Task 1 – code cells 4-8: We begin by first splitting the dataset into training, validation, and testing partitions. All subsequent exploratory analysis revolve around the training split to prevent any form of leakage, as well as respecting the convention that the val/test set are unseen. Our training-set consists of 73525 patients, each with 50 healthcare-related features measured, thus yielding a sample-to-ratio of 1471. Such high ratio allows the model to learn more robust patterns, possibly yielding stronger generalization capability. There are 37 categorical features such as medical specialty of admitting physician, and 13 numeric, count features like the number of medications taken, all of which are used to predict probability of hospital readmission. There are several challenges tied to this dataset, such as high missing rate for variables like patient weight (96%), and variables containing many categories with a heavy skew to only a couple of them, thus it may bias the model towards ignoring minority classes, impacting generalization. We can anticipate from now that the presence of many categories causes one-hot encoding to introduce sparsity and increased dimensionality of the dataset, which together increases model complexity, and training times. Furthermore, several patients (not all) have multiple encounters in the hospital, which introduces rows with correlated features. See Table 1 for initial descriptive statistics and DeShazo et al. (2014) for a more intricate description of each variable.

Task 2 code cells 10-63: There are two kinds of preprocessing we perform: "global" preprocessing refers to procedures that are agnostic to the train-validation-test split, while "train-set dependent" preprocessing refers to those that rely on training-set statistics, where we take care to avoid data leakage to both test and validation splits. We first drop 5 features which according to the train-set have 1) high missing rate (>80%) and imputation wouldn't be able to interpolate with non-repetitive values to the column, 2) medications 'examide' and 'citoglipton' because all measurements are the same and thus uninformative, and 3) the unique IDs of patients and their encounters because we argue they offer no insight into readmission rate. The same columns are dropped for validation/test splits because the model can't be evaluated on those features. Next, we perform imputation on features with missing values using the column-wise mode from the trainset. We globally remove patients with 'discharge disposition' IDs of 11, 13, 14, 19, 20, or 21 as they are related to death or hospice (DeShazo et al., 2014), thus, by common sense they can't be readmitted to the hospital, but our model's objective is to use features from patients alive to predict hospital readmission. We proceed in re-grouping categories into a dummy 'Other' category for 4 categorical variables 'admission type', 'admission source', 'disposition ID' and 'medical specialty' when the relative proportion of a category falls below certain threshold. This threshold is decided based on the appearance rate in the trainset's respective column feature, and the same categories are regrouped in the val/test set. Regrouping may lead to some information lost on minority categories, but it helps decrease sparsity owed to one-hot encoding and model complexity which aids interpretability. Figures 1 and 2 show these variables before and after grouping. Afterwards, we globally encode diagnosing features ('diagnoses 1, 2 and 3') according to their ICD-9 coding scheme (see Table 2). This also helps in reducing model complexity by grouping different values, such as numbers between [390, 459], into a category like "Circulatory". We observe the categories' relative proportions in Figure 3. We globally binarize the "hospital readmission" feature by grouping '>30' and '<30' into class 1 ('readmitted'), else into class 0 ('not readmitted'), where we don't observe class imbalance, which stabilizes training of the model. Code cell 46 shows the main preprocessing pipeline, annotated with whether it's a "global" or "train-dependent" procedure, and it's implemented in a way that if it takes the val/test splits, we avoid data leakage. We compute summary stats for the trainset's numerical features in Table 3 and observe their distributions in violin plots (Figure 4) and histograms (Figure 5). We choose not to rescale to maintain their characteristic as count data. We observe some extreme values like 129 lab procedures; however, we don't treat them as outliers for the following reasons: if we treat it as an outlier, then we'd have to remove patients with amount of lab procedures greater than 129 in the test set too. However, the test set is designed to emulate real-life, and there is no way for us to decide that it's impossible for a patient to have more than 120 lab procedures in a complicated healthcare setting. Additionally, whether our models have strong generalization capability is also dependent on their tolerance/robustness to these 'outliers'. Lastly, these values might seem extreme if we assume a Gaussian distribution, however for count data it's more appropriate to view them through the lenses of the Poisson distribution. We have 71771 patients remaining. Further preprocessing needed to train the machine learning models is to one-hot encode categorical variables, where we take care in ignoring unrecognized categories in the validation/test split when evaluating the model. For example, as we mentioned in Figures 1 and 2, by chance in the trainset we only observe 16 values out of 21 for ‘admission source’, thus in the val/test split we ignore any category unseen in the trainset. This is a reasonable approach for ML models given that the test split is supposed to emulate real-life settings, where unseen categories are unavoidable, reflecting a core limitation of ML being unable to accommodate unseen features in training. Code cell 51 shows implementation details. As expected, one-hot encoding introduces a total of 216 features, which greatly increases sparsity and subsequent model complexity. To mitigate this, especially because later tasks require us to visualize the models’ learnt coefficients, we add to our preprocessing pipeline feature selections steps: we 1) drop highly correlated features by computing the correlation matrix on the trainset’s features and identify 17 columns with a correlation coefficient greater than 0.8. These same columns are dropped in the val/test splits. Next, we use the Boruta algorithm to select the most relevant features that predicts the readmission rate. It works by leveraging a random forest classifier (RFC) trained on predicting the target variable and comparing the importance scores of the original features with shadow features, which are random permutations of the original features, and then ranking the most relevant features that predict the target. The Boruta algorithm is very intensive, thus due to computational constraints we run it once and manually set a global variable (see code cell 57) which is a list of 31 selected features which will be used to index the train-val-test folds in the nested cross-validation (CV) later. These selected features include all numerical features, some categories of admission sources, ages between 70-90, 2 medical specialties Emergency/Trauma and Obstetrics/Gynecology among others, which are reasonable predictors of hospital readmission. See Table 4 for descriptive stats of the selected variables after this preprocessing pipeline.

Algorithm 1 (Nested) Cross-Validation

```
1: Input: model, dataset df, hyperparameters hparams, outer cross-fold,
   inner cross-fold
2: Output: cross-validation scores per hyperparameter, test scores of best
   hyperparameter
3: test_scores_best_hparam = [ ] # list where each index is a test score for best
   hyperparameter according to inner CV. Note that it could be a list of lists
   if inner CV disagrees in best hyperparameter
4: cv_scores_per_hparam = [ ] # list of lists, where each index are the CV scores
   for one hyperparameter
5: for outer cross-fold validation do
6:   for hparam in hparams do
7:     df_train_val, df_test = k_fold_split(df)
8:     cv_scores = [ ]
9:     for inner cross-fold validation do
10:      fold_df_train, fold_df_val = k_fold_split(df_train_val) # ensure no over-
         lap in-between folds
11:      preprocess(fold_df_train, in-place)
12:      preprocess(fold_df_val, given = fold_df_train, in-place)
13:      model.fit(fold_df_train) # model instantiated with hparam
14:      cv_score = accuracy_metric(fold_df_val, model.predict(fold_df_val))
         cv_scores.add(cv_score)
15:    end for
16:    cv_scores_per_hparam.add(cv_scores)
17:  end for
18:  Select hyperparameter with highest CV score
19:  preprocess(df_test, given=fold_df_train, in-place)
20:  test_score = accuracy_metric(df_test, model.predict(df_test)) # model fit
     with best hyperparameter according to CV
21:  test_scores_best_hparam.add(test_score)
22: end for
23: return cv_scores_per_hparam, test_scores_best_hparam
```

Task 3: code cells 64-78: For binary classification, we choose to mainly report the F1 score, which is the harmonic mean between sensitivity and specificity, thus it accounts for false negatives (FN) and false positives (FP) unlike ‘accuracy’, which is important in a healthcare setting since a FN may compromise a patient’s need for rehospitalization. We also record supplementary metrics like log-loss to monitor the stability of training (high log-loss may indicate poor model hyperparameter setting), and the ROC-AUC score because it summarizes the model's ability to distinguish between classes across different classification thresholds. In addition to the linear SVM, we choose to implement a logistic regression (LogReg), Gaussian Naïve Bayes (NB) classifier and a RFC. These are based on the following considerations: they are simple to implement, have fast training times, and are structurally similar so we can visualize their coefficients or feature importance later. Furthermore, we also consider that they can be embedded in the same (nested) CV framework for simplifying computations, thus we don’t resort to deep neural networks as they’re trained differently, non-parametric KNNs as we can’t visualize them, nor Bayesian models since they’re harder to implement and thus embed in our CV framework, don’t natively provide point estimates and are harder to interpret. In **Algorithm 1** we describe this general, nested CV framework we use to train all candidate model, where we retrieve CV when the outer fold is set to $k = 1$ and only a single test score is returned for the best hyperparameter. Code cell 65 implements it. We only work with a subset of 4200 patients due to computational constraints, especially for training the linear SVM. For the linear models, we explore over a grid of values for their regularizers, for NB we search over Laplace smoothing values, and for RFC it’s number of trees. Figures 6-9 show the mean CV performance metrics with standard deviation (SD) as a function of the different hyperparameters for each model, where we notice similar mean F1 CV score across models for their optimal hyperparameters, which is around 0.65. The main difference comes from that non-linear models (NB and RFC) have more stable training and consistent predictions than their linear counterparts (linear SVM and LogReg) as noted by generally lower SD values for mean CV score and faster training times without running into errors such as non-convergence in LogReg. We report test performance for optimal hyperparameter per model in Table 5, where we highlight the highest ranking, as per test F1 score is Naïve Bayes with a variance smoothing of 0.42, but it also shows the lowest AUC, indicating that at different classification thresholds, the NB may classify poorly. This is followed by the linear SVM (C: 0.042), LogReg (C: 4.2) and RFC (trees: 242).

Task 4: code cells 79-92. For the linear models trained on their optimal hparams, we plot their model coefficients in Figures 10 and 11. For the NB, we plot the Gaussian distribution estimated for a subset of 2 features (time in hospital and number of lab procedures) out of 31 for simplicity (Figure 12). For the RFC, we both plot the average feature importance across 242 estimators (Figure 13) and the decision path of tree 42 with a max depth of 3 (Figure 14). We mainly observe similar estimated coefficients between linear models, agreeing that for example age [80-90] significantly predicts hospital readmission, which makes sense because it's reasonable that older patients need more hospital care. Both linear models also agree on admission from another hospital being significant to predict no readmission. The RFC seems to mainly focus on numerical features which doesn't make much sense. This is probably because it's sensitive to the scale of training data as we didn't rescale it. In contrast, the regularization for both linear models geared them to avoid being affected by the scale of the numerical features. There is no "feature" importance in NB. It differs with the other models because it mainly operates with probability distributions, which can offset some of the problems brought by features with values of different scales as they are represented by probabilities rather than the values themselves.

Task 5 code cells 93-111: We follow **Algorithm 1** and implement nested CV (code cell 93) by setting outer and inner $k = 3$. We explore the same hyperparameters for each model and compute the mean F1 test score with SD using the outer test splits, with the model trained on the optimal hyperparameters according to their inner CV results. We note that the inner CV returns different optimal hyperparameters depending on the outer train-test split. We report the test performance for all models that achieved the highest inner CV F1 score for each outer fold. Naïve Bayes is the de-facto highest performing model with a smoothing of 4.2 and a F1 test score of `0.681 +/- 0.002.` . This is different to the hyperparameter 0.42 found in Table 5, elucidating that the outer k-fold helps identifying a better hyperparameter as it considers different training distributions.

Task 6: a main limitation with this dataset is that it doesn't follow a standardized format for recording patient features, thus not all hospital records the same measurements, which may lead to a lot of features like medications with null values, impacting performance in real life. The dataset can be improved by following unifying guidelines like FHIR. In technicalities, we could have also rescaled the numerical features at the expense of the semantics of count data that may help especially the RFC. One key limitation with nested CV is that there are inconsistent results for hyperparameters identified across outer folds, and averaging performance across folds could be a form of "train-test" leakage. The models we implemented are also constrained by their expressivity, i.e., we could have resorted to hybrid deep learning + standard machine learning architectures that can improve performance without compromising explainability of linear models. We could have improved performance if we implement a Poisson-distribution based NB instead of Gaussian as it models count data better. We could have also resorted to Binomial distributions for modelling the categorical features. Another limitation with our pipeline is that we don't explore a lot of hyperparameters, nor do we employ the entire dataset due to computational constraints. With enough computational power, we can explore more.

Bibliography:

Strack, B., DeShazo, J. P., Gennings, C., Olmo, J. L., Ventura, S., Cios, K. J., & Clore, J. N. (2014). Impact of HbA1c Measurement on Hospital Readmission Rates: Analysis of 70,000 Clinical Database Patient Records. *BioMed Research International*, *2014*, 1–11. https://doi.org/10.1155/2014/781670

Kaggle competitor. (2021). Predicting Hospital Readmission of Diabetics. Kaggle.com. https://www.kaggle.com/code/chongchong33/predicting-hospital-readmission-of-diabetics/notebook

For example, in the trainset, we observe for ‘admission source’ 16 categories, and we regroup 9 into ‘Other’ because each have a relative proportion below a threshold of 0.1%, resulting in their cumulative proportion being 0.5%. These same 9 categories are regrouped in the validation/test sets regardless of their proportion to ensure consistency between training and val/test splits.