasets represent missing values, and unfilled squares with solid lines represent imputed values for originally missing ones. The prediction portion of the Autopopulus workflow is illustrated in Fig. 4.2.

We reframe the pipeline in full while highlighting the major changes from its earlier iteration.

4.1.1 Data Processing

First, we ingested a one-hot encoded dataset and the corresponding labels for the downstream predictive task. For datasets with a fully-observed subset where we could ampute the data, we separated the samples between the fully-observed subset and the remaining semi-observed subset. The fully observed subset encapsulated all the samples in the original dataset that had no missing values, while the remaining data were semi-observed: all the samples had at least one value missing or unobserved.

When investigating different missingness scenarios, we used only the fully-observed subset, where one copy of the data was saved as the ground truth, and the other copy was amputed according to the given missingness scenario to produce input data with missing values. In a special case of evaluation, we tested the trained imputer model on the remaining semi-observed subset of data. In this case, we split the fully observed subset into train and validation, and we designated the remaining semi-observed subset as test data. Otherwise, we split the ground truth and input data into train, validation, and test splits. When we were not investigating missingness scenarios, we used the entire original dataset as-is as both the ground truth and the input data, and there were no subsets.

4.1.1.1 Ampute

We amputed the fully-observed subset of data via the `pyampute` package (refer to Sec. 4.2) [SZS22a, SLV18]. Amputing masked or injected `NaN` (not a number) into the dataset according to a specified pattern or missingness scenario.

Missingness Scenarios A missingness pattern or scenario was defined by a missingness mechanism, the percentage of missing data, a score-to-probability function, and the set of features that should be masked. If we chose the data to be MAR, we also included the set of observed features to influence the missingness of the target to-be-masked features. The missingness mechanisms were any of MCAR, MAR, or different versions of MNAR. Here, MNAR alone referred to the scenario where the data were missing because of the values themselves, making it not-recoverable. For the other two MNAR scenarios, we generated a latent feature not based on any other features that explained the missingness of the to-be-missing target features, making them recoverable. MNAR(Y) generated a latent feature using a sample from a Yule-Simon distribution with $\alpha = 1.5$. In clinical data, there tend to be highly skewed categorical data (e.g., some procedures are more common than others), and Yule-Simon allowed us to create a skewed discrete distribution. MNAR(G) generated a

latent feature using a sample from a standard normal Gaussian distribution $\mathcal{N}(0, 1)$. The Gaussian distribution allowed us to create a continuous latent feature to ampute, as opposed to the discrete latent feature from Yule-Simon. The percent missing requested resulted in $x\%$ of the samples missing values.

Amputation Procedure `pyampute` would assign a missingness score to each sample by computing a dot product between the sample and a set of weights for each feature, where the missingness mechanism and features involved defined the weights. The score-to-probability function then took the missingness score and converted the score into a probability for that sample to have missing values. The framework would draw from a binomial distribution $B(n = 1, p)$ to convert the probability p of a sample to be missing values for the target to-be-missing features into a missing indicator. `pyampute` then would apply the missingness indicator, inserting `NaN`s for all the specified target features and samples.

4.1.1.2 Data Transform Setup

We then learned the parameters for the data transforms on the train split of the input data and applied it to the train, validation, and test splits for both the input data and ground truth. We could not use separately trained data transforms for the ground truth as it would leak unseen (observed) data in the model training and would have left the two datasets misaligned and incomparable. If the input data were semi-observed, as a subset or the entire dataset, we set the data transforms using the observed values, which passed the missing values forward untouched.

4.1.1.3 Scale Continuous

Same as before, we first applied minimum-maximum scaling on the continuous features, where the minimum and maximum were obtained from the train split of the input data.

Usually, this scaling method is guaranteed to produce outputs between 0 and 1 in the input dataset. But because there were unseen values in the ground truth data that were not amputed, we could have produced values less than 0 and greater than 1 in the ground truth dataset as a result.

4.1.1.4 Feature Map

After the continuous features were scaled, we optionally applied a feature mapping in order to train the model on data in the same feature-type space, because data for continuous and categorical features follow different distributions. We independently applied the following feature mappings:

- No mapping; keep mixed categorical (binary and one-hot encoded multicategorical) and continuous features to impute in mixed feature space.
- Discretize continuous features to impute entirely in categorical space.
- Target encode categorical features to impute entirely in continuous space.

Discretization We continued to use minimum description length (MDL) discretization. Here, we did not impose a uniform distribution on the missing values after in order to keep the feature mapping component aligned with its continuous counterpart. The lowest and highest value-ranges learned by MDL were bounded by the minimum and maximum values observed in the input dataset (0 and 1 after scaling). However, this time, when we applied the learned bins, the bins at the edge of the ranges were unbounded at the low and high ends, respectively. For example, if the smallest bin was $[0, 0.2)$ and the largest was $[0.829, 1]$, the discretization rule applied for those bins were $(-\infty, 0.2)$ and $[0.829, \infty)$, respectively. This rule allowed us to discretize both the input data and ground truth with equivalent bins, using bins learned on the input dataset, even though the ground truth might have had values outside the strictly bounded ranges learned on the input data.

Target Encoding Unlike our previous iteration, we included target encoding. Target encoding would map each categorical value to a probability estimate based on the prediction task labels, producing a continuous feature that correlates with the target [Mic01]. For each categorical feature, categorical values for each sample n were encoded with a mixture of the posterior probability of the target y_n given the category c_n $\Pr(y_n \mid c_n)$ and the prior probability of the target $\Pr(y_n)$.

4.1.2 Imputation

We passed the scaled and potentially feature-mapped data and ground truth to the autoencoder model. However, autoencoders cannot inherently handle missing values, so we first temporarily filled in missing values in the input data with 0. In previous work, we also tried temporarily filling values with a per-batch simple imputation (mean impute continuous features and mode impute categorical features) to enable Autopopulus to implement and compare existing autoencoder-led imputation literature. However, we only filled missing values with 0 in this work to disentangle the performance of the autoencoder for imputation with the simple imputation, as we saw in Sec. 3.3.4.

4.1.2.1 Autoencoder Architecture

We used Autopopulus to flexibly build many types of autoencoders. Autoencoders may be adjusted by:

- The number of layers (depth) and the size of each layer (overcomplete vs. undercomplete).
- Variational structure that adds two layers at the code index to output the learned means and log-variances, transforming the autoencoder into a generative model.
- Non-linear activation functions.

- Deep learning regularization techniques such as interlayer dropout or batch normalization, or corruption of the input layer either by batch-swap or replacing values with 0 (denoising) [IS15b, Mue].

In this iteration, we introduced batch normalization and batchswap corruption, while the rest were also parameters available for flexibly building an autoencoder in the previous iteration of Autopopulus.

4.1.2.2 Loss

We passed the output of the autoencoder model on the input data into the loss. This time, we treated the binary and one-hot encoded features differently. The reconstruction loss (Eq. 4.1a) could be any differentiable function for each loss component. We weighed each component in Eq. 4.1a equally. When the autoencoder was variational, we added an extra Kullback-Leibler Divergence (KL Divergence) error term in addition to the reconstruction loss (Eq. 4.1b) [KL51].

$$\mathcal{L}_{\text{reconstruction}} = \mathcal{L}_{\text{multicategorical}} + \mathcal{L}_{\text{binary}} + \mathcal{L}_{\text{continuous}} \tag{4.1a}$$

$$\mathcal{L}_{\text{variational}} = \mathcal{L}_{\text{reconstruction}} + \mathcal{L}_{\text{KL-div}} \tag{4.1b}$$

We used Cross Entropy (CE) for each one-hot encoded multicategorical feature, Binary Cross Entropy (BCE) for binary features, and element-wise (EW) Mean Arctangent Absolute Percent Error (MAAPE) for continuous features (see 4.1.2.4 for more on EW metrics) [KK16b]. If we target encoded the features so that all features were continuous, then $\mathcal{L}_{\text{multicategorical}} + \mathcal{L}_{\text{binary}} = 0$ and $\mathcal{L}_{\text{reconstruction}} = \mathcal{L}_{\text{continuous}}$. If we discretized the features to be categorical, then $\mathcal{L}_{\text{continuous}} = 0$ and $\mathcal{L}_{\text{reconstruction}} = \mathcal{L}_{\text{multicategorical}} + \mathcal{L}_{\text{binary}}$. If we applied a feature map, the loss was computed in feature-mapped (target encoded or discretized) space, not in the original feature space. We wanted to train the model in mapped space, and additionally, the feature mapping inversions were not differentiable. When we did not use the fully-observed subset to train the model, the ground truth would have missing values.

In this case, we only computed the loss on the observed values, training the model on the data we did have.

4.1.2.3 Training

We ran training on the training split with the Adam optimizer [KB14]. We tuned the model using the validation split on the number of layers and size of layers, learning rate, L2 penalty, and batch normalization layers using automatic sweep of hyperparameters with asynchronous hyberband scheduling (ASHA) [LJR18]. While Autopopulus supports batch normalization and dropout, we only used batch normalization for the purposes of our experiments as it is not advised to use both together, and batch normalization helps the model converge more quickly [LCH18]. Additionally, we manually tuned the model on the maximum number of epochs, early stopping patience, activation function, and batch-size.

4.1.2.4 Model Output and Metric Evaluation

If we applied a feature map, we produced the output and evaluated the imputation performance in both the feature-mapped and original feature space. All our metrics were evaluated under two different reductions: column-wise (CW) and EW, which could potentially be different. Otherwise, the metrics were always in the original feature space. If we were evaluating in the original feature space when the data were feature-mapped, we first applied a feature-mapping inversion to the model output. Note that it was unnecessary to invert the original input data and ground truth, as we already had those data in the original feature space. The following steps applied to both mapped and original-space metrics. We used the sigmoid function for all binary features and the softmax function for each one-hot encoded multicategorical feature. We kept the originally observed values in the input data as-is, and we only filled in values from the model where they were missing in the input data. Note that this diverged from how the loss was computed on all data, which was meant to train

the model to learn the data manifold. Conversely, the task of imputation only required us to estimate the missing values. We then compared these imputed data to the (either original or mapped) scaled ground truth data to evaluate imputation performance.

Column-wise (CW) versus Element-wise (EW) Metrics Assume a dataset $\mathbb{D}$ of N samples and M features, where all values are observed, notated with `(obs)`, except for three values. The difference between the true values in $\mathbb{D}$ and the imputed dataset $\hat{\mathbb{D}}$ results in errors y_i:

$$\hat{\mathbb{D}} - \mathbb{D} = \begin{bmatrix} \cdots & y_1 \text{ (obs)} & y_3 & \cdots \\ \cdots & y_2 & y_4 & \cdots \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix} \tag{4.2}$$

The column-wise (CW) RMSE for only the missing values in this example would be

$$\frac{\sqrt{\frac{y_2^2}{N-1}} + \sqrt{\frac{y_3^2+y_4^2}{N}}}{M} \tag{4.3}$$

On the other hand, the element-wise (EW) RMSE for only the missing values in this example would be:

$$\sqrt{\frac{y_2^2 + y_3^2 + y_4^2}{N \cdot M - 1}} \tag{4.4}$$

If $y_2 = 4$, $y_3 = 9$, and $y_4 = 2$, then the CW-RMSE would be 1.753, while the EW-RMSE would be 3.030.

The scaled CW-MAAPE for only the missing values in this example would be

$$\frac{2}{\pi} \frac{\frac{\arctan\left(\left|\frac{y_2}{d_2+\epsilon}\right|\right)}{N-1} + \frac{\arctan\left(\left|\frac{y_3}{d_3+\epsilon}\right|\right)}{N}}{M} \tag{4.5}$$

On the other hand, the scaled EW-RMSE for only the missing values in this example would be:

$$\frac{2}{\pi} \frac{\arctan\left(\left|\frac{y_2}{d_2+\epsilon}\right|\right) + \arctan\left(\left|\frac{y_3}{d_3+\epsilon}\right|\right) + \arctan\left(\left|\frac{y_4}{d_4+\epsilon}\right|\right)}{N \cdot M - 1} \tag{4.6}$$

If under the same errors, then the CW-MAAPE would be 0.333 while the EW-MAAPE would be 0.273.

Feature Map Inversion To invert discretized data, we selected the bin with the highest score or pseudo-probability and took the mean of the range. Recall that the bins at the lowest and highest ends were unbounded when applying discretization to both the input data and ground truth. However, upon inversion of the model output, we used the strict bounds to produce a continuous value from the bin. Concretely, if the smallest bin was $[0, 0.2)$ and the largest was $[0.829, 1]$, the inversion for those bins were 0.1 and 0.9145.

To invert target encoded data, we selected the encoded value most similar to the model output value based on L1 distance and then replaced the encoded value with its corresponding category. For example, consider the case where we mapped category A to 0.1, category B to 0.5, and category C to 0.9, and the model output for that feature was 0.428. We would invert the value 0.428 to category B, as 0.5 would be the closest matching encoded value based on L1 distance.

Validity of Discretization Inversion We cannot directly apply the discretization learned on the train split of the input data on the ground truth when amputing because it has unseen or observed value that are missing in the input data.

Given some mock data in Table 4.1, say a discretizer learns the following bins for age: $[10, 15)$, $[15, 20)$, and $[20, 25]$. Note that the ground truth has values 59 and 3 that are unseen in the input data, which are both outside the ranges of the strict 2-sided bins. Attempting to directly apply these ranges will result in the data in Table 4.2, where the values not within any ranges are unable to be inverted and are replaced with NaN. If we unbound the edge bins such that the discretization instead applies < 15, $[15, 20)$, and ≥ 20, the discretization will result in Table 4.3.

Semantically, ground truth has the same number of bins, but the outer edges represent

Table 4.1: Example of Age Data

Ground Truth Age	Data Age
59	NaN
10	10
25	25
3	NaN

Table 4.2: Example of Age Data Discretized with Bounded Bins

Ground Truth Age Discretized	Data Age Discretized
NaN	NaN
$[10, 15]$	$[10, 15]$
$[20, 25]$	$[20, 25]$
NaN	NaN

a potentially wider range of values than for the input data (i.e., contains the sub-range that is observed in the input data). For the same discretization of the ground truth (if the bins were literally larger), the input data would still fall into the same bin and be discretized the same way. In other words, the unbounded version still refers to the same categorical classes for both ground truth and the input data, however the input data only has access to its own observed minimum and maximum bounds that are realized upon feature inversion.

For example, if the model estimates that the age for the first sample falls in the bin ≥ 20, upon inversion the model will map that bin to the continuous value 22.5 as the mean of the strict bounded bin $[20, 25]$. Although the original value was 59 and dramatically far away from the guessed 22.5, this discrepancy is acceptable because it reflects the blind spots to the discretization caused by amputation.

Table 4.3: Example of Age Data Discretized with Unbounded Bins

Ground Truth Age Discretized	Data Age Discretized
≥ 20	NaN
< 15	< 15
≥ 20	≥ 20
< 15	NaN

4.1.3 Prediction

We applied the tuned and trained autoencoder imputation model to the train, validation, and test splits. If we applied a feature map, we only saved the imputed data in the original space, not the mapped space. We passed the scaled imputed data to a predictor model, which needed to handle mixed feature types (as the data are in the original space). We trained the predictor on the train set, tuned on the validation set, and evaluated on the test set. We then compared the predictions to the corresponding labels to evaluate downstream predictive performance.

4.1.4 Implementation

We implemented the system[1] using Python, `Pandas` [McK10] and `NumPy` [Oli06] for data processing, `Scikit-learn` [PVG11] for baseline models and basic data transforms such as minimum-maximum scaling, `lightgbm` [KMF17] for the LGBM models, `Orange` [DCE13] for MDL discretization, and `category_encoders` for target encoding, `PyTorch-Lightning` [PGM19, Fa19] for deep learning model implementation and training execution, `Ray` [MNW18] for model tuning, `pyampute` for amputation [SZS22b], `guildai` [Smi23] and `MLflow` [ZCD18] for experiment tracking, and `hypothesis` [MHm19] for unit testing. All methods were de-

[1]The code for the Autopopulus framework is open-source and available at https://github.com/davzaman/autopopulus.

terministic where allowed by the framework/hardware with a seed of 42.

4.2 Amputation

Amputation is the inverse of imputation; it is the process of masking or removing data, and therefore introducing missingness. To properly evaluate and characterize imputation models, we needed to be able to control the missingness scenarios under which those models would impute. Multivariate amputation was originally developed by Schouten et al. in the statistical language R with the function `ampute` [SLV18] in the package `mice` [VG11]. `pyampute`[2] implements multivariate amputation in Python, and extends it with additional functionality [SZS22c]. We assisted Schouten et al. in transferring the functionality to Python and expanding features.

We were able to impose a single type of missingness for chosen features jointly at varying amounts of missingness in order to characterize imputers under particular missingness mechanisms. However, in real-world processes and consequently in real-world data, there is usually a multitude of missingness mechanisms occurring at once on different combination of variables. This flexible framework also allowed us to dictate multiple patterns of missingness within a single dataset to evaluate imputation on these mixed-mechanism missingness scenarios. We are currently developing future features and methodology.

4.3 CURE-CKD Updated

While the target task and features did not change, when we refreshed the CURE-CKD Registry, the statistics of the data changed. Instead of 12 years of data from 2006-2017, our data now included 15 years of data from 2006-2020. We provide a more detailed breakdown of the updated CURE-CKD Registry below.

[2]The implementation is available at `https://rianneschouten.github.io/pyampute/`.

	Site Source	Providence		UCLA	
	Patient Cohort	At-Risk	CKD	At-Risk	CKD
Rapid Decline	No RKFD	1,924,650	447,970	380,968	86,452
		(99.49%)	(98.29%)	(99.60%)	(98.68%)
	RKFD	9,957	7,813	1,520	1,156
		(0.51%)	(1.71%)	(0.40%)	(1.32%)

Table 4.4: Details the counts and percentages of outcome breakdown of RKFD for patients across UCLA and PSJH, and whether they have or are at-risk for CKD.

4.3.0.1 Descriptive Statistics

Table 4.4 elucidates the breakdown of outcomes per site source and CKD diagnosis, while Fig. 4.3 shows a deeper breakdown by race and ethnicity across both sources of data only for CKD patients. On a macro-level, there were $N = 2,860,486$ patients in the registry across the two healthcare systems. UCLA patients made up 16.43% ($N = 470,096$) of those patients. In total, there were $543,391$ (19%) patients with CKD, and $20,446$ (0.71%) patients that experienced RKFD, making this dataset largely imbalanced.

During the entry period there were 310 features total, 308 features missing less than 89% data, and $41,611$ CKD patients who were not missing any data. Of those fully-observed patients, 318 were from UCLA (0.76%), $41,293$ were from PSJH (99.24%), and 877 (2.11%) experienced rapid decline while $40,734$ (97.89%) did not. When the data were discretized into categorical features (see 4.1.1.4), there were 336 features, and when the data were target encoded into continuous features (see 4.1.1.4), there were 31 features.

Missingness Profile While the overall dataset was missing 20.15% of data, there were high levels of missingness across certain patients and features in the data. For example, Fig. 4.4 shows systolic blood pressure and A1c, both significant clinical risk factors for CKD,

were missing a significant number of values even in the earlier years of the registry where patient dropout was much less. Other features were much less sparse, such as age or the count of ambulatory visits. A deeper breakdown is shown in Fig. 4.5.

4.4 Evaluating CURE-CKD Under a Microscope

We evaluated Autopopulus across ten missingness scenarios, five autoencoder variations, and two baseline imputation methods. Compared to our first application of Autopopulus to the CURE-CKD dataset, we were more focused on predictive performance under the missingness scenarios rather than separately evaluating predictive performance on the entire dataset. Therefore, while functionally, we could execute the same analyses before, we used a different: flow of data, metric scheme, baseline methods, and predictive models.

4.4.1 Missingness Scenarios

Like before, we amputed at low and high percent missingness (33% and 66%, respectively) across all mechanisms. However, this time, we investigated five missingness mechanisms: MCAR, MAR, MNAR, MNAR(Y), and MNAR(G). MCAR and MAR here both refer to their normal definitions. However, we broke up MNAR into recoverable and non-recoverable scenarios (see Sec. 4.1.1.1).

To convert the score into a probability of being missing, we used sigmoid-mid and sigmoid-tail. The former assigned higher probabilities to be missing to more average scores (in the middle). In comparison, the latter assigned higher probabilities to be missing to the more extreme scores (in the tails).

We selected the HbA1c entry period mean and hypertension indication at study entry to be missing. When data were MAR, we used the age at entry and the average ambulatory visit count at entry period (the first 90 days of observation) to jointly influence the missingness of the HbA1c and hypertension features. The features we selected to be missing are

clinically important factors for predicting RKFD, and the features we selected to influence the missingness are reasonable causal factors validated by nephrologists.

As a summary experimental grid for exploring missingness scenarios with more scrutiny on the CURE-CKD dataset:

- Percent missing: 33%, 66%.
- Missingness mechanism: MCAR, MAR, MNAR, MNAR(Y), MNAR(G).
- Score to probability function: sigmoid-mid, sigmoid-tail.
- Features missing: HbA1c mean during entry period, hypertension indication at study entry.
- Observed features that influence missingness: age at entry, ambulatory visit count during entry period.

4.4.2 Imputer Models and Feature Mappings

Whereas before, we implemented other existing instantiations of autoencoder-led imputation in the literature, here, we compared generic autoencoder variations. We evaluated across a vanilla autoencoder, denoising autoencoder (DAE), autoencoder with batchswap corruption, variational autoencoder (VAE), and denoising variational autoencoder (DVAE). For all the autoencoder methods except the variational ones (VAE and DVAE), we tried all three feature spaces: mixed categorical and continuous, categorical only through discretization, and continuous only through target encoding. However, as VAEs all assume each feature follows a normal distribution, it imposes that all features must be continuous. Therefore, for VAE and DVAE, we only applied target encoding and did not allow mixed or categorical-only feature spaces.

We limited training to 100 epochs, with early stopping patience of 2, rectified linear unit (ReLU) activation function, and batch size of 256. These were determined by manually

tuning, otherwise the following were dynamically tuned per experiment. We tuned the hidden layers between {[0.5, 0.25, 0.5], [0.5], [1.0, 0.5, 1.0], [1.5], [1.0, 1.5, 1.0]} where the decimal values represent a fraction of the original sample size. For example, $[0.5, 0.25, 0.5]$ would have two hidden layers in the encoder: one to reduce from all N features to $0.5 \cdot N$, then another to reduce that in half again $0.25 \cdot N$. The decoder would mirror that in reverse. We tuned the inclusion of batchnorm regularization. We tuned the learning rate, sampling from a log-uniform space between $1 \cdot 10^{-5}$ and $1 \cdot 10^{-1}$. Lastly, we tuned the L2 penalty sampling from a log-uniform space between $1 \cdot 10^{-5}$ and 1. We sampled from the tuning grid for 30 trials for each experiment.

4.4.2.1 Baseline Imputation Methods

Our baseline methods for imputation similarly included simple imputation and KNN, however, since we needed to handle mixed feature data we removed MICE. Both simple imputation and KNN have continuous and categorical versions that can be applied. We did not employ feature mappings, as these methods served as a basic comparison. KNN for continuous features was tuned between using $[3, 5, 10]$ nearest neighbors and treating all neighbors uniformly versus weighing their contribution by distance. KNN for categorical features only used the nearest neighbor but was tuned on uniform versus distance weights. The data processing, imputation, and prediction pipeline look similar to Figs. 4.1 and 4.2, however, there was no feature mapping and no inverse feature mapping step, no fill 0 step, and no loss.

4.4.3 Evaluation

Imputation Performance We evaluated the imputation models on the validation and test splits of the data in both original and mapped space. All metrics, except for the loss, were computed on the missing values only. All metrics have a continuous and categorical

component, where if we applied a feature map, the corresponding non-existing component will have become 0. We computed the Root Mean Squared Error (RMSE) and Mean Arctangent Absolute Percent Error (MAAPE) for continuous features and categorical error for categorical features. The MAAPE was scaled between $[0, 1]$ by multiplying by $\frac{2}{\pi}$. If we applied a feature-mapping, we computed the metrics in both the mapped feature space and the original feature space (after inverting). Additionally, we computed the metrics both column-wise (CW) and element-wise (EW). CW metrics were first aggregated per column and then averaged over all columns. EW metrics were aggregated over all elements.

We ran an extra set of experiments to evaluate the best imputation model per missingness mechanism for imputation performance, which we did not do before. We ran bootstrap evaluation of the imputation model on the test set from the fully observed model on 100 bootstrap samples. We limited to the best per mechanism instead of for all models in the interest of time and storage limitations.

When bootstrap evaluating, we report 95% confidence intervals as symmetric intervals with a magnitude of $1.96 \cdot$ standard error.

Imputation Performance Legend Key Each legend for graphs depicting imputation performance follows the same structure, described as follows. Each color represents a different feature space: brown corresponds to mixed continuous and categorical, blue to target encoded, and red to discretized. Each shape size represents the amount of missingness: smaller for 33% and larger for 66%. Each shape represents a different score to probability function: a filled in circle for sigmoid-mid and a hollow X for sigmoid-tail.

Prediction Performance We similarly used random forest (RF) as a predictor model, however, instead of a logistic regression we used a light gradient-boosting machine (LGBM). We picked RF and LGBM as they can handle mixed features and perform well across a wide set of tasks. The RF model was tuned over the grid: $[5, 10, 15, 20, 25, 30, 35]$ estimators,

and a maximum depth between 3 and 10 inclusive. The LGBM model was tuned over the grid: $[5, 10, 15, 20, 25, 30, 35]$ estimators, a maximum depth between 3 and 10 inclusive, and a learning rate of $[1 \cdot 10^{-5}, 1 \cdot 10^{-3}, 1 \cdot 10^{-1}]$. Like before, we bootstrapped our predictive performance by bootstrap sampling the model outputs 100 times and evaluating (test-set bootstrap only).

While we did not run predictive performance experiments on the entire dataset as-is as before, we ran an extra set of experiments to evaluate the best autoencoder imputation model per missingness mechanism for predictive performance. To mirror the previous experiments more closely to inform predictive performance changes based on the imputation method chosen, we retrained the best autoencoder model per missingness mechanism on the fully observed subset and applied it to the remaining semi-observed subset for downstream prediction.

Predictive Performance Legend Key Each legend for predictive performance follows the same structure, described as follows. Each color represents a different feature space: brown corresponds to one-hot mixed continuous and categorical, blue to target encoded, and red to discretized. Each shape size represents the amount of missingness: smaller for 33% and larger for 66%. Each shape represents a different predictor model: a filled in plus sign for LGBM and a hollow diamond for RF. The 95% confidence intervals are represented by the horizontal bars.

4.4.3.1 Imputation Performance

We compared imputation performance across multiple parameters. The parameters from the missingness scenarios were the mechanism, missing percentage, and score to probability function. The parameters from the imputation methods were the models and the feature mappings. Lastly, the parameters from the evaluation methods were the different continuous feature metrics (since we only used one metric for categorical features), reduction methods,

and feature spaces. When controlling for a particular metric characteristic, the default would be the CW RMSE metric in the original space. For example, when comparing the different continuous feature metrics, they would be CW in the original space, but when comparing CW and EW metrics, they would be MAAPE in the original space. We chose this default as CW is semantically more intuitive as a reduction for metrics than EW, and MAAPE is differentiable and therefore also used in the loss compared to RMSE, and we were interested in imputing the original dataset, not the mapped version. All metrics are reported on the test set except for the loss, which is reported for the validation set.

Imputation performance was slightly different depending on the imputation method used, and varied depending on the metric used (Fig. 4.6). Of all the imputation methods, the KNN tended to perform the worst, with the autoencoders generally outperforming both baseline models. Of the autoencoder methods, the vanilla autoencoder tended to perform the best and variational flavors tended to perform the worst. Similarly, those that target encoded the categorical features tended to have the worst imputation performance, while leaving the features mixed tended to perform the best or similarly to discretizing continuous features. With a few rare exceptions, the models performed better when fewer data were missing. The baseline methods were generally very similarly behaved across missingness mechanisms, while there were slight variations in performance for the autoencoder imputers across the mechanisms. When target encoding, the vanilla and batchswap autoencoder methods that impute under sigmoid-tail tended to perform worse than sigmoid-mid under MNAR(Y) and MNAR(G). However, this trend reversed for the other mechanisms.

Imputation Performance by Metric (Fig. 4.6) Under MAAPE, the baseline methods performed more poorly relative to the other methods than when evaluated using RMSE for the continuous features. Under RMSE, KNN performed similarly for 33% and 66% missing, while there was a more pronounced difference under MAAPE. The autoencoder methods had more consistent trends between the two metrics, though the performances were slightly

Table 4.5: Bootstrapped Test Imputation Performance for Best Autoencoder Imputation Methods per Mechanism

Mechanism	Percent Missing	Score to Probability Missing	Method	Feature Mapping	MAAPE & CategoricalError
MCAR	33.0	Sigmoid (Mid)	Batchswap	Mixed Features	0.5225 +/- 0.0017
MAR	33.0	Sigmoid (Mid)	Vanilla	Mixed Features	0.5093 +/- 0.0018
MNAR	33.0	Sigmoid (Mid)	Vanilla	Mixed Features	0.5229 +/- 0.0019
MNAR(G)	33.0	Sigmoid (Mid)	Vanilla	Mixed Features	0.5131 +/- 0.0018
MNAR(Y)	33.0	Sigmoid (Mid)	Vanilla	Mixed Features	0.5016 +/- 0.0018

more spread out for RMSE.

Imputation Performance by Metric Reduction (Fig. 4.7) It seems that for CURE-CKD, there were no differences between CW and EW reductions, even though, in theory, they can produce different values.

Imputation Performance by Feature Space (Fig. 4.8) There were no mapped performances for the baselines as we did not use feature mappings for the baseline models. We observed a notable improvement in performance for the target encoded methods in the mapped space, or rather, a drop in performance after inverting to the original space. In the mapped space, the target encoded methods generally outperformed their discretized counterparts. There seemed to be a spike in performance loss for VAE under MNAR(Y). Overall, the error in the mapped space was less than in the original space.

Bootstrapped Imputation Performance (Table 4.5) We selected the best performers based on the default metric: the CW MAAPE and categorical error metric in the original

feature space. We saw the worst performance over each mechanism under MNAR. The second worst performance was under MCAR. All models tended to have very similar 95% confidence intervals, though the widest was under MNAR. The confidence intervals were tight relative to the averages. As noted above, the vanilla autoencoder performed best under most mechanisms, except for batchswap. We also observed that the lower missing percentage tended to have a lower error, as well as metrics in the mixed feature space.

Loss (Fig. 4.9) We report the last recorded validation loss for the autoencoder methods in the mapped space. The target encoded methods generally converged to the smallest loss, while the discretized methods usually struggled the most.

Training Convergence with Early Stopping (Fig. 4.10) The models tended to converge most quickly when data were MCAR, except for discretized data missing at 66% under sigmoid-mid. The variational autoencoders tended to converge more quickly than the other models. This trend did not hold for target encoded data imputed by the other autoencoder types. There were more models that took longer to converge under MNAR and MNAR(Y).

Training Speed (Fig. 4.11) The most noticeable difference in training speed for the autoencoder methods was dictated by the feature mapping chosen. The slowest training was driven by discretization, followed by target encoding, and then the fastest training was in the mixed feature space where there was no feature mapping and inversion to be done. The feature mappings were also most variable in the confidence intervals of time following the same order, with discretized imputation being the most variable, followed by target encoding, and then finally mixed feature imputation with very little variability in training time. Across the autoencoder methods, batchswap tended to be a little slower than the rest.

4.4.3.2 Predictive Performance

Predictive performance (Fig. 4.12) proved to be different depending on the imputation method used. The baseline models led to the best predictive performance, and LGBM was generally the better predictor. The confidence intervals for all the methods were the tightest for the baseline methods. Simple imputation led to the best PR-AUC when data were MAR compared to the other missingness mechanisms. However, imputation performance differences were not visibly distinguishable for ROC-AUC and the Brier score across missingness mechanisms. Each of the autoencoder method configurations varied greatly in performance for each missingness mechanism. The PR-AUC was quite poor compared to the ROC-AUC.

The baseline models had very slight differences in ROC-AUC and PR-AUC under the different amounts of missingness. However, when fewer data were missing, the calibration of the predictive models improved greatly. Interestingly, the LGBM was better calibrated under simple imputation compared to the RF, but less calibrated under KNN imputation. For the denoising, batchswap, and vanilla autoencoders, target encoding with RF tended to lead to some of the poorest ROC-AUC performance across mechanisms, except for MNAR(G). The variational methods had less spread of ROC-AUC and Brier-score performance across missingness scenarios, however they were equally spread out for PR-AUC. There was no consistent pattern between the feature mapping and Brier-score for the autoencoder methods.

The bootstrapped results on the best autoencoders on the semi-observed remaining subset (Table 4.6) similarly showed LGBM outperforming RF. The LGBM performed best under MCAR and MNAR and worst under MNAR(Y), with the widest 95% confidence interval under MCAR, MNAR, and MNAR(Y). The RF performed best under MNAR and worst under MCAR, with the widest 95% confidence interval under MNAR.

Table 4.6: Predictive Performance on Best Autoencoder Imputation Methods per Mechanism on Semi-Observed Remaining Subset

Mechanism	Percent Missing	Score to Probability Missing	Method	Feature Mapping	Predictor	ROC-AUC
MCAR	33.0	Sigmoid (Mid)	Batchswap	Mixed Features	LGBM	0.7260 +/- 0.0005
					RF	0.4887 +/- 0.0006
MAR	33.0	Sigmoid (Mid)	Vanilla	Mixed Features	LGBM	0.6961 +/- 0.0004
					RF	0.5448 +/- 0.0006
MNAR	33.0	Sigmoid (Mid)	Vanilla	Mixed Features	LGBM	0.7268 +/- 0.0005
					RF	0.6558 +/- 0.0007
MNAR(G)	33.0	Sigmoid (Mid)	Vanilla	Mixed Features	LGBM	0.6961 +/- 0.0004
					RF	0.5448 +/- 0.0006
MNAR(Y)	33.0	Sigmoid (Mid)	Vanilla	Mixed Features	LGBM	0.6319 +/- 0.0005
					RF	0.6046 +/- 0.0006

4.4.4 Model Selection

On the CURE-CKD dataset, we aimed to predict RKFD on CKD patients. Despite the various trends in imputation performance, we saw the best predictive performance when we simple-imputed the data, and particularly when using the LGBM predictor. Due to the simple imputation with the LGBM predictor leading to the best performance across ROC-AUC, PR-AUC, and the Brier-score, we would select that combination moving forward for that dataset and predictive task. If we only cared about imputation, we would likely choose the vanilla autoencoder without any feature mapping. This imputation scheme consistently achieved low errors across missingness scenarios and different evaluation metrics, and trained the fastest.

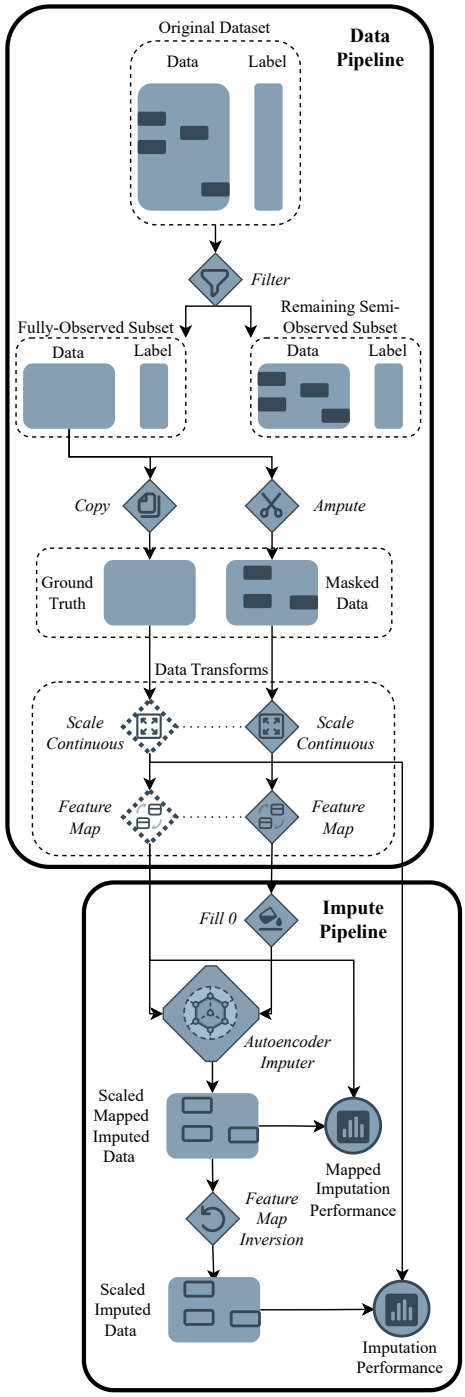

Figure 4.1: Illustration of the data processing and imputation steps of the updated Autopopulus workflow when a fully observed subset of the data was available, and we applied a feature mapping.

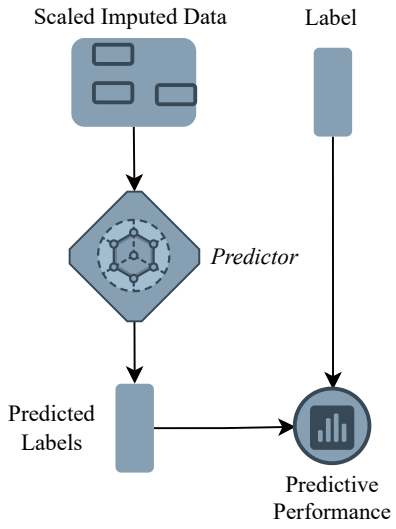

Figure 4.2: Illustration of the prediction step of the Autopopulus workflow.

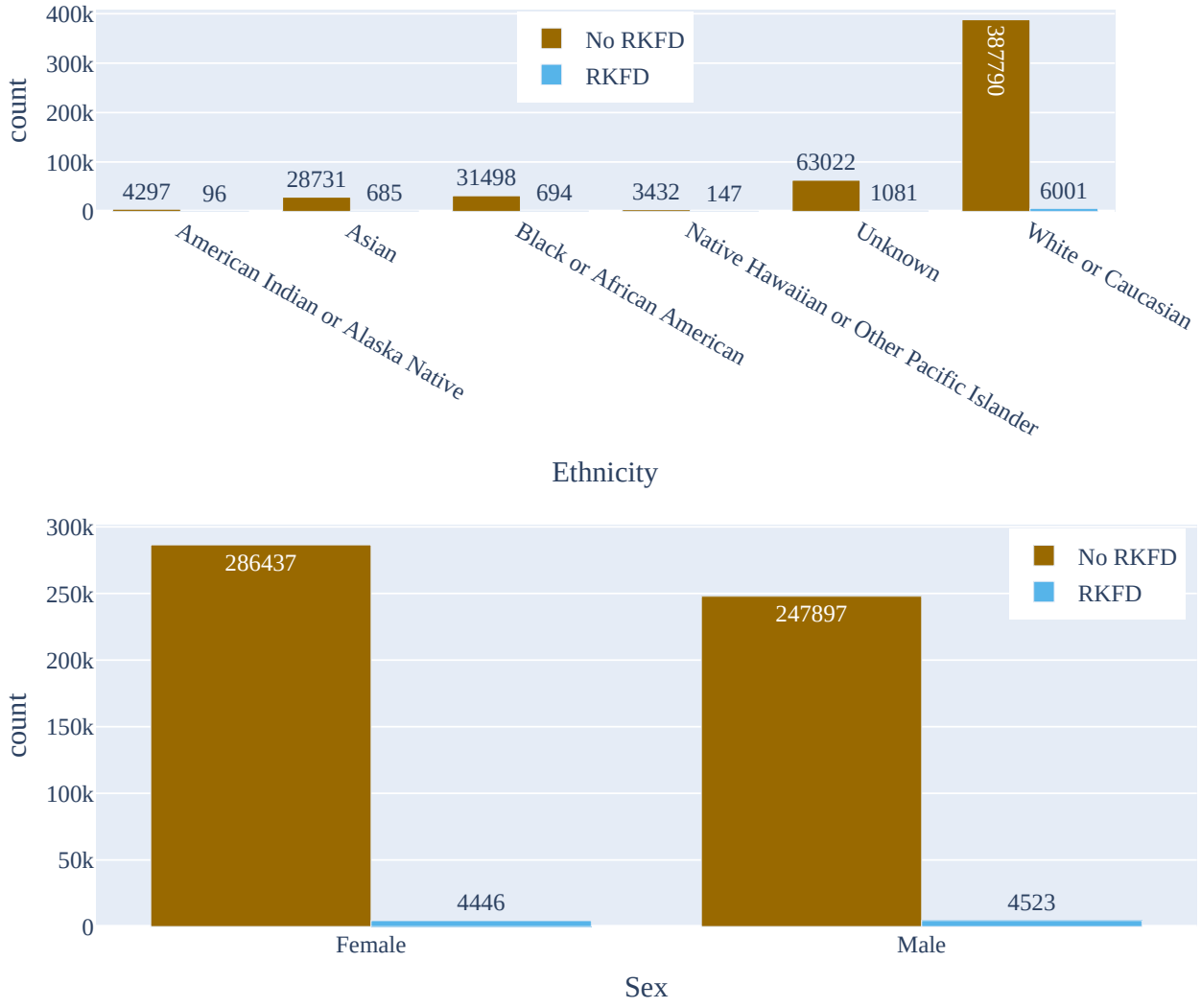

Figure 4.3: The counts of the number of CKD patients at both UCLA and PSJH with and without RKFD broken down by ethnicity and sex.

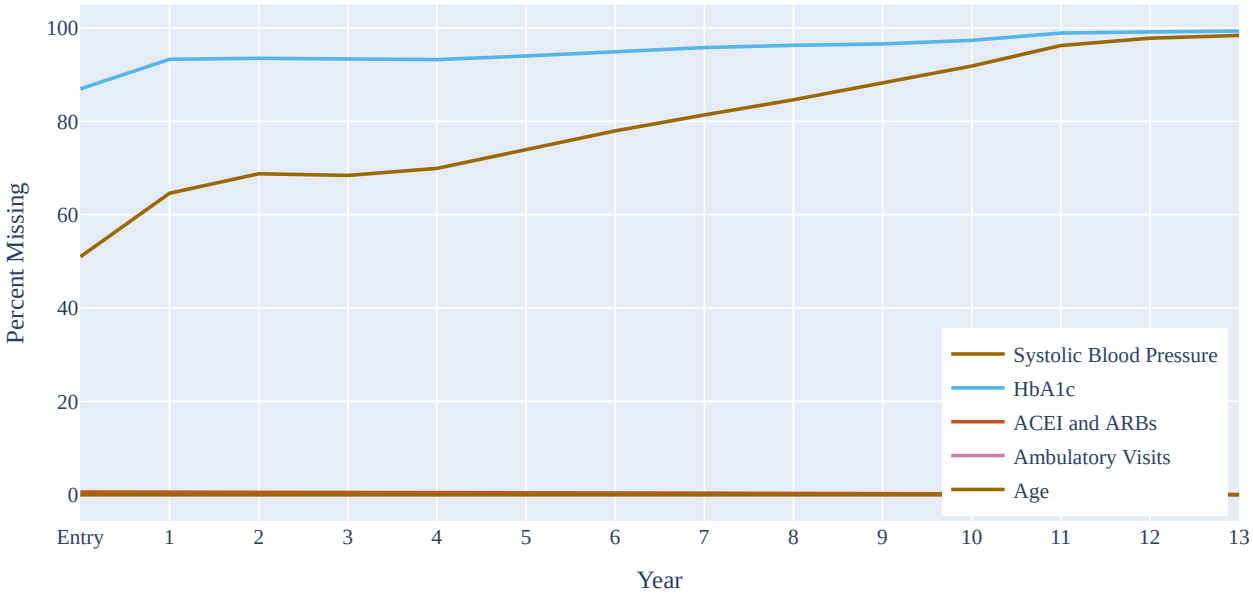

Figure 4.4: Percent of missing data for all patients across time. For all years of the study, a large portion of patients were missing their systolic blood pressure and A1c.

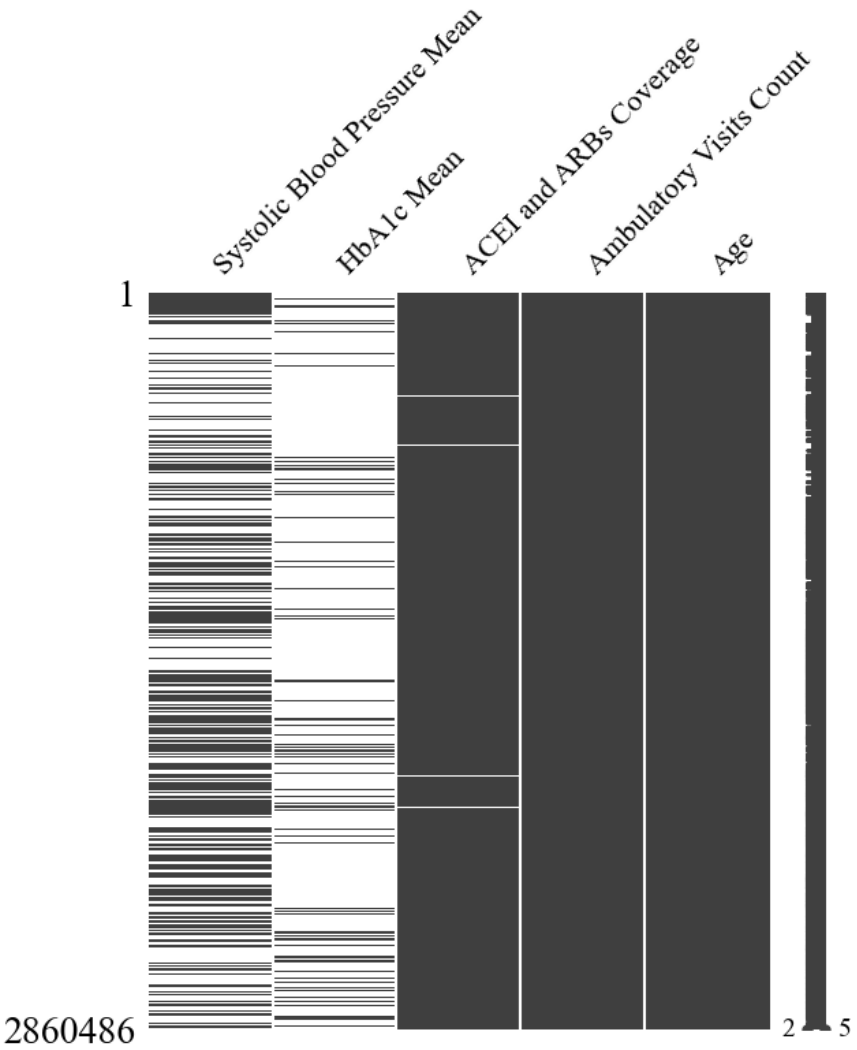

Figure 4.5: Missing values visualized in the CURE-CKD dataset at the entry period. The dark blocks represent observed values, while otherwise white or empty blocks represent missing values.

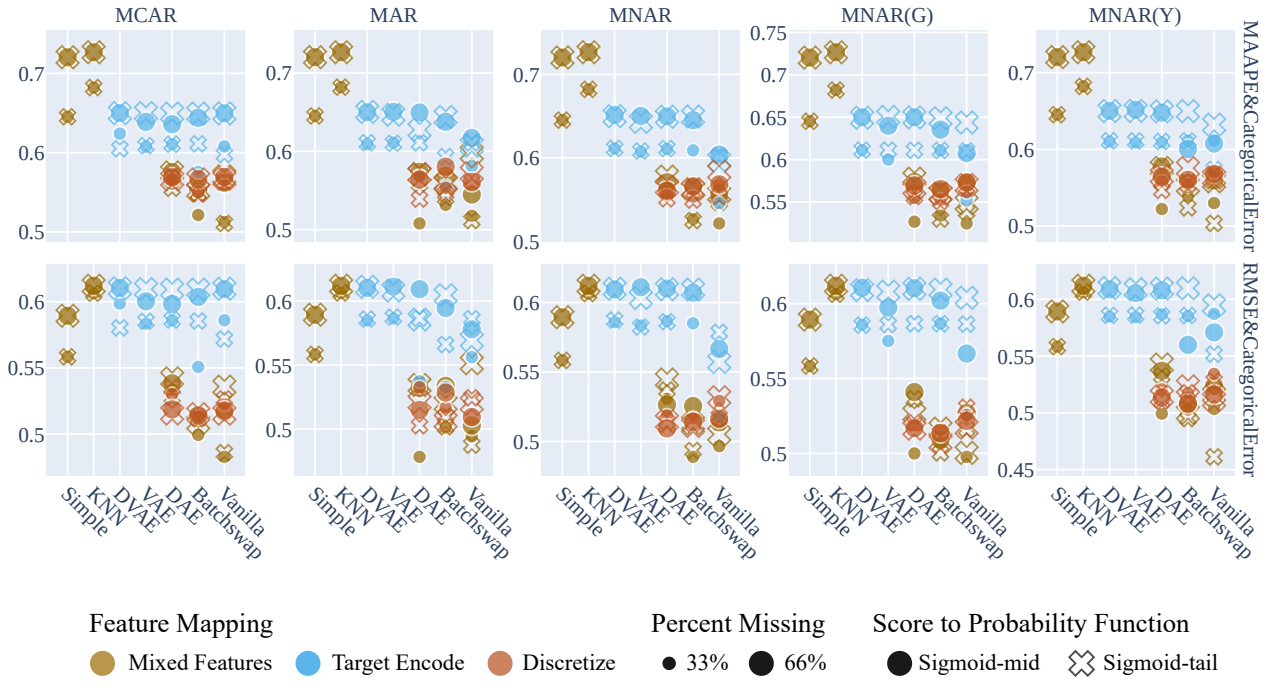

Figure 4.6: Imputation performance for the mixed column-wise (CW) metrics in original feature space on the originally missing values on the test split of the CURE-CKD dataset. Each metric is a row, and each column is a missingness mechanism. The mixed feature-space metric name is represented by `<continuous metric>&<categorical metric>`. The x-axis represents each imputation method, including both baselines and autoencoders. The y-axis represents the metric value. See 4.4.3 for more.

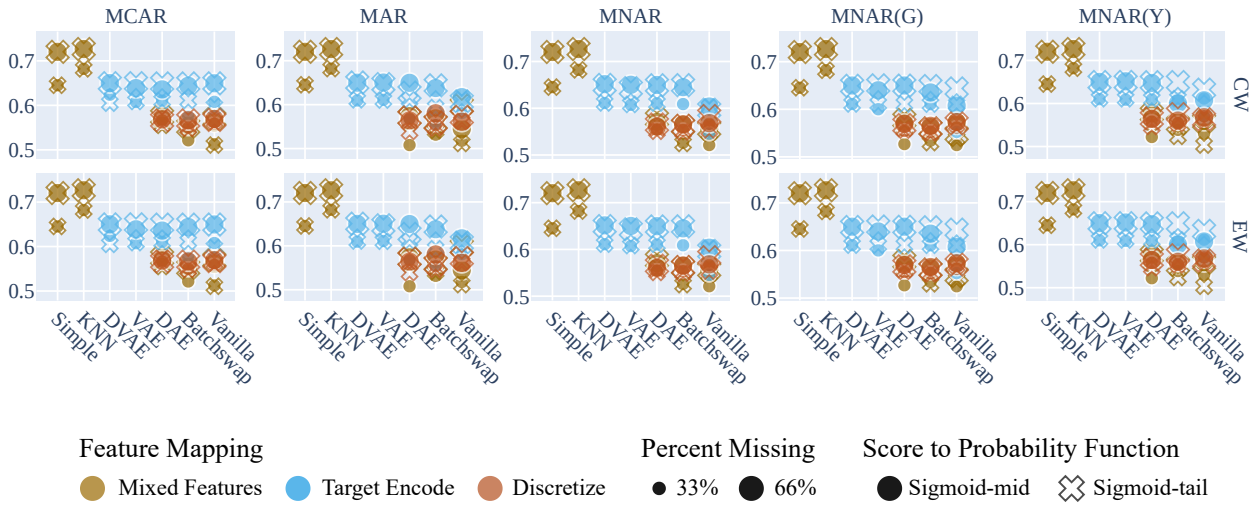

Figure 4.7: Imputation performance for the mixed Mean Arctangent Absolute Percent Error (MAAPE) metric combined with categorical error on both column-wise (CW) and element-wise (EW) metrics in the original feature space on the originally missing values on the test split of the CURE-CKD dataset. Each metric is a row, and each column is a missingness mechanism. The x-axis represents each imputation method, including both baselines and autoencoders. The y-axis represents the metric value. See 4.4.3 for more.

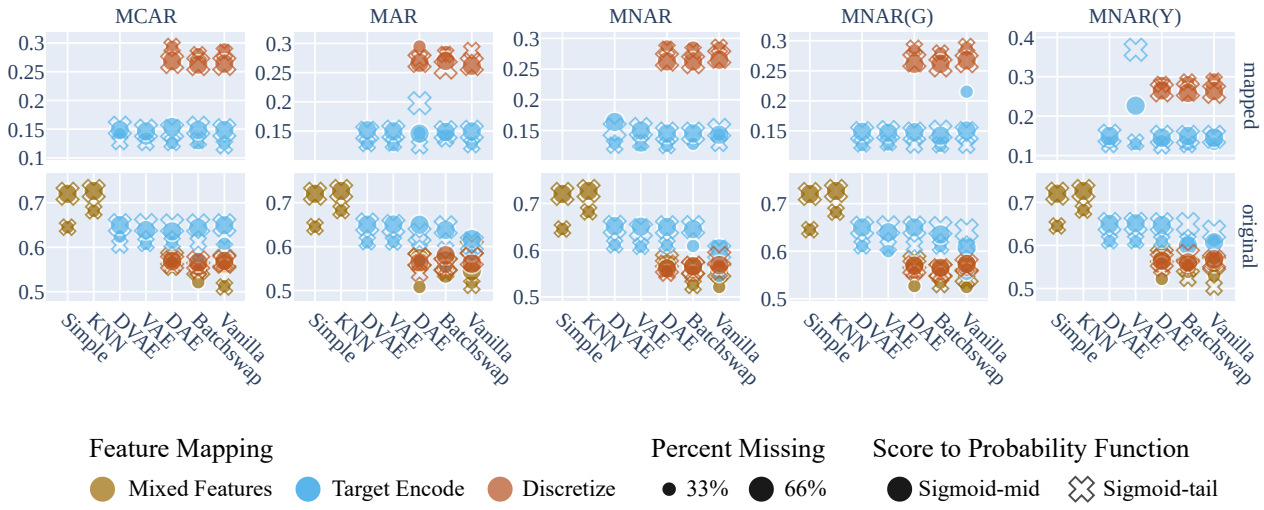

Figure 4.8: Imputation performance for the mixed CWMAAPE combined with categorical error metric in both the original and mapped feature space on the originally missing values on the test split of the CURE-CKD dataset. Each feature space is a row, and each column is a missingness mechanism. The x-axis represents each imputation method, including both baselines and autoencoders. The y-axis represents the metric value. See 4.4.3 for more.

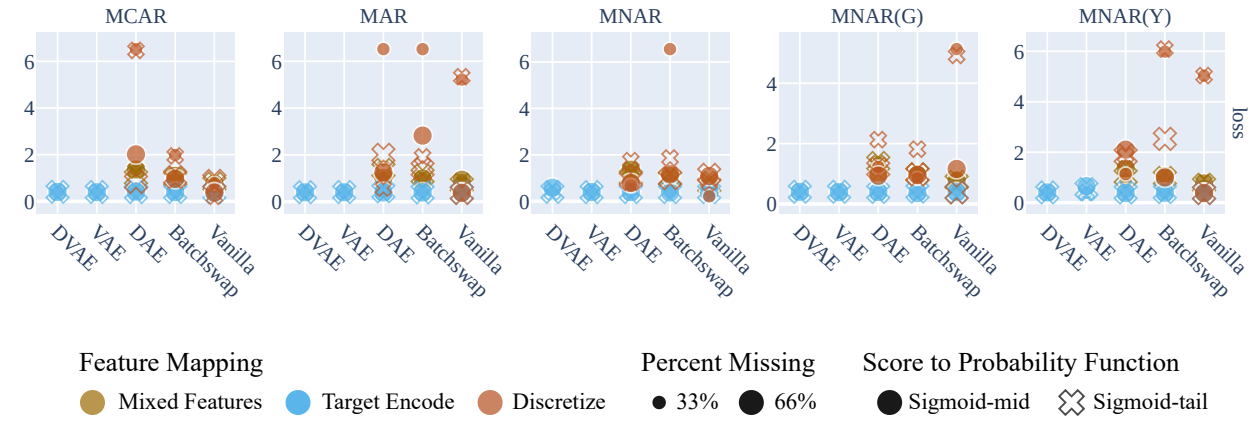

Figure 4.9: Last recorded validation loss in the mapped feature space on all values only in the CURE-CKD dataset. Each column is a missingness mechanism. The x-axis represents each autoencoder imputation method; there are no baselines, as those are trained differently. The y-axis represents the metric value. See 4.4.3 for more.

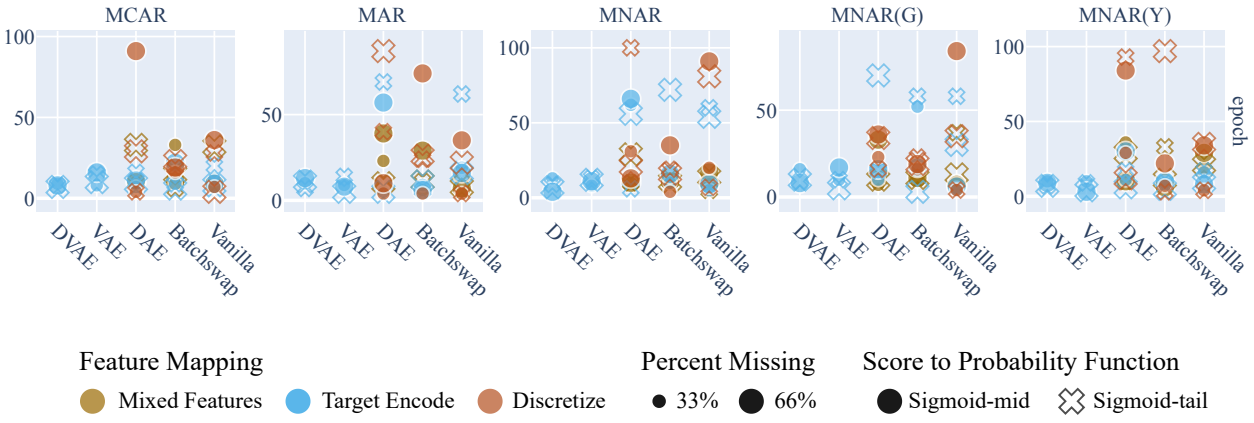

Figure 4.10: Number of final epochs for training convergence when training with a maximum of 100 epochs, and early stopping with patience of 2 epochs. Each column is a missingness mechanism. The x-axis represents each autoencoder imputation method; there are no baselines, as those are trained differently. The y-axis represents the metric value. See 4.4.3 for more.

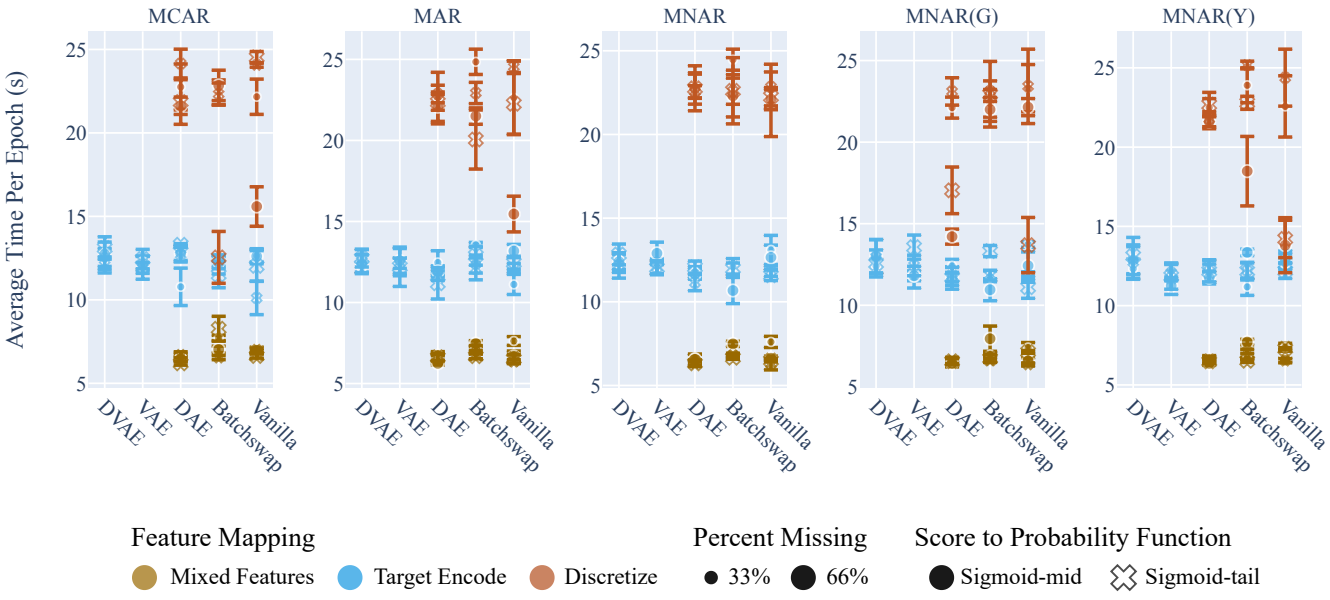

Figure 4.11: The average duration of each training epoch for the autoencoder imputer models. Each column is a missingness mechanism. The x-axis represents each autoencoder imputation method; there are no baselines, as those are trained differently. The y-axis represents the average duration of each epoch in seconds. See 4.4.3 for more.

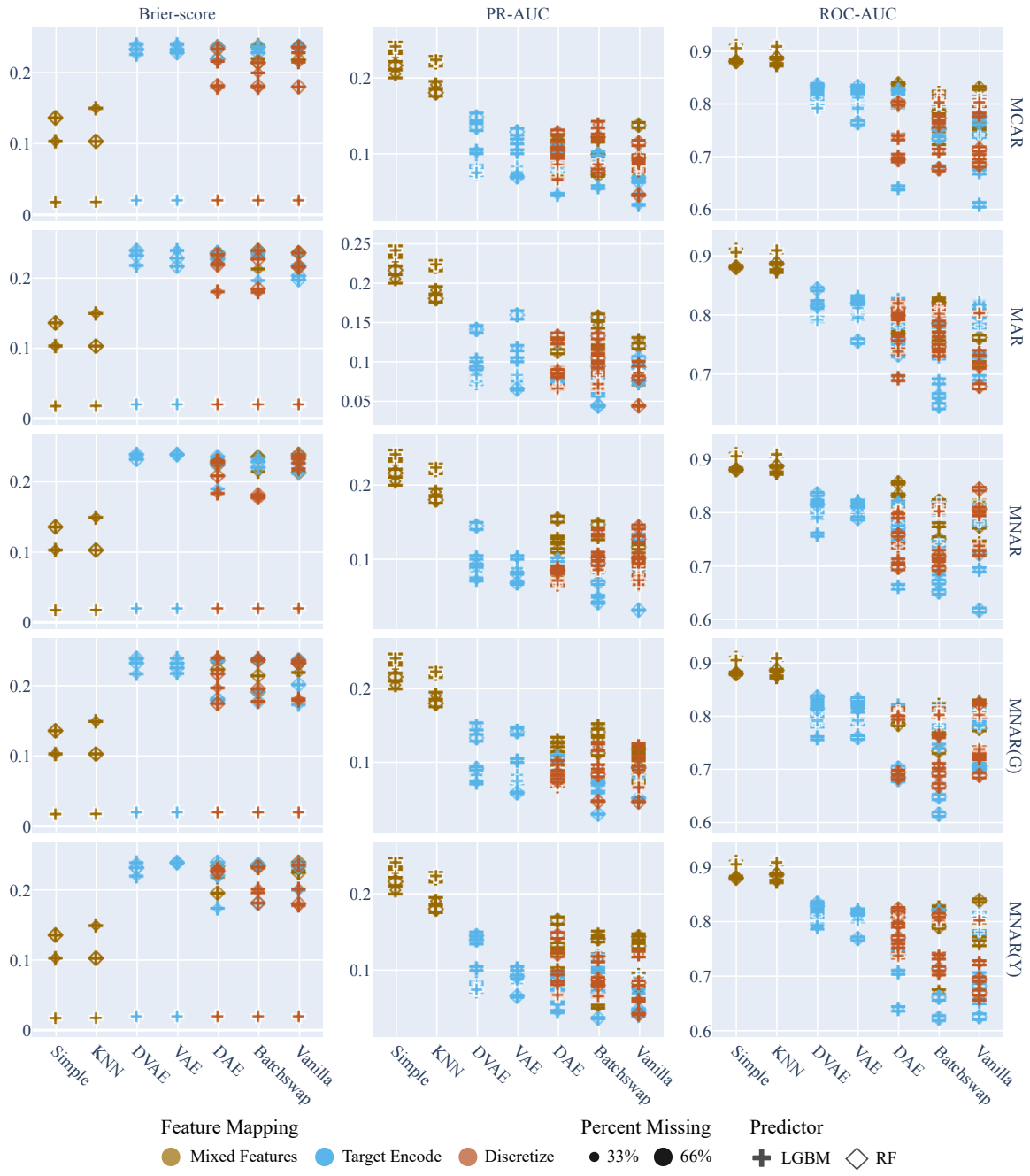

Figure 4.12: Predictive performance on the test split of the CURE-CKD dataset. Each missingness mechanism is a row, and each column is a different predictive metric. The x-axis represents each imputation method, including both baselines and autoencoders. The y-axis represents the metric value.

CHAPTER 5

Expanding to Raw EHR

We previously built and demonstrated Autopopulus using the CURE-CKD dataset, which originated from the CURE-CKD Registry. The registry, while real-world, was derived and already aggregated from raw EHR and also had a large subset of fully-observed samples. However, in many instances, curated data is not common, and neither do they have fully-observed subsets available. We chose to demonstrate the tuning and model selection process of Autopopulus on a raw, real-world EHR dataset, as well as use those results to help build a larger profile of autoencoders for imputation on an external dataset to the CURE-CKD registry. Toward that end, we applied Autopopulus to the continuous renal replacement therapy (CRRT) dataset. The CRRT dataset had no fully observed subset and was drawn directly from the raw EHR available to us. In this case, we could not control the missingness scenarios and therefore could not evaluate imputation performance under the different settings we produced. Instead, we provide a more comprehensive analysis of the predictive task at hand on the CRRT dataset.

5.1 The CRRT Dataset

When kidneys no longer function adequately, many patients are placed on a treatment called *dialysis*, which filters the blood in place of the kidney. Most patients that require dialysis are placed on *hemodialysis*, a process which may only take a few hours. However, this type of dialysis pushes a high volume of blood in and out of the system, causing large fluid shifts. An alternative treatment, continuous renal replacement therapy (CRRT), is a 24/7

gentler form of dialysis that a hospitalized patient might receive because they cannot tolerate regular hemodialysis due to issues with low blood pressure that might be caused by severe infection, liver disease, or heart disease. Due to its nature, CRRT is more costly: it is more invasive and uncomfortable, it requires more time, it requires personnel to monitor for longer, and ultimately there are few machines that need to service a given healthcare institution's population. With the COVID-19 pandemic, CRRT machine allocation became an even more pressing issue. According to [AMF21], "During the first 6 months of the pandemic, the United States is believed to have experienced a shortage of 1088 CRRT machines, with demand outweighing supply in up to eight states." Healthcare providers are then required to solve limited resource allocation issues: who should be put on CRRT, and if someone is on CRRT should they stay on it another day?

We aimed to predict whether we should initiate CRRT for a patient, meaning whether we recommended CRRT. Toward this aim, we collected the largest CRRT dataset to date. It comprised a comprehensive, real-world, longitudinal dataset of raw EHR data of patients from two large healthcare systems (UCLA Health, and Cedars-Sinai (CS)) over a period of 8 years (2014-2021) and their corresponding outcomes on CRRT. The CRRT dataset consisted of three cohorts. The first cohort was the UCLA: CRRT population, which contained adult patients treated at UCLA that were all placed on CRRT. The second was the UCLA: Control population, which contained adult patients treated at UCLA that were considered for CRRT but were not placed on it. This control group was matched to the UCLA: CRRT cohort based on age, sex, and disease status (via the Charlson Comorbidity Index). The last cohort was the CS: CRRT population, which contained adult patients treated at Cedars that were all placed on CRRT.

In this work, we included patients 21 and over and focused only on patients at UCLA who were placed on CRRT, as investigating the wider scope of cohorts is currently under active investigation. We created the binary outcome of recommending CRRT from more granular ones. If a patient's kidney function recovered, or they stabilized to the point of

transitioning to regular hemodialysis, we recommended initiating CRRT for a patient. In contrast, if a patient transitions to palliative care or passes away, we did not recommend CRRT.

We used a patient's demographics, labs, procedures, documented problems, diagnoses, vitals, and medications to predict the outcome. As the EHR were not preprocessed, we first aggregated the longitudinal features over a 7-day window before the start date of each patient's treatment. For continuous feature aggregates, we reported the 7-day mean, standard deviation, number of entries, skew, minimum, and maximum. For categorical feature aggregates, we reported the 7-day total count. All multicategorical features, such as patient race, were one-hot encoded. Due to the sparseness of the data, we dropped features missing more than 80% data.

Descriptive Statistics The UCLA: CRRT population included $N = 3,666$ patients. Compared to the CURE-CKD dataset, this dataset was well-balanced as seen in Table 5.1. There were $9,787$ features missing more than 80% data, such as height, diastolic blood pressure, and multiple labs. We were left with $1,398$ features, which was a much higher cardinality than the CURE-CKD dataset. When the data were target encoded into continuous features (see 4.1.1.4), there were $1,392$ features, and when the data were discretized into categorical features (see 4.1.1.4), there were $2,679$ features.

Missingness Profile This dataset was more scarce compared to the CURE-CKD dataset. The overall dataset was missing 31.16% of data. Since the data were aggregated over a single window of 7 days, we do not present an over-time missingness sketch. However, we can visualize how different features have various levels of missingness (Fig. 5.2).

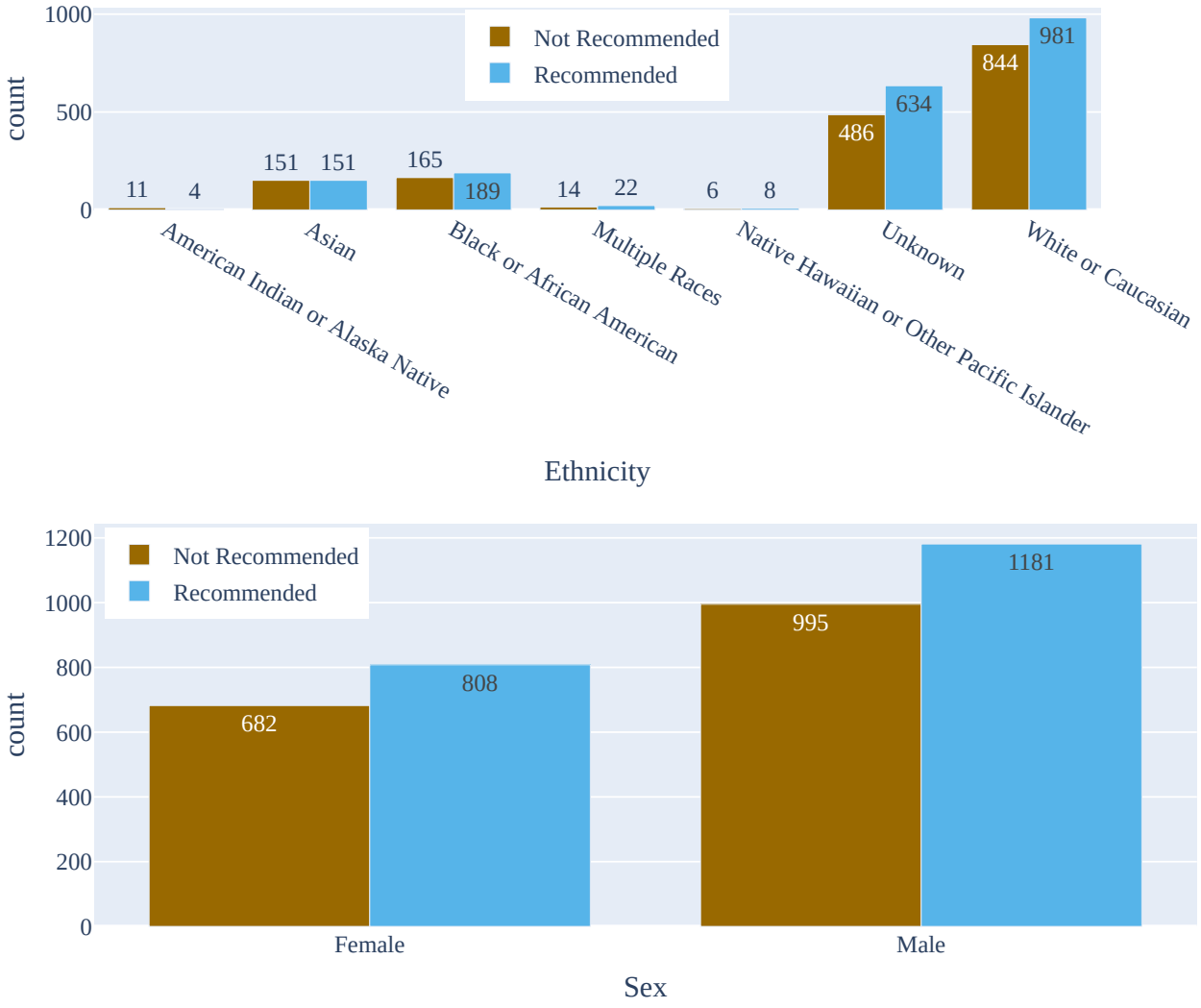

Figure 5.1: The counts of the number of CRRT patients at UCLA and whether we recommended CRRT broken down by ethnicity and sex.

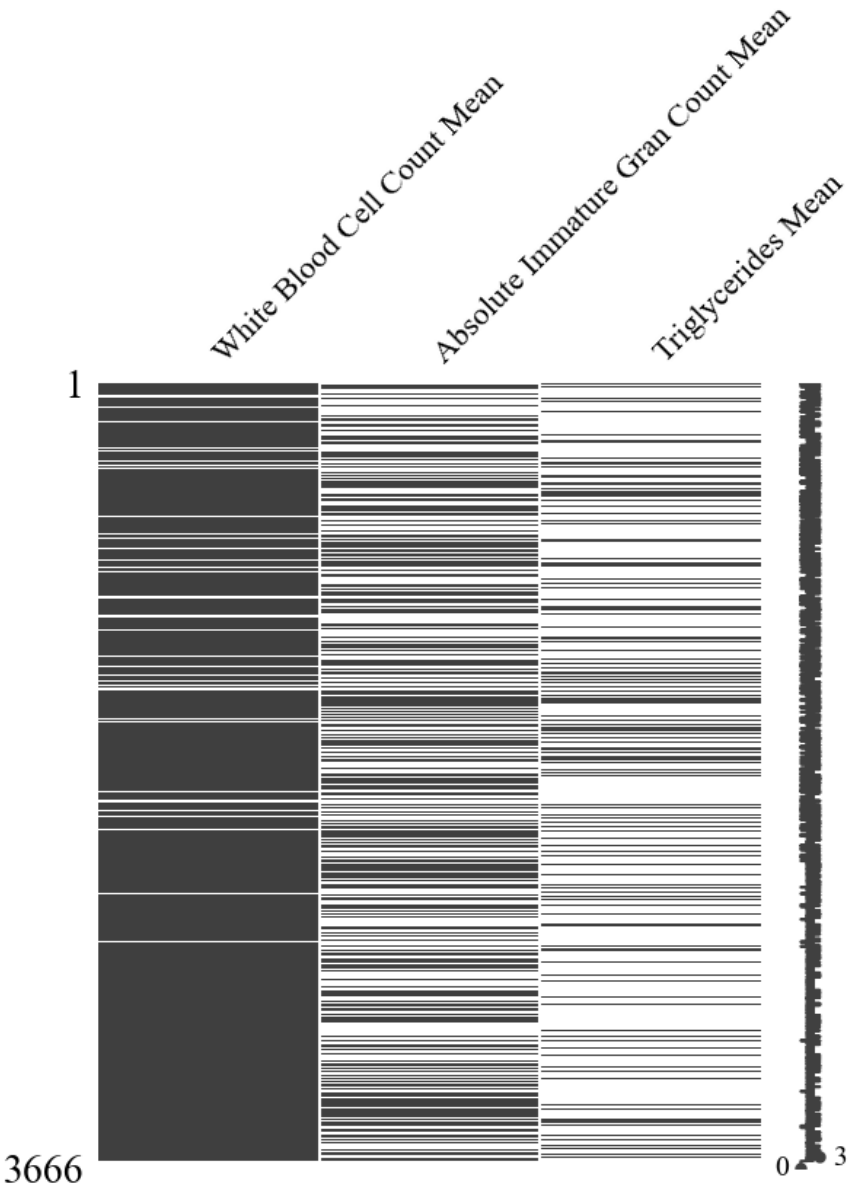

Figure 5.2: Missing values visualized in the CRRT dataset for three features with various missingness levels. The dark blocks represent observed values, while otherwise white or empty blocks represent missing values.

Cohort	Recover Renal Function	Transition to Hemodialysis	Comfort Care	Expired	Total
	Recommend CRRT		Do Not Recommend CRRT		
UCLA: CRRT	501 (13.67%)	1488 (40.59%)	1064 (29.02%)	613 (16.72%)	3666
	1989 (54.26%)		1677 (45.74%)		

Table 5.1: Breakdown of Outcomes for the UCLA: CRRT cohort. All numeric values are reported as N (%), their raw count and their percentage, aside from the reported totals.

5.2 Evaluation

5.2.1 Imputation Performance

We can only report the loss, number of epochs, and training speed on CRRT dataset as we did not have the true values.

Loss (Fig. 4.9) We report the last recorded validation loss for the autoencoder methods in the mapped space on the observed values only. The target encoded methods generally converged to the smallest loss, while the discretized methods usually struggled the most. The mixed feature performance was slightly worse than target encoded, but it was not obviously visible in the graph. The discretized performance was much worse here when there were four times as many features.

Training Convergence with Early Stopping (Fig. 4.10) The models training on data in the discretized feature space seemed to converge around the same epoch across the non-variational autoencoders. The variational-type autoencoders converged at a similar epoch to the autoencoders that did not use any feature mapping. However, the vanilla autoencoder

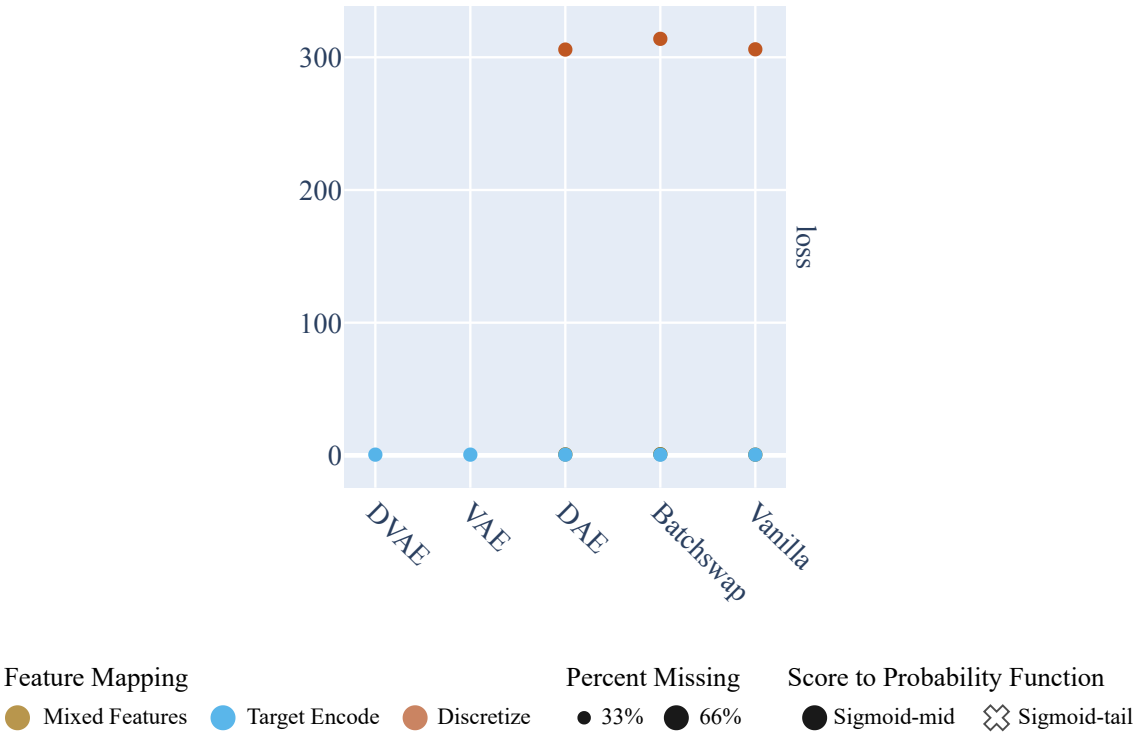

Figure 5.3: Last recorded validation loss in the mapped feature space on observed values only in the CRRT dataset. The x-axis represents each autoencoder imputation method; there are no baselines, as those are trained differently. The y-axis represents the metric value. See 4.4.3 for more.

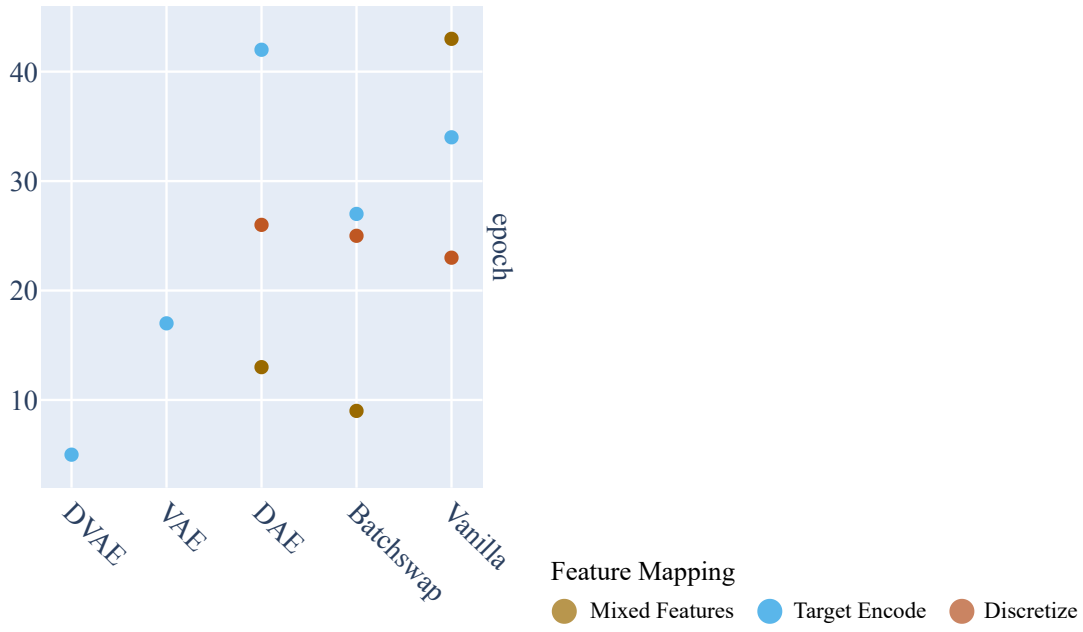

Figure 5.4: Number of final epochs for training convergence when training with a maximum of 100 epochs, and early stopping with patience of 2 epochs. Each column is a missingness mechanism. The x-axis represents each autoencoder imputation method; there are no baselines, as those are trained differently. The y-axis represents the metric value. See 4.4.3 for more.

converged the slowest in the mixed feature space. These patterns overall aligned with the CURE-CKD dataset patterns for convergence.

Training Speed (Fig. 5.5) Though the CRRT dataset had a higher cardinality than the CURE-CKD dataset, the mixed feature space and the target encoded feature space completed each epoch with similar timing to the CURE-CKD dataset. Here we also see that discretizing was the slowest, followed by target encoding, followed by no feature mapping. Though, the difference in epoch time for the discretized feature space was much more pronounced in this dataset than in the CURE-CKD dataset. The discretized data was almost four times as slow

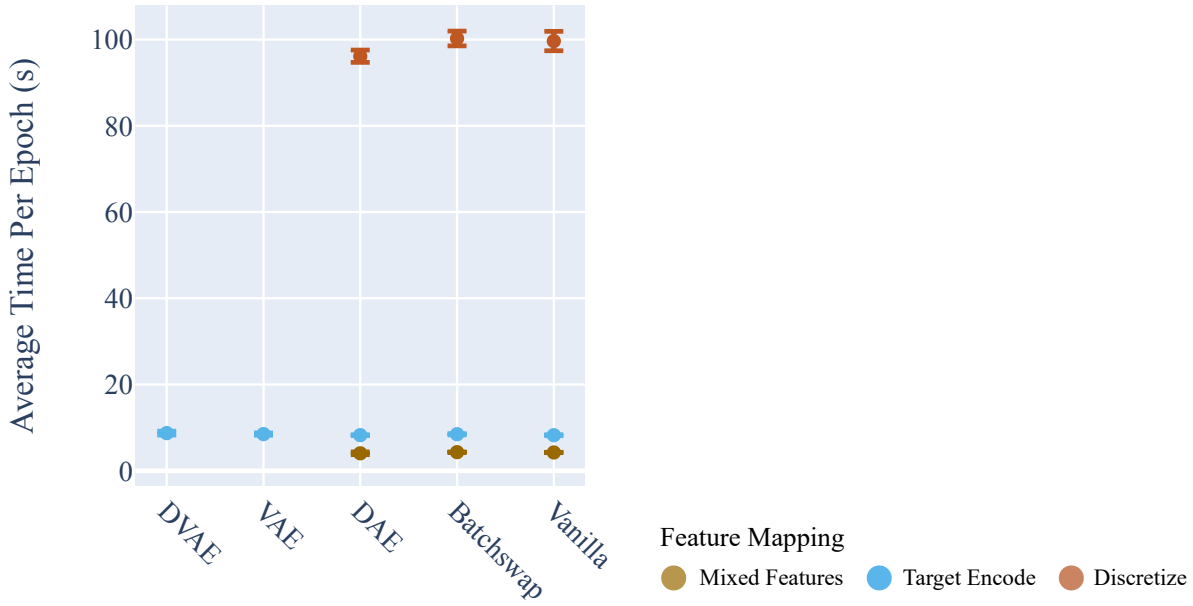

Figure 5.5: The average duration of each training epoch for the autoencoder imputer models on the CRRT dataset. The x-axis represents each autoencoder imputation method; there are no baselines, as those are trained differently. The y-axis represents the average duration of each epoch in seconds. See 4.4.3 for more.

per epoch, with eight times as many features.

5.2.2 Predictive Performance

Predictive performance (Fig. 5.6) proved to be different depending on the imputation method used. Refer to 4.4.3 for a deeper explanation of predictive performance legends in each figure.

Like in the CURE-CKD dataset, the LGBM tended to outperform the RF, and the baselines outperformed the autoencoder models. Particularly, the simple imputation method led to the best predictive performance for both ROC-AUC and PR-AUC. Overall, the performance on PR-AUC and ROC-AUC were both good and similar to each other, which was not the case for the CURE-CKD dataset. However, for CRRT, for the DAE, RF outperformed LGBM. Out of all autoencoder methods, the DAE could achieve the best ROC-AUC perfor-

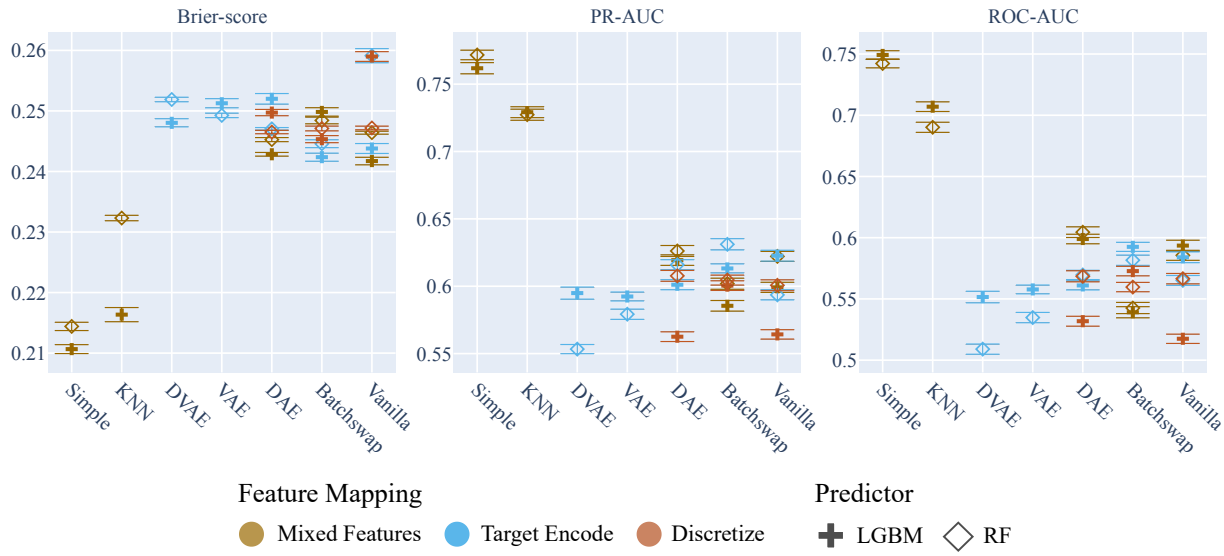

Figure 5.6: Predictive performance on the test split of the CRRT dataset. Each column is a different predictive metric. The x-axis represents each imputation method, including both baselines and autoencoders. The y-axis represents the metric value.

mance in mixed feature space, while the batchswap autoencoder model performed the best for PR-AUC. DAE and vanilla autoencoders discretizing the data performed poorly when predicting using an LGBM.

The LGBM was consistently more calibrated for all the baseline imputation methods, which was not the case for the CURE-CKD dataset. However, like in the CURE-CKD dataset, there was no consistent pattern otherwise across the autoencoder imputation methods. There was also no pattern between feature mapping and the Brier-score. The variational-flavor autoencoders tended to perform the worst, although the others did better when target encoding.

5.2.3 Model Selection

On the CRRT dataset, we aimed to predict if a patient would do well on CRRT and if we should therefore start that patient on CRRT. Similarly, we saw the best predictive perfor-

mance across all metrics when we simple-imputed the data and used an LGBM for prediction. Therefore, we would also select the same combination for the CRRT predictive task as for the CURE-CKD task. Though there was no imputation performance on this dataset, if we were to pick based off of the autoencoder learning ability, we would probably go with any of the methods but with target encoding.

5.2.4 A Deeper Dive into Prediction

Given that simple and KNN imputation led to the best predictive performance, and that we could not control the missingness to perform a more extensive analysis of imputation performance on the CRRT dataset, we present a deeper analysis of predictive performance. The data and training pipelines differed slightly as part of ongoing work on a wider set of experiments across all cohorts mentioned in 5.1. We present 95% confidence intervals, but computed them using percentiles rather than the normality assumption that produces symmetric bounds we used for the other analyses. We can better visualize asymmetries in these analyses, whereas before we were concerned with overall interval size, where we had more parameters for comparison.

Training Pipeline In these sets of analyses, we trained and validated our models on patients who were on CRRT for 7 days or fewer and evaluated on all patients. We produced the "best" model over the following hyperparameter grid. We selected the look-back window size, or the number of days before each patient's treatment start date from which to aggregate data, from the options 1, 2, 3, 4, 5, 6, 7, 10, and 14 days. We selected a model from the options: light gradient-boosting machine (LGBM), extreme gradient-boosted decision tree (XGB), random forest (RF), or logistic regression (LR). Note that each model type has its own (possibly different) hyperparameters that we tuned depending on which model was selected. We selected an imputation method between simple and KNN imputation. We also decided on a feature selection procedure based on the Pearson correlation coefficient

between each feature and the target variable by selecting the top features (high correlation) for a particular number of features (k Best) or by using a correlation threshold. We ran 50 trials of tuning by sampling from the above hyperparameter grid using the tree-structured parzen estimator (TPE) algorithm [BBB11]. We then selected the model with the best (highest) recorded Receiving Operator Characteristic - Area Under the Curve (ROC-AUC) on the validation dataset, and evaluated its performance on the testing data split. Note that since we tuned the model, we did not compare across different models, but rather more deeply explored the best one. We found that the best model was the XGB model, on a 5-day window, using simple imputation, and correlation threshold of 0.08. This largely aligned with our setup from our previous analyses on predictive performance, with gradient-boosting algorithms and simple imputation producing the best outcomes, though the other parameters either did not exist (such as feature selection) or slightly differed (such as 5 instead of 7 day window) due to differences in the training pipelines.

5.2.4.1 Performance by Subgroup (Fig. 5.7)

We evaluated our model more deeply on subgroups of the dataset based on different characteristics. One set of characteristics was a patient's medical indication for requiring CRRT, based on the most common conditions that may lead a patent to be hemodynamically unstable and therefore require CRRT. We classified patients having indications of heart issues, liver issues, infection, and none of the above via ICD code diagnoses. Note that these groups were not mutually exclusive, for example, a patient may have had a heart condition and was also suffering from a severe infection. Other characteristics were based on sex, race, and age groups.

Among the metrics, the PR-AUC is the least stable, with the widest confidence intervals across all subgroups. Among the patient indicators, we are able to predict slightly better on the infection, liver, and heart patients compared to patients who did not have any indications for all metrics. The models are more calibrated for those groups, and more accurate under

Figure 5.7: Predictive performance on the CRRT dataset by subgroup. Each column is a different predictive metric. The x-axis represents the average metric value. The y-axis represents the category for each subgroup. The vertical dotted line is the average metric value for all patients, and the gray rectangle is the 95% confidence interval for all patients.

both PR-AUC and ROC-AUC. There is almost no difference between male and female patients, although the model is slightly better calibrated for female patients. We see multiple artifacts under the subgroup concerning race. Particularly, there were very few Americans Indian or Alaska Natives and we did not have any positive samples and therefore our model struggled on those patients and could not evaluate metrics on them. Performance on Native Hawaiian or Other Pacific Islander or Multiple Race patients was poorer than the average had the widest confidence intervals, meaning our model was the least confident on them. This can also be heavily attributed to the very small subsample size (Fig. 5.1) for those subgroups. The model performed slightly better on Black or African American patients, as well as Asian patients, compared to the overall average. Regarding ethnicity, the model performed slightly worse on Hispanic or Latino patients compared to Non-Hispanic or Latino patients. The model performed increasingly better on all metrics as the patients got older, with it being least calibrated and accurate on the youngest age range of patients. While the graph labels the range as 20 to 30 it was actually 21 to 30. The confidence intervals presented more of a bell-curve shape, being the widest at the extremes, and the tightest towards the middle-aged patients.

5.2.4.2 Rolling Window Performance (Fig. 5.8)

Our model could only make a prediction on data from a patient *before* they may or may not have started CRRT, otherwise we would be leaking information that we would not have for new and unobserved samples. During that time, patients were under active care and their condition may have fluctuated due to their own biological processes or because of direct medical intervention. A patient's outcome may have stemmed from events that occurred *after* they began treatment, meaning we would not be able to capture that data for new patients, and therefore we could not train our model on that data. However, we could *evaluate* our model on future data.

We implemented a *rolling window analysis*, evaluating the model (without retraining) on

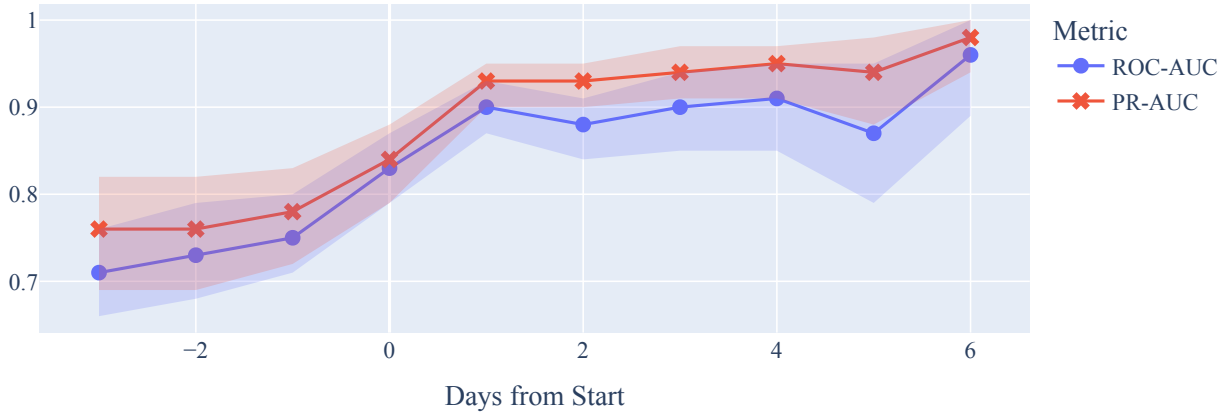

Figure 5.8: Predictive performance on the CRRT dataset by rolling a 5-day window forward and backward from the treatment start date of each patient. The x-axis represents the number of days the window was displaced from the treatment start. The y-axis represents the metric value. The filled in sections represent the 95% confidence intervals.

the same sized window some number of days displaced from the start date, starting from 3 days *before* to 6 days *after* the start of treatment. Each CRRT patient was on CRRT for a disparate number of days, so we would be predicting on an outcome horizon 4 days out or even 14 days out from the start of treatment. Therefore, we limited the evaluation to patients who were on CRRT for a maximum of 7 days to match the maximum number of days we would slide forward in this analysis to get as close to each patient's outcome horizon as possible. This way, we were not predicting potentially a week out from one patient's outcome but 1 day before the other, where the former is much more likely to be less accurate. By monitoring metrics over each window, we could analyze how the predictions and the performance of the model changed as the data neared the outcome horizon. Do note that this was not a dynamic model, as this procedure was completely unaware of previous days or how measurements change overtime, which would be used in an ongoing day-by-day predictive task.

We observed a change in performance, and inferred that the model's predictions did

change for some patients on later days, which we could reason implied that their outcomes were influenced by events after they started CRRT. The performance of the model improved each day as it was evaluated closer and closer to the outcome horizon without retraining, indicating that the model learned meaningfully. We see the steepest decline in performance just one day before starting CRRT, and the steepest increase in performance just one day after start CRRT. While the performance overall gradually continued to trend in the same direction, the day-to-day performance does not change as much. There was a slight dip in performance at day 5, but otherwise no breaks in the positive trend of performance for both metrics. The PR-AUC was always higher than the ROC-AUC. The 95% confidence interval for the PR-AUC narrowed by day 0 and then slowly widened as it approached the outcome horizon. Similarly, the confidence intervals for the ROC-AUC narrowed by day 0 and then widened as the model approached data at the outcome horizon, however, for ROC-AUC, the model became much more uncertain than for PR-AUC. This may be connected to how the performance of ROC-AUC dropped more dramatically at day 5 than PR-AUC.

5.3 CURE-CKD vs CRRT: Building a Profile

There are noticeable differences in imputation and predictive performance depending on the imputation method, the feature mapping, and the missingness scenario. Although the baseline methods tend to see worse imputation accuracy, they lead to better predictive performance. This might be because a simpler estimation smooths out the data in a way that makes it easier for the predictor model to delineate between the two classes. Inversely, the autoencoder methods tend to achieve better imputation accuracy but lead to worse predictive performance. We believe a deeper dive into comparing different deep learning regularization techniques would expand more on this hypothesis. While overall consistent for the autoencoders, the different metrics produced slightly different imputation performance rankings for the baseline models. We believe some of these differences can be attributed to

the difference in possible output ranges between RMSE, which can be potentially infinite, and MAAPE, which caps at 1.0. We expected but did not observe a difference in CW versus EW metrics, which may be because our amputation procedure is quite simple compared to the space of missingness possibilities —although it is complex compared to what is being used in the literature.

In the original space, the vanilla autoencoder seems to perform the best. However, in mapped space, the mapped methods achieve better imputation accuracy. While under target encoding, we would see a reduction in cardinality, i.e., for one-hot encoded multicategorical features which would be condensed into a single continuous feature, which might explain the better performance. However, we also find better performance when discretizing, which would instead grow the cardinality of the data since it would take a single continuous feature and expand it into two or more one-hot encoded categorical ones. Poorer imputation performance when feature mapping the data in the original space is likely due to the loss of information when inverting the feature mappings, as none of the mappings is one-to-one and cannot be directly and precisely reconstructed. This trend is not as clear in the predictive performance on the CURE-CKD dataset, but is consistent in the predictive performance on the CRRT dataset. In the future, we would also experiment with training and evaluating the predictive model on the mapped data. The imputation performance patterns for the autoencoder methods are not reflected by the validation loss. We believe the discretized methods have a more unstable loss trajectory due to their high cardinality. Therefore, they tend to be stopped earlier and achieve worse loss even though their predictive performance is better.

The trends of imputation performance across the missingness mechanisms vary between the other characteristics of missingness and the imputation models, with the trend sometimes reversing under certain mechanisms. From the confidence intervals (Table 4.5), we can account for some of this variability due to the size of the confidence interval. However, sometimes the difference in performance is much wider than what we would expect to

be reasonable variation, in which case we would explore further in future work. For each missingness mechanism, we achieve the best imputation performance when there are fewer missing data, the score to probability function is sigmoid-mid, and there is no feature mapping. The best performing mechanisms in this context are MAR and MNAR(Y), while the worst are MCAR and MNAR. Though the differences are minimal, this validates the idea that non-recoverable MNAR can be detrimental to imputation accuracy.

Although the gap in imputation performance across missingness patterns is slight in the CURE-CKD dataset, the variation in performance in addition to relatively narrow confidence intervals suggest that these different mechanisms do behave differently. In fact, there are sometimes differences in predictive performance depending on if average or extreme values are more likely to be missing from the score to probability function within the same mechanism. These findings lead us to believe that not only are the three traditional mechanisms potentially not granular enough, but rather a wider, more robust paradigm that accounts for missingness linked to distribution tendency (e.g., extreme values) might be more useful. There are potentially more dimensions or characteristics that would help better characterize and understand missingness behavior in data, however, these factors serve as a launching point to explore further.

The predictive performance rankings are inconsistent across datasets, highlighting that there is no one method that performs better than all the others for all situations. In general, the LGBM tends to perform better and be more robust. Strangely, while the imputation performance is the poorest and least confident under MNAR, it leads to the best predictive performance. It may be that the samples with data MNAR are more distinct and easier to delineate, although their exact values are harder to recover. The ROC-AUC is higher for the CURE-CKD than CRRT. This is likely due to how imbalanced the CURE-CKD outcome is compared to the very balanced CRRT outcome, and therefore the models can inflate their ROC-AUC on the negative samples. We saw in Chapter 3 that the Precision-Recall - Area Under the Curve (PR-AUC) on the CURE-CKD dataset is quite poor.

Our results are somewhat consistent with Jäger et al. who report a wide comparison between multiple machine learning imputation methods and simpler baseline ones [JAB21]. Like we see in our analysis, simple imputation method tends to not perform as well for imputation accuracy as the other deep learning models, aside from one, on their benchmark datasets. In contrast to our work, Jäger et al. observe that KNN tends to perform better compared to other methods. Both of our works observe that machine learning imputation could sometimes lead to worse downstream predictive performance compared to their simpler alternatives. We believe there is legitimate divergence between faithful data recreation and producing data on which prediction is easier. Ultimately, both our work and Jäger's work highlight variability in imputation and downstream predictive performance across tasks and imputation methods.

CHAPTER 6

Conclusion

The adoption of EHR has made patient data increasingly accessible, precipitating the development of machine learning models to help physicians. ML-based imputation methods have shown promise in various domains for the task of estimating values and reducing uncertainty to the point that a predictive model can be employed. To address missing data, researchers have been developing, analyzing, and comparing statistical and machine learning techniques for missing data estimation or imputation. In this context, we first built an early version of our original framework, Autopopulus. We aimed to tune imputation for each dataset and task, create a profile by comparing imputation methods widely on controlled missingness scenarios, and enable other researchers to easily tune imputation and select a model. Based on what we learned from our first attempt in Chapter 3, we expanded Autopopulus to push beyond the current understanding of autoencoder-led imputation in a deeper, more meaningful way. We showed in Chapters 4 and 5 how we tuned imputation for the respective predictive tasks on the CURE-CKD dataset (predicting rapid decline), and the CRRT dataset (recommending a patient start CRRT). We ran a suite of experiments to build a profile of autoencoders on the CURE-CKD dataset, which had a fully observed subset. We also ran another suite of experiments to show the Autopopulus workflow for model comparison and selection on the CRRT dataset. Based on these outcomes, we built a profile of autoencoders. Finally, we will outline lessons learned during this process, mistakes to avoid, and how this work may be expanded upon in the future.

6.1 Reframing Imputation as a Task

For many, imputation is seen as a data transformation step completely separate but necessary for their main task, be it clustering, prediction, or any other possible machine-learning task. Typically, data transformation and data wrangling is considered a side note to a larger, more attractive spectacle, which is the main ML task. However, we believe imputation may be more of a "predictive" task than we regard it to be. We typically regard predictive tasks for machine learning to be predicting a continuous or categorical target, or maybe even multi-label targets. Imputation is quite similar: predicting many continuous or categorical targets at once. The main caveat being that we typically do not have any labels or true values for the missing values. Imputation is a bit like predicting in the dark. It is certainly a less than ideal problem, but a reality one that most practitioners have to solve. We believe there should be a shift in attitude regarding imputation; rather than thinking that you must do something to the data so that you can do the "more interesting" task, you are predicting on the data now so you can predict on the data later. That shift in attitude and perspective might incline more practitioners to regard imputation with more care, and actually tune the methods to their needs.

6.2 Occam's Razor: Imputation

It may be ironic for a dissertation dedicated to the use and analysis of deep learning imputation methods to conclude with the remark that other, simpler methods may be more desirable for downstream tasks. Had we only analyzed imputation accuracy, we would not have come to this conclusion. We repeatedly saw, not only in our work, but in other work, that simpler imputation methods tended to perform better for predictive tasks even if they were not as accurate at recreating the missing data. We believe that more accurate data imputation is not necessary for more accurate downstream predictive performance. In other words, the imputers that are better at recovering reality are not the same ones that can facil-

itate better predictive performance. There may be information in the more uniform, but less accurate, imputed values from simpler imputers. In a way, they may behave as an indicator beyond the values themselves. Perhaps encoding values that are decently accurate but in a part of the data manifold that might communicate that they both have a quantitative value and a qualitative latent quality, which is that they were originally missing.

As with any set of tools, the standard is not that one is always better than the rest. Within our current understanding of missingness, it seems as though empirically, simpler methods tend to lend to better predictive performance. We believe that while we should continue to explore autoencoders for imputation, in an applied context, we would recommend reaching for simpler imputation methods first before expanding to more complex ones. More importantly, that imputation exists in larger contexts and should be considered for their behavior in each setting. We have room to learn more about autoencoders and other deep learning methods for imputation, and in which circumstances more exactly are they less useful than simpler imputation methods.

6.3 The Taxonomy of Missingness

In conjunction with the work of Mohan et al., our extensive experimentation using the more fine-grained missingness scenarios calls into question the current taxonomy we have for reasoning about missingness. Researchers have been relying on understanding and dealing with missingness using the three mechanisms of missingness of Rubin's work since 1976. While it certainly is a powerful tool and has allowed researchers like us to tackle the problem of missing data, we are increasingly aware of its limitations. It may not be difficult to believe that data can be missing in forms and patterns that themselves require a whole body of characterization, the way we do for observed distributions of data. When we reason about observed data in order to handle them, the causal characteristics are not always impactful. We have many other tools to distinguish different types of data: such as continuous and

categorical. As of now, the only factors available to describe missingness rely solely on the causality of the missingness.

Just the same, this idea itself is too broad and difficult to transform into actionable statements. With our work, we believe we may have found a few of the missing puzzle pieces. The trends that manifest in missingness are important as well, such as the difference between extreme or average values being more likely to be missing. There may not be a Normal distribution equivalent in missing data, however. Missing data are unique in that they live in both a discrete and continuous world. There is discrete qualities to them: whether missingness exists or not, and where it exists. But there is also quantitative qualities to them: the quantity of missing data, and the tie to a latent value that itself could be either categorical or continuous. Missing data is challenging and unique in this way compared to observed data.

Researchers comparatively have a very strong grasp of qualities of observed data themselves. However, there is a large gap when in comes to missing data. We believe that this gap is an opportunity for progress, and that development in this area will provide a strong benefit to all the sciences that rely on data. A more robust paradigm for reasoning about missingness can enable not only directed iterations of existing methods, but also the creation of new imputation methods. These methods lead to both more accurate imputation methods and provide values that can be used for predictive tasks more easily.

6.4 Technical Takeaways and Pitfalls

In creating and maintaining such a large and extensible framework, through failure and in turn through success, we have become familiar with common pitfalls in implementing and working with machine learning and particularly deep learning code.

6.4.1 Computational Resource Management

One particular challenge was with compute resource management and data input/output between hardware components. While having more GPUs available can allow for faster computation, the dataset needs to be large enough to warrant the overhead of setting up the data on each one and synchronizing the results between each one. While it is inevitable that the data must be passed between the two components upon transferring to and from the central processing unit (CPU), doing so more than once is costly. Taking care to minimize the amount of data passed between the CPU and GPU on the machine can vastly improve training time. This includes not only the data passed to the model but also any auxiliary data as well. For example, one challenge was ensuring certain computations were executed on specific columns, such as continuous ones only. In order for this to be done on the GPU, we needed the auxiliary data of which columns were continuous to be on the GPU as well. Other challenges included the feature map inversions. While they were not differentiable, most of the mappings were trained on the CPU when preparing the data. Transferring the imputed data to the CPU to compute the feature map inversions, and then additionally computing the metrics on the CPU proved to be slow. A workaround we used for this was auxiliary data from the mappings themselves, and recreating the computational abilities for those data transformations with the more limited set of GPU instructions. While not a straightforward endeavor, the time saved during training was multiplicative.

6.4.2 Validating and Testing a Data Pipeline

How to validate and test complex data pipeline and machine learning models is not straightforward. Aside from lessons learned shared from other researchers attempting a similar feat such as Tilman et al.'s work, guidance is sparse [Kro21]. Testing requires isolated functionality and integration tests. When isolating, you might mock data and/or functions.

The Challenge of Mocking Data Mocking data has many more degrees of freedom compared to regular programming objects with limited fields. Data can have any number of features, any order of features, any combination of types of features, any number of samples, and so on. Each of those features can have any range of values: missing values, integers, large float-point numbers, negative scalars, and so on. Mocking data can prove to be a delicate balance between creating data complex enough to match the functionality or behavior you would expect to see on real world data, but also data simple enough to be able to know the expected behavior ahead of time.

When mocking data and writing tests, start simple, and ensure that the mocked values are also tested to be in the format you expect. For example, ensure the function was called with the arguments you expect, or the data indices are the same, or the columns are formatted the same. Importantly, do not adjust the data pipeline code or test code to that specific simple example. For example, you may end up accidentally encoding some expected column order or column name format that does not exist in the real data. Similarly, your test should not pass because some auxiliary piece of information was convenient to the test, but fails on real data. We found that packages like `hypothesis` help to catch edge case examples (i.e. NaNs, infinities, etc.) [MHm19].

While tests are meant to explicitly encode assumptions, usage, and intended behavior of the code, it may help to also insert breaking lines of code such as assertions from Python into the main body of logic. This is particularly helpful for layer-style pipelines.

Layer-Style Pipelines Many frameworks such as `pandas` or `sklearn` offer functionality for and encourage the use of chained and/or layered applications of operations. While this approach makes the code more readable and leads to fewer errors, it can be harder to debug depending on how the operations are applied. For example, `sklearn` pipelines are trained and applied with `fit()` and `transform()`, however, if there is an error there is sometimes no direct way to know which step failed. For example, at a certain point of a multistep pipeline,

you might expect there to be no missing values. If the pipeline breaks because of that, it will simply fail as a whole, and it will be clear which step failed. It might be difficult to test the output of an incomplete pipeline, but you can include an assertion in the pipeline. This also helps make debugging chained or layer-style pipelines easier, as the assertion will fail directly at the step where it is expected.

6.4.3 Pitfalls with Masks

When working with missing data in particular, data masks will be used throughout the project. Surprisingly, working with masks can be rather tricky, and how and where you create and name a mask matters. Create and use masks right where you plan to use them. Passing the same mask through and adjusting based on your needs—perhaps in one place you need the inverse mask—is an easy way to confuse what the purpose of the mask is or accidentally mutate it. It is not always obvious if the mask that has been applied is inverse of what you intended to use. Naming masks properly will also help avoid this issue. Some masks *keep* values where the indicator is 1, while others will *drop* values at those locations. Rather than simply naming masks `*_mask`, label them by what they indicate. For example, use `where_fully_observed` instead of `fully_observed_mask`. The latter does not communicate if the mask will be used for ignoring and keeping fully observed data. Similar to the pitfall revolving around masks, it is advisable to use filtered or altered versions of data right where you need them, unless that version of the data is the only version of the data you will use for the rest of the pipeline. If you need both, carefully name and separate the two versions so that it is clear.

6.4.4 Coordinating and Tracking Many Experiments

When coordinating a large set of experiments, it is important you:

- organize and design all the experiments properly.

- test the *whole* process from start to finish, including initial visualization and analysis.

In order to move quickly, it helps to do the initial runs on a small subset of data, with any variations cut down to a minimum. For example, limiting the number of training epochs, or tuning samples. Simple visualization is not enough to validate results, as corrupt results can be obscured by other reasonable ones. Attempt some form of result validation, such as whether it is acceptable if all your errors are 0 or 1.

6.4.5 Be a Part of Open Source

A lot of open source frameworks are either new or recently well-established. Over the course of this work, many frameworks vastly changed in their functionality and API. We recommend you be a part of the community, recommend or implement new features, or make existing ones more robust. Sometimes you may need to implement your own version of functionality that you can either submit as a feature, or find that a more polished version has been made available in the latest package release. Being involved, and keeping up with new developments, vastly improved our framework. In fact, some crucial documentation was not made available until later after users asked questions. For example, the specific circumstances under which all metrics, data, and auxiliary data would be placed on the correct device was not made explicit or clear until more recently for `pytorch-lightning`. It was because we checked in later that we found the answers to some mysteries surrounding slow performance in our framework.

6.5 Future Work

To our knowledge, autoencoders are beginning to be used for multivariate longitudinal imputation, but have not been profiled at all as imputation models. Longitudinal amputation procedures do not exist, and our current experimental pipeline does not account for datasets

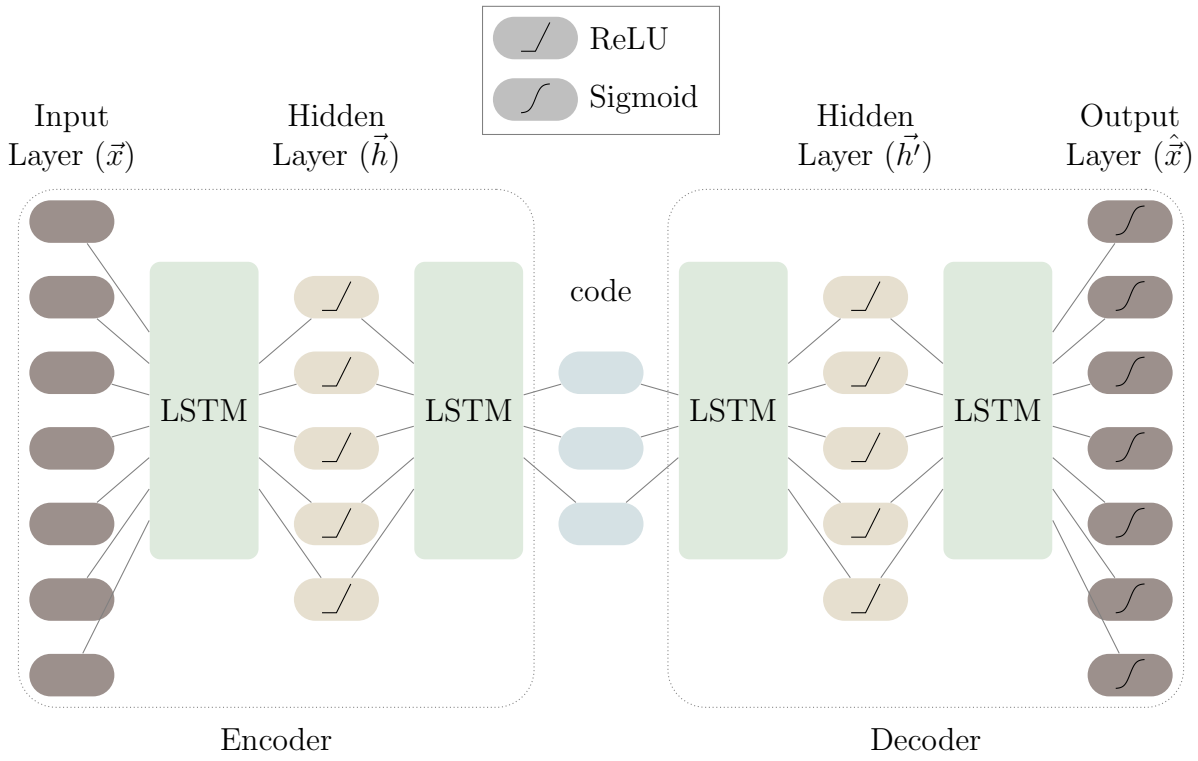

Figure 6.1: An example of an undercomplete autoencoder architecture that can handle longitudinal data of T time points and D features by swapping fully connected layers with LSTM networks.

missing data according to a mixture of missingness mechanisms.

Experimental Results for Longitudinal Imputation For longitudinal imputation we can combine autoencoders with LSTMs, such that, instead of fully connected layers as in Fig. 2.1, each layer is an LSTM which is updated in Fig. 6.1.

We chose LSTM layers due to their ability to handle multivariate longitudinal data of unequal or variable sequence lengths. We chose LSTMs over a more general RNN due as they do not suffer from the exploding/vanishing gradient problem, and they can easily be extended to be bidirectional which could leverage data moving forward and looking backward towards imputation, which could be particularly useful for missing longitudinal data.

To handle sequences of unequal or variable lengths, we pad the data to the maximum sequence length per batch. Not pictured in Fig. 6.1 is the cell and hidden states that each LSTM network outputs. Within the encoder or within the decoder, the output cell and hidden states of each LSTM layer are fed forward as the input cell and hidden states to the following LSTM layer. The states are not shared across the encoder and decoder. It is possible to use a stacked LSTM instead of multiple LSTM layers, however, with the current available tools it becomes more difficult to control each subsequent hidden layer size which in turn makes the bottle-necking shape of an undercomplete autoencoder harder to achieve. Additionally, the cell and hidden states will refresh, or be reinitialized, for each subsequent layer instead of passing them forward across layers. For this reason, we believe it is more useful to separate layers as proposed.

While this is currently implemented in Autopopulus, it has not been used for experiments. We will run experiments for both static and longitudinal imputation on both the CURE-CKD and CRRT dataset for validation and using the `pyampute` package for amputation. Until longitudinal amputation is fully fleshed out, we will follow a simple amputation procedure that amputes per time point. Analogous to the experiments we have run for static imputation with autoencoders, we will complete running all the experiments for longitudinal imputation.

This will explode our experiments in the following way:

$$\text{Time} \times \substack{\text{Loss: All/}\\\text{Observed}} \times \left(\substack{\text{Imputation}\\\text{Model Type}} \times \left(\substack{\text{Missingness}\\\text{Mechanism}} \times \substack{\text{Missingness}\\\text{Percent}} + \substack{\text{All Data}\\\text{No Ampute}}\right) + \substack{\text{No Impute}\\\text{Fully Observed}}\right) \times \substack{\text{Classifier}\\\text{Model Type}}$$

A challenging aspect of the data is that we need comparative models that can deal with multivariate longitudinal data with unequal sequence lengths. Due to the complex nature of the problem, there are less non-deep learning methods that can meet this requirement for both imputation and classification. Our decisions for the imputation model types are formed from a combination of autoencoder types and baseline comparisons {simple, KNN, BRITS [CWL18], SAITS [DCL22]}. The mechanisms are {MCAR,MAR,MNAR}. The percentages are {33%, 66%}. Finally, the static classification models are {KNN, shapelet time series

classifier}.

Longitudinal Amputation Previously, we discussed the importance and utility of amputation (4.2). With `pyampute` we are able to achieve multivariate amputation, however, it is currently only possible with static data. Currently, there is no known paradigm to describe missingness in the domain of longitudinal data. In fact, a large body of the literature for longitudinal imputation either ignores evaluating their methods across different missingness mechanisms or proposes a simplistic approach to amputation that ignores the time dimension. The first step to ampute longitudinal data is to extend the traditional missingness mechanism paradigm to a longitudinal setting. Since time introduces a new dimension, we much be able to model a relationship of missingness in the new time dimension. One approach would be to nest the mechanisms into the time dimension itself, such that for a particular variable:

- MCAR would imply that missingness for any time-point is completely unrelated to any other time-point.
- MAR would imply that missingness for a time-point is explained by some combination of other time-points.
- MNAR would imply that missingness for a time-point is missing due to the value itself, or due to some unobserved time-point or measurement.

We could then apply a similar methodology that ensures joint missingness probabilities according to these mechanisms across sequences of data.

Another challenge we aim to address is generating useful amputation schemes. While we currently hand-pick combinations of features for the given experiments, a more regimented scheme for amputation scenarios may be aid in being able to uncover data characteristics that are useful for imputation model selection. For instance, we may decide to ampute based

on covariance or correlation, variability of feature values across samples, or we may decide to incorporate and compare different mixtures of missingness mechanisms (e.g., MAR and MNAR vs. MAR and MCAR). These schemes would behave analogous to automatic and systematic ablation experiments to help us generate imputation profiles.

Imputing by Mechanism Missingness mechanisms in real-world datasets are never clean-cut: there may be multiple mechanisms at play across different combinations of variables. Instead of training a single network to impute a single dataset, we can take the approach of creating a singular imputation expert per mechanism. That is, an expert on MCAR, an expert on MAR, and an expert on MNAR. If we can learn to impute by mechanism, we can then transfer their expertise to a single student that attempts to impute a dataset that may consist of missing data caused by various mechanisms. Each teacher, or expert, model would be trained on a stream of data consistently amputed according to a particular mechanism. The variables involved and the amount missing may vary, but the mechanism would remain the same. After training the experts, we would train the student on a separate stream of data that is amputed using multiple missingness mechanisms in a multi-teacher single-student transfer learning schema. A potential drawback of this approach is it is computationally expensive. One workaround would be to pre-train the models on a large dataset, and then allow an additional projection layer to project any dataset dimensionality onto the space of the pre-trained expert models.

Predicting in Feature-Mapped Space As seen in Sec. 4.4, inverting a feature mapping lost information and affected imputation performance. In order to take advantage of the cohesive feature space, a natural subsequent analysis is to analyze a predictive pipeline that is also in the mapped feature space. There are trade-offs to this approach, namely, that some problems require examination or results directly based on the inputs as-is. In such a case, this analysis would not be desirable. However, there are benefits to predicting in a mapped feature space, since the feature mappings may act as feature engineering and make

classification easier. For example, the continuous age value for patients may not be as strong of a predictor compared to bins of certain age ranges.

Autoencoder Artifacts Compared to other deep learning techniques, autoencoders have a unique property in that they leave behind a special artifact: the code, or latent representation of the data. We can then further investigate how using the code compares to using the imputed data itself for downstream tasks. In the fully-observed case, we expect that the code contains a more succinct, compressed representation of the data that represents useful extracted features from the more granular and possibly noisy original input space. However, this has not been explored for data with missing values and the difference between the behavior of the code itself versus the imputed data itself is not apparent.

Advanced Patterns of Missingness Our experiments served as a starting point for understanding imputation under complex amputation patterns. We did not explore mixing mechanisms of missingness within the same sample subsets when amputing data in our experiments. An important next step in this line of analyses is to explore further into more advanced patterns of missingness, since real-world data are much more complex than a single mechanism at a time. A principled comparison of the mechanisms would try different combinations of mechanisms, for example, MCAR combined MAR, or MCAR combined with MNAR(Y). Other dimensions of comparison would dive into the number of features influencing the missingness of another under MAR.

Warm Start Imputation When we first implemented and experimented with Autopopulus we followed work we had seen in the literature in using a per-batch simple imputation procedure as warm start for imputation. We found that the performance of the autoencoder got heavily entangled with the performance of the warm start imputation. When we later used Autopopulus on the CURE-CKD and CRRT datasets, we used a fill-0 procedure as to isolate the performance and behavior of the autoencoder itself for imputation. However, the

same way the different initializations of weights can greatly assist model training, it may be a more intelligent warm start procedure may provide a substantial boost to autoencoder performance.

REFERENCES

[AAL20] Haleh Akrami, Sergul Aydore, Richard M. Leahy, and Anand A. Joshi. "Robust Variational Autoencoder for Tabular Data with Beta Divergence.", June 2020. arXiv:2006.08204 [cs, eess, stat].

[AMF21] Michael S. Anger, Claudy Mullon, Linda H. Ficociello, David Thompson, Michael A. Kraus, Pete Newcomb, and Robert J. Kossmann. "Meeting the Demand for Renal Replacement Therapy during the COVID-19 Pandemic: A Manufacturer's Perspective." *Kidney360*, **2**(2):350–354, February 2021.

[Ath11] Jim Atherton. "Development of the Electronic Health Record." *AMA Journal of Ethics*, **13**(3):186–189, March 2011.

[BBB11] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. "Algorithms for Hyper-Parameter Optimization." In *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011.

[BM16] Brett K. Beaulieu-jones and Jason H. Moore. "Missing Data Imputation In The Electronic Health Record Using Deeply Learned Autoencoders." *Pacific Symposium on Biocomputing. Pacific Symposium on Biocomputing*, **22**:207–218, 2016.

[BVM19] Guillem Boquet, Jose Lopez Vicario, Antoni Morell, and Javier Serrano. "Missing Data in Traffic Estimation: A Variational Autoencoder Imputation Method." In *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2882–2886, 05 2019.

[Cha13] "Chapter 2: Definition, Identification, and Prediction of CKD Progression." *Kidney International Supplements*, **3**(1):63–72, January 2013.

[CHS19] Ramiro D. Camino, Christian A. Hammerschmidt, and Radu State. "Improving Missing Data Imputation with Deep Generative Models." 02 2019.

[CKH18] Enrico Coiera, Baki Kocaballi, John Halamka, and Liliana Laranjo. "The Digital Scribe." *NPJ Digital Medicine*, **1**:58, October 2018.

[Cla12] "Classification Systems for Acute Kidney Injury: Background, RIFLE Classification, Acute Kidney Injury Network." **2**, 03 2012.

[CPC18] Zhengping Che, Sanjay Purushotham, Kyunghyun Cho, David Sontag, and Yan Liu. "Recurrent Neural Networks for Multivariate Time Series with Missing Values." *Scientific Reports*, **8**(1):6085, April 2018.

[CS19] Junjie Chen and Xinghua Shi. "Sparse Convolutional Denoising Autoencoders for Genotype Imputation." *Genes*, **10**(9):652, 08 2019.

[CWL18] Wei Cao, Dong Wang, Jian Li, Hao Zhou, Lei Li, and Yitan Li. "BRITS: Bidirectional Recurrent Imputation for Time Series." In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[DCE13] Janez Demšar, Tomaž Curk, Aleš Erjavec, Črt Gorup, Tomaž Hočevar, Mitar Milutinovič, Martin Možina, Matija Polajnar, Marko Toplak, Anže Starič, Miha Štajdohar, Lan Umek, Lan Žagar, Jure Žbontar, Marinka Žitnik, and Blaž Zupan. "Orange: Data Mining Toolbox in Python." *Journal of Machine Learning Research*, **14**:2349–2353, 2013.

[DCL22] Wenjie Du, David Côté, and Yan Liu. "SAITS: Self-Attention-based Imputation for Time Series." *arXiv:2202.08516 [cs]*, February 2022.

[Doe] "Does Hospital EHR Adoption Actually Improve Data Sharing?" https://www.definitivehc.com/blog/hospital-ehr-adoption.

[Du23] Wenjie Du. "PyPOTS: A Python Toolbox for Data Mining on Partially-Observed Time Series." 2023.

[End10] Craig K. Enders. *Applied Missing Data Analysis*. Methodology in the Social Sciences. Guilford Press, New York, 2010.

[Fa19] WA Falcon and .al. "PyTorch Lightning." *GitHub. Note: https://github.com/PyTorchLightning/pytorch-lightning*, **3**, 2019.

[FHB17] Mallorie H. Fiero, Chiu-Hsieh Hsu, and Melanie L. Bell. "A Pattern-Mixture Model Approach for Handling Missing Continuous Outcome Data in Longitudinal Cluster Randomized Trials." *Statistics in medicine*, **36**(26):4094–4105, 2017.

[FI93] U. Fayyad and K. Irani. "Multi-Interval Discretization of Continuous-Valued Attributes for Classification Learning." In *IJCAI*, 1993.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[GC20] Yan Gao and Yan Cui. "Deep Transfer Learning for Reducing Health Care Disparities Arising from Biomedical Data Inequality." *Nature Communications*, **11**(1):5131, December 2020.

[GPM14] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. "Generative Adversarial Networks." *arXiv:1406.2661 [cs, stat]*, June 2014.

[GSS20] Luise Gootjes-Dreesbach, Meemansa Sood, Akrishta Sahay, Martin Hofmann-Apitius, and Holger Fröhlich. "Variational Autoencoder Modular Bayesian Networks for Simulation of Heterogeneous Clinical Study Data." *Frontiers in Big Data*, **3**:16, May 2020.

[GW18] Lovedeep Gondara and Ke Wang. "MIDA: Multiple Imputation Using Denoising Autoencoders." 02 2018.

[Hea19] Office of the National Coordinator for Health Information Technology. "Office-Based Physician Electronic Health Record Adoption Health IT Quick-Stat #50." https://www.healthit.gov/data/quickstats/office-based-physician-electronic-health-record-adoption, January 2019.

[Hou20] Xiurui Hou. *Hybrid Deep Neural Networks for Mining Heterogeneous Data.* PhD thesis, New Jersey Institute of Technology, 2020.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory." *Neural Comput.*, **9**(8):1735–1780, nov 1997.

[IS15a] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.", March 2015. arXiv:1502.03167 [cs].

[IS15b] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.", March 2015.

[JAB21] Sebastian Jäger, Arndt Allhorn, and Felix Bießmann. "A Benchmark for Data Imputation Methods." *Frontiers in Big Data*, **4**:693674, July 2021.

[JPR19] Anil Jadhav, Dhanya Pramod, and Krishnan Ramanathan. "Comparison of Performance of Data Imputation Methods for Numeric Dataset." *Applied Artificial Intelligence*, **33**(10):913–933, August 2019.

[JYB21] Daniel Jarrett, Jinsung Yoon, Ioana Bica, Zhaozhi Qian, Ari Ercole, and Mihaela van der Schaar. "Clairvoyance: A Pipeline Toolkit for Medical Time Series." In *International Conference on Learning Representations*, 2021.

[KB14] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization.", 2014.

[KBB19] Joseph Kearney, Shahid Barkat, and Arnab Bose. "Autoimpute." https://github.com/kearnz/autoimpute, 2019.

[KK16a] Sungil Kim and Heeyoung Kim. "A New Metric of Absolute Percentage Error for Intermittent Demand Forecasts." *International Journal of Forecasting*, **32**(3):669–679, July 2016.

[KK16b] Sungil Kim and Heeyoung Kim. "A new metric of absolute percentage error for intermittent demand forecasts." *International Journal of Forecasting*, **32**(3):669–679, July 2016.

[KL51] S. Kullback and R. A. Leibler. "On Information and Sufficiency." *The Annals of Mathematical Statistics*, **22**(1):79 – 86, 1951.

[KMF17] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. "Lightgbm: A highly efficient gradient boosting decision tree." *Advances in neural information processing systems*, **30**:3146–3154, 2017.

[Kro21] Tilman Krokotsch. "The Great Autoencoder Bake Off." https://github.com/tilman151/ae_bakeoff, 2021.

[KSR17] Andrzej S. Krolewski, Jan Skupien, Peter Rossing, and James H. Warram. "Fast Renal Decline to End-Stage Renal Disease: An Unrecognized Feature of Nephropathy in Diabetes." *Kidney International*, **91**(6):1300–1311, June 2017.

[KW14] Diederik P. Kingma and Max Welling. "Auto-Encoding Variational Bayes." *arXiv:1312.6114 [cs, stat]*, May 2014.

[LCH18] Xiang Li, Shuo Chen, Xiaolin Hu, and Jian Yang. "Understanding the Disharmony between Dropout and Batch Normalization by Variance Shift.", January 2018. arXiv:1801.05134 [cs, stat].

[Comment: 9 pages, 7 figures.]

[Lea] "Learning Internal Representations by Error Propagation — Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1." https://dl.acm.org/doi/10.5555/104279.104293.

[Lit88] Roderick J. A. Little. "A Test of Missing Completely at Random for Multivariate Data with Missing Values." *Journal of the American Statistical Association*, **83**(404):1198–1202, December 1988.

[LJR18] Lisha Li, Kevin Jamieson, Afshin Rostamizadeh, Katya Gonina, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. "Massively Parallel Hyperparameter Tuning.", 2018.

[LMX18] Tianyu Li, Yukun Ma, Jiu Xu, Björn Stenger, Chen Liu, and Yu Hirate. "Deep Heterogeneous Autoencoders for Collaborative Filtering." In *2018 IEEE International Conference on Data Mining (ICDM)*, pp. 1164–1169, November 2018.

[LTS14] Hiddo J. Lambers Heerspink, Hocine Tighiouart, Yingying Sang, Shoshana Ballew, Hasi Mondal, Kunihiro Matsushita, Josef Coresh, Andrew S. Levey, and Lesley A. Inker. "GFR Decline and Subsequent Risk of Established Kidney Outcomes: A Meta-Analysis of 37 Randomized Controlled Trials." *American Journal of Kidney Diseases*, **64**(6):860–866, 2014.

[McK10] Wes McKinney. "Data Structures for Statistical Computing in Python." In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pp. 56 – 61, 2010.

[MHm19] David R. MacIver, Zac Hatfield-Dodds, and many other contributors. "Hypothesis: A new approach to property-based testing." November 2019.

[Mic01] Daniele Micci-Barreca. "A preprocessing scheme for high-cardinality categorical attributes in classification and prediction problems." *ACM SIGKDD Explorations Newsletter*, **3**(1):27–32, July 2001.

[Mic22] Umberto Michelucci. "An Introduction to Autoencoders." *arXiv:2201.03898 [cs]*, January 2022.

[MKA18] John T. McCoy, Steve Kroon, and Lidia Auret. "Variational Autoencoders for Missing Data Imputation with Application to a Simulated Milling Circuit." *IFAC-PapersOnLine*, **51**(21):141–146, 01 2018.

[MNW18] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I. Jordan, and Ion Stoica. "Ray: A Distributed Framework for Emerging AI Applications.", September 2018.

[MP21] Karthika Mohan and Judea Pearl. "Graphical Models for Processing Missing Data." *Journal of the American Statistical Association*, **116**(534):1023–1037, April 2021.

[MPL13] David M Maslove, Tanya Podchiyska, and Henry J Lowe. "Discretization of Continuous Features in Clinical Datasets." *Journal of the American Medical Informatics Association : JAMIA*, **20**(3):544–553, 2013.

[MSW18] Christina Mack, Zhaohui Su, and Daniel Westreich. *Types of Missing Data*. Agency for Healthcare Research and Quality (US), February 2018.

[MTT20] Chao Ma, Sebastian Tschiatschek, Richard Turner, José Miguel Hernández-Lobato, and Cheng Zhang. "VAEM: A Deep Generative Model for Heterogeneous Mixed Type Data." In *Advances in Neural Information Processing Systems*, volume 33, pp. 11237–11247. Curran Associates, Inc., 2020.

[Mue] Zachary Mueller. "Using AutoEncoders with Tabular Data." https://walkwithfastai.com/tab.ae.

[NDA19] Keith C. Norris, O. Kenrik Duru, Radica Z. Alicic, Kenn B. Daratha, Susanne B. Nicholas, Sterling M. McPherson, Douglas S. Bell, Jenny I. Shen, Cami R. Jones, Tannaz Moin, Amy D. Waterman, Joshua J. Neumiller, Roberto B. Vargas, Alex

A. T. Bui, Carol M. Mangione, Katherine R. Tuttle, and CURE-CKD investigators. "Rationale and Design of a Multicenter Chronic Kidney Disease (CKD) and at-Risk for CKD Electronic Health Records-Based Registry: CURE-CKD." *BMC nephrology*, **20**(1):416, 11 2019.

[NOG20] Alfredo Nazábal, Pablo M. Olmos, Zoubin Ghahramani, and Isabel Valera. "Handling incomplete heterogeneous data using VAEs." *Pattern Recognition*, **107**:107501, November 2020.

[Code: https://github.com/probabilistic-learning/HI-VAE VAEs to adapt to heterogeneous data + MCAR.]

[Oli06] Travis E Oliphant. *A guide to NumPy*, volume 1. Trelgol Publishing USA, 2006.

[PC23] Steven Pan and Sixia Chen. "Empirical Comparison of Imputation Methods for Multivariate Missing Data in Public Health." *International Journal of Environmental Research and Public Health*, **20**(2):1524, January 2023.

[PGM19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library." In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, et al., editors, *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019.

[PVG11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. "Scikit-learn: Machine Learning in Python ." *Journal of Machine Learning Research*, **12**:2825–2830, 2011.

[RHJ04] John D. Rozich, Ramona J. Howard, Jane M. Justeson, Patrick D. Macken, Mark E. Lindsay, and Roger K. Resar. "Standardization as a Mechanism to Improve Safety in Health Care." *Joint Commission Journal on Quality and Safety*, **30**(1):5–14, January 2004.

[Rub76] Donald B. Rubin. "Inference and Missing Data." *Biometrika*, **63**(3):581–592, 12 1976.

[SHK14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting." *Journal of Machine Learning Research*, **15**(56):1929–1958, 2014.

[SLV18] Rianne M Schouten, Peter Lugtig, and Gerko Vink. "Generating missing values for simulation purposes: a multivariate amputation procedure." *Journal of Statistical Computation and Simulation*, **88**(15):2909–2930, 2018.

[Smi23] Garrett Smith. "guildai." https://github.com/guildai/guildai, 2023.

[SS23] Heajung Suh and Jongwoo Song. "A Comparison of Imputation Methods Using Machine Learning Models." *Communications for Statistical Applications and Methods*, **30**(3):331–341, May 2023.

[SZS22a] Rianne M Schouten, Davina Zamanzadeh, and Prabhant Singh. "pyampute: a Python library for data amputation.", August 2022.

[SZS22b] Rianne M Schouten, Davina Zamanzadeh, and Prabhant Singh. "Pyampute: A Python Library for Data Amputation.", August 2022.

[SZS22c] Rianne M Schouten, Davina Zamanzadeh, and Prabhant Singh. "pyampute: a Python library for data amputation." Scientific Computing with Python conference, 2022.

[TAD19] Katherine R. Tuttle, Radica Z. Alicic, O. Kenrik Duru, Cami R. Jones, Kenn B. Daratha, Susanne B. Nicholas, Sterling M. McPherson, Joshua J. Neumiller, Douglas S. Bell, Carol M. Mangione, and Keith C. Norris. "Clinical Characteristics of and Risk Factors for Chronic Kidney Disease Among Adults and Children: An Analysis of the CURE-CKD Registry." *JAMA Network Open*, **2**:e1918169–e1918169, 2019.

[VG11] Stef Van Buuren and Karin Groothuis-Oudshoorn. "mice: Multivariate imputation by chained equations in R." *Journal of statistical software*, **45**(1):1–67, 2011.

[VLB08] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. "Extracting and Composing Robust Features with Denoising Autoencoders." In *Proceedings of the 25th International Conference on Machine Learning - ICML '08*, pp. 1096–1103, Helsinki, Finland, 2008. ACM Press.

[VPL17] Isabel Valera, Melanie F. Pradier, Maria Lomeli, and Zoubin Ghahramani. "General Latent Feature Models for Heterogeneous Datasets." June 2017.

[WNN19] Christopher K. I. Williams, Charlie Nash, and Alfredo Nazábal. "Autoencoders and Probabilistic Inference with Missing Data: An Exact Solution for The Factor Analysis Case." 02 2019.

[YJS18] Jinsung Yoon, James Jordon, and Mihaela Schaar. "GAIN: Missing Data Imputation Using Generative Adversarial Nets." In *Proceedings of the 35th International Conference on Machine Learning*, pp. 5689–5698. PMLR, July 2018.

[ZCD18] M. Zaharia, A. Chen, A. Davidson, A. Ghodsi, S. Hong, A. Konwinski, Siddharth Murching, Tomas Nykodym, Paul Ogilvie, Mani Parkhe, Fen Xie, and Corey Zumar. "Accelerating the Machine Learning Lifecycle with MLflow." *IEEE Data Eng. Bull.*, 2018.

[ZPD21] Davina J. Zamanzadeh, Panayiotis Petousis, Tyler A. Davis, Susanne B. Nicholas, Keith C. Norris, Katherine R. Tuttle, Alex A. T. Bui, and Majid Sarrafzadeh. "Autopopulus: A Novel Framework for Autoencoder Imputation on Large Clinical Datasets." *Annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE Engineering in Medicine and Biology Society. Annual International Conference*, **2021**:2303–2309, November 2021.