Case Study 2: Predicting Critical Temperature

By: Allen Hoskins and Brittany Lewandowski

September 19, 2022

1. INTRODUCTION

Diabetes is a metabolic disease impacting 37.3 million Americans. Those affected by the disease have complications producing insulin, a chemical messenger that our body uses to store energy. Although it is uncommon, diabetics can be hospitalized for having critically low or high blood glucose levels. These hospitalizations can be life threatening and should be minimized at all costs.

In this case study, we will use a diabetes data set procured by Dr. Slater, to identify what factors most significantly result in diabetics getting readmitted to hospitals. To accomplish this, we will build a Logistic Regression model and extract its respective feature importances. It is our hope that this research can be leveraged by medical professionals to help treat hospitalized diabetics and to ensure that these patients are not readmitted in the future.

2. METHODS

DATA UNDERSTANDING:

Data used in this case study was a diabetes.csv provided by Dr. Slater. Our diabetes.csv contained data related to hospitalized diabetic patients including columns such as: `readmitted`, `patient_nbr`, `insulin`, and `time_in_hospital`, Upon reviewing the contents of our data set, we saved the data into a data frame named "diabetes_data" and began pre-processing.

DATA PREPROCESSING:

The first step we performed in pre-processing was reviewing our full data set. Immediately, we recognized that missing values existed in the columns of:

1. race
2. weight
3. payer_code
4. medical_specialty
5. diag_1
6. diag_2
7. diag_3

Given that machine learning models do not handle missing values well, we imputed them using appropriate statistical methods. Full details on how these columns were imputed can be found in the sub-header of this case study titled, "Data Imputation."

After identifying that missing values existed in our data set, we ran the command, `diabetes_data.info()` and noted the following details of our data frame:

> Our data frame contained 101,766 rows. Our data frame contained 50 columns. No null values existed in our data frame. Our data frame contained 13 numeric columns. Our data frame contained 37 categorical columns.

From this output we recognized that one hot encoding (OHE), would need to be performed on our categorical columns. For additional details on our OHE process, please see the sub-header of this case study titled, `One Hot Encoding.`

The next step performed in pre-processing was running the command `diabetes_data.describe()` to view the summary statistics of our data frame. Output from this command showed that several columns contained outliers. This was something we remained cognizant of throughout our analysis.

Finally, to view the distributions of our categorical columns with missing values, we created count plots. Visualizing these columns was important, as it helped us determine what data imputation method was most appropriate for our data. Output from our count plots showed that all seven of our columns with missing data contained non-normal distributions (Exhibit 1.0). Since all seven columns were of the categorical data type, we noted that imputing these columns with either the mean or median value would be appropriate.

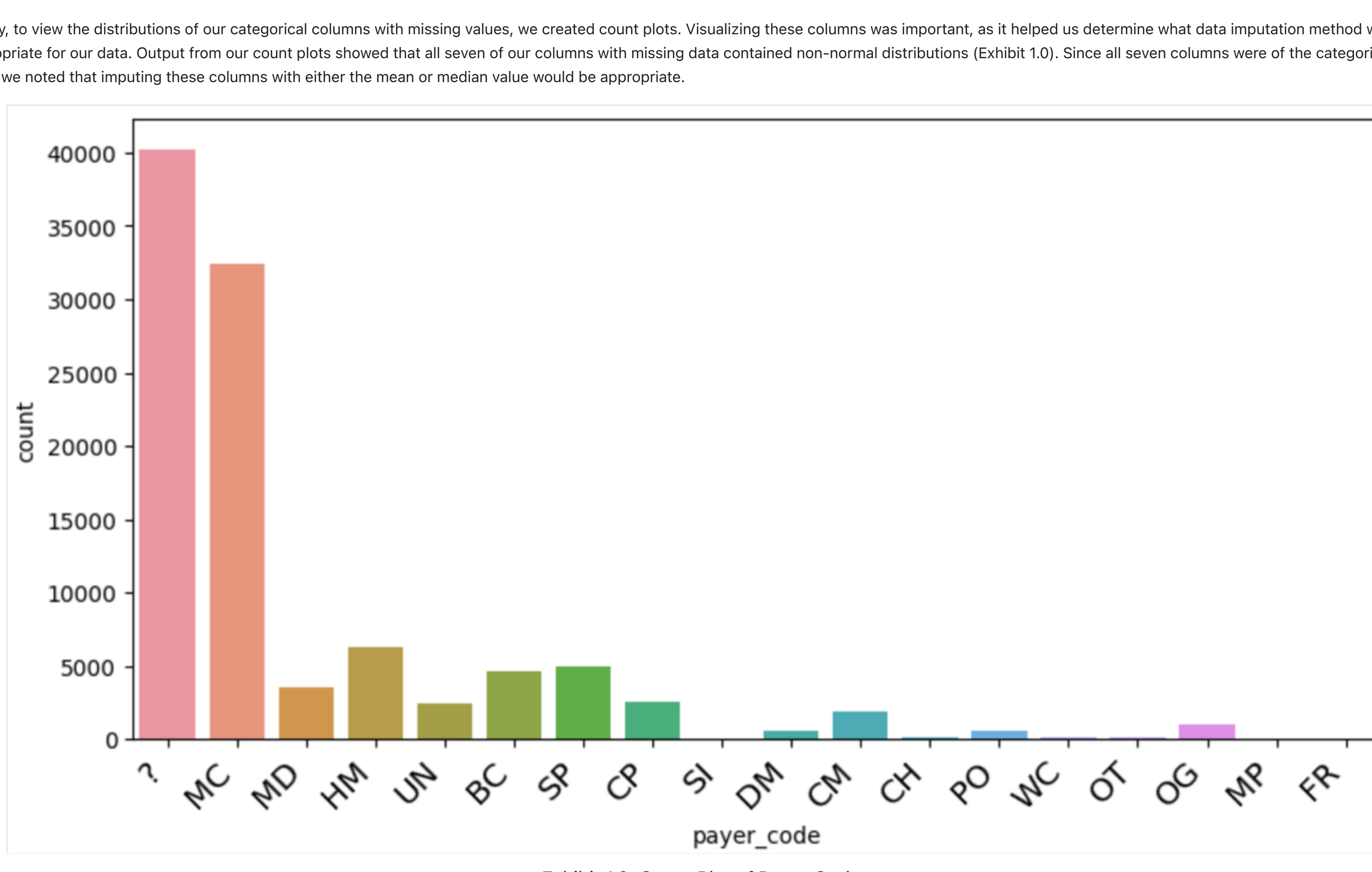

Exhibit 1.0: Count Plot of Payer_Code

The last step performed in pre-processing was calculating the percentage of missing values in our categorical columns. Exhibit 1.1 details our findings:

Column Name	Percentage of Missing Values
race	2.23%
weight	96.85%
payer_code	39.55%
medical_specialty	49.08%
diag_1	0.02%
diag_2	0.35%
diag_3	1.39%

Exhibit 1.1: Percent Missing Values

DATA IMPUTATION:

Upon reviewing our full data set and calculating the percentage of missing values that existed in our categorical columns, we proceeded to impute our missing values.

All the missing values in our data set were denoted by: `?`. Since computers cannot impute data with special characters, we converted the question marks to "NaN". Once this was complete, we re-calculated the sum of missing values in our columns to validate that no data loss had occurred in our conversion process.

When we considered imputing the columns: `race`, `payer_code`, `medical_specialty`, `diag_1`, `diag_2`, and `diag_3`, we tried two different approaches. One approach was imputing these columns with the mode of each column, and the second approach was leaving the columns as is with missing values. We fit our Logistic Regression model on both approaches and found that our performance results were negligible. Consequently, we decided to leave the columns with missing values, as we felt this represented our data the best.

The first column we chose to impute was our `weight` column. Given that 96% of the data in our `weight` column were missing, we chose to drop the column from our data set.

Next, we imputed the columns: `diabetesMed`, `change`, and `readmitted` with values of 0 and 1. This was done to simplify OHE as these columns had a maximum of three classes. Please note that although the column: `readmitted` contains three classes, we chose to convert it to a binary variable as we are only concerned with whether a patient has been readmitted or not. Exhibit 1.2 details our conversion process of these columns:

Column Name	Original Classes	Data Dictionary for Converted Classes
diabetesMed	No Yes	0=No 1=Yes
Change	Ch No	0=No 1=Yes
readmitted	NO <30 >30	0=No 1=Yes

Exhibit 1.2: Imputation process for the columns: "diabetesMed", "change", and" readmitted"

RE-CODING CATEGORICAL COLUMNS:

When viewing the shape of our data set, we recognized that if we one hot encoded all 37 of our categorical variables, that our data set would be extremely wide. As a result, we decided to reduce the classes in each categorical variable by specifying a threshold for infrequent observations. Exhibit 1.3 details the thresholds that were chosen for each variable, as well as explanations as to why thresholds were chosen.

Column Name	Selected Threshold	Explanation
payer_code	0.02	~90% of our data falling into the top 7 classes
medical_specialty	0.03	~85% of our data falling into the top 5 classes
max_glu_serum	0.02	~96% of our data falling into the top 2 classes
A1Cresult	0.08	~91% of our data falling into the top 2 classes
metformin	0.1	~98% of our data falling into the top 2 classes
repalglinide	0.01	~99% of our data falling into the top 2 classes
nateglinide	0.9	~99% of our data falling into the top class
chloropropamide	0.9	~99% of our data falling into the top class
glimepiride	0.9	~95% of our data falling into the top class
glipizide	0.1	~97 of our data falling into the top 2 classes
glyburide	0.09	~98 of our data falling into the top 2 classes
pioglitazone	0.06	~98 of our data falling into the top 2 classes
rosiglitazone	0.05	~98% of our data falling into the top 2 classes
acarbose	0.9	~99% of our data falling into the top class
miglitol	0.9	~99% of our data falling into the top class
tolazamide	0.0004	~99% of our data falls into the top class
glyburide_metformin	0.9	~99% of our data falling into the class "No"
diag_1	0.0075	many of the columns contained values <= 0.000010
diag_2	0.0075	many of the columns contained values <= 0.000010
diag_3	0.0075	many of the columns contained values <= 0.000010

Exhibit 1.3: Detailed Recoding Threshold

ONE HOT ENCODING:

Once we imputed our missing values and re-coded our categorical columns we separated our diabetes data set into two variables, one containing all our numeric columns and the other containing all our categorical columns. Next, using Pandas' get dummies function, we one hot encoded our categorical columns and joined our one hot encoded data to our numeric columns to arrive at our final full data set.

Please note that for modeling, we scaled our non-hot encoded data to ensure that our full data set was on the same scale. For additional modeling details, please see our sub-header below titled "Modeling."

EXPLORATORY DATA ANALYSIS:

For our exploratory data analysis (EDA), we began by viewing histograms and pair plots of our data (Exhibits 1.4 & 1.5). Two takeaways from these visualizations included:

1. Many of our numeric columns exhibited non-normal distributions.
2. Our variables were not on the same scale.

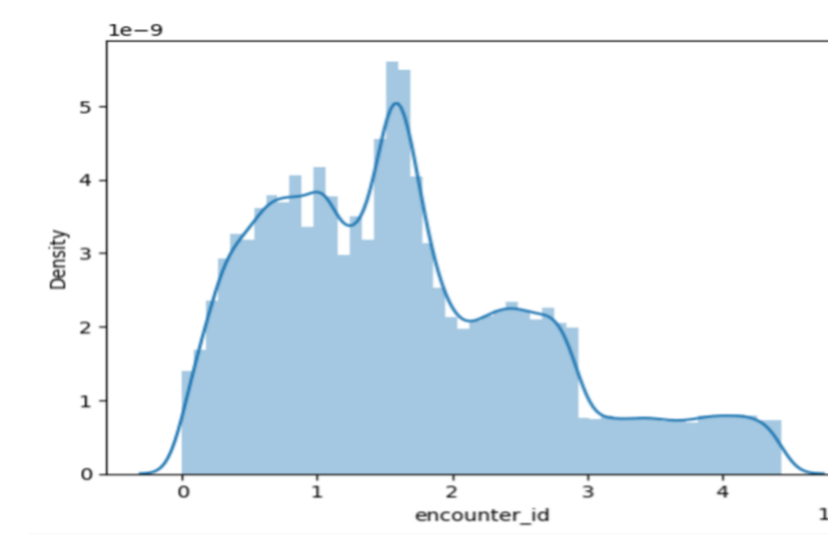

Exhibit 1.4: Encounter ID Density Plot

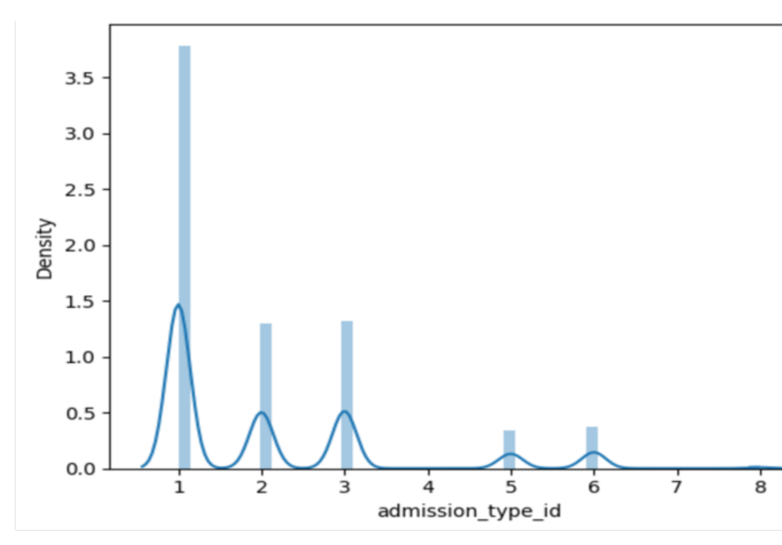

Exhibit 1.5: Admission Type ID Density Plot

After reviewing our histograms and pair plots, we assessed multicollinearity in our data by creating a correlation plot. Seeing that no columns had a correlation coefficient of 1.0, we chose not to remove any columns from our data set. At this point we were satisfied with our data and began evaluating if our data met our Logistic Regression modeling assumptions.

ASSUMPTIONS OF LOGISTIC REGRESSION MODELS:

The three key assumptions of Logistic Regression models include:

1. Independent variables have a linear relationship to the log loss of the response.
2. Absence of multicollinearity.
3. Lack of outliers.

To assess our first assumption, we created two log odds linear plots of our response variable versus the independent variables (Exhibit 1.6): "time_in_hospital" and "num_medications"

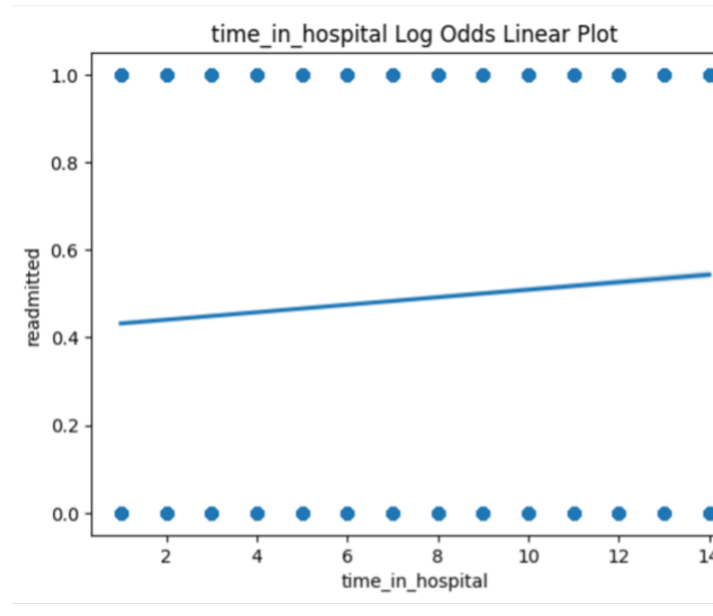

Exhibit 1.6: Log Odds for Time in Hospital

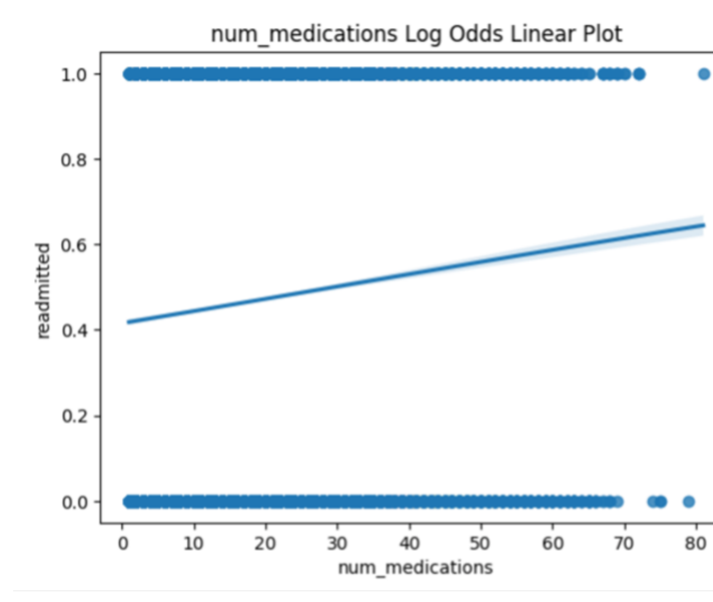

Exhibit 1.7: Log Odds for Num Meds Feature

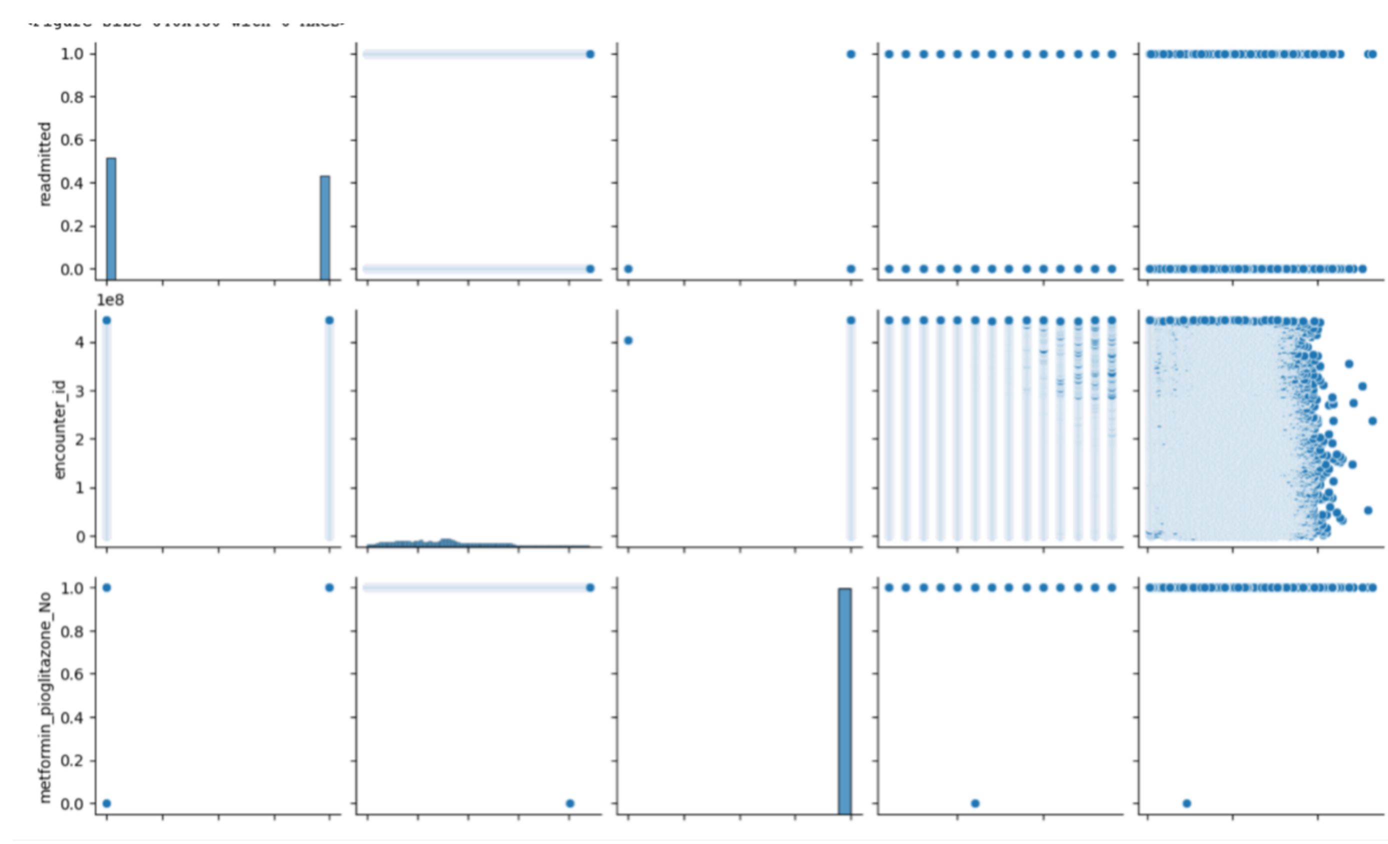

Exhibit 1.8: Snapshot of several pair plots generated from our data

Seeing that our log odds plots showed that our independent variables had a linear relationship to the log loss our response, we deemed that our first assumption was met.

To address our multicollinearity assumption, we created a correlation plot. Given that no columns had a correlation coefficient of 1.0, we proceeded assuming that this assumption was met.

The final assumption we addressed was lack of outliers. (Exhibit 1.8). As illustrated in our pair plots below, we did see that our data contained outliers. Given that the pair plots were built on un-scaled data, we proceeded in our analysis assuming that this assumption was met.

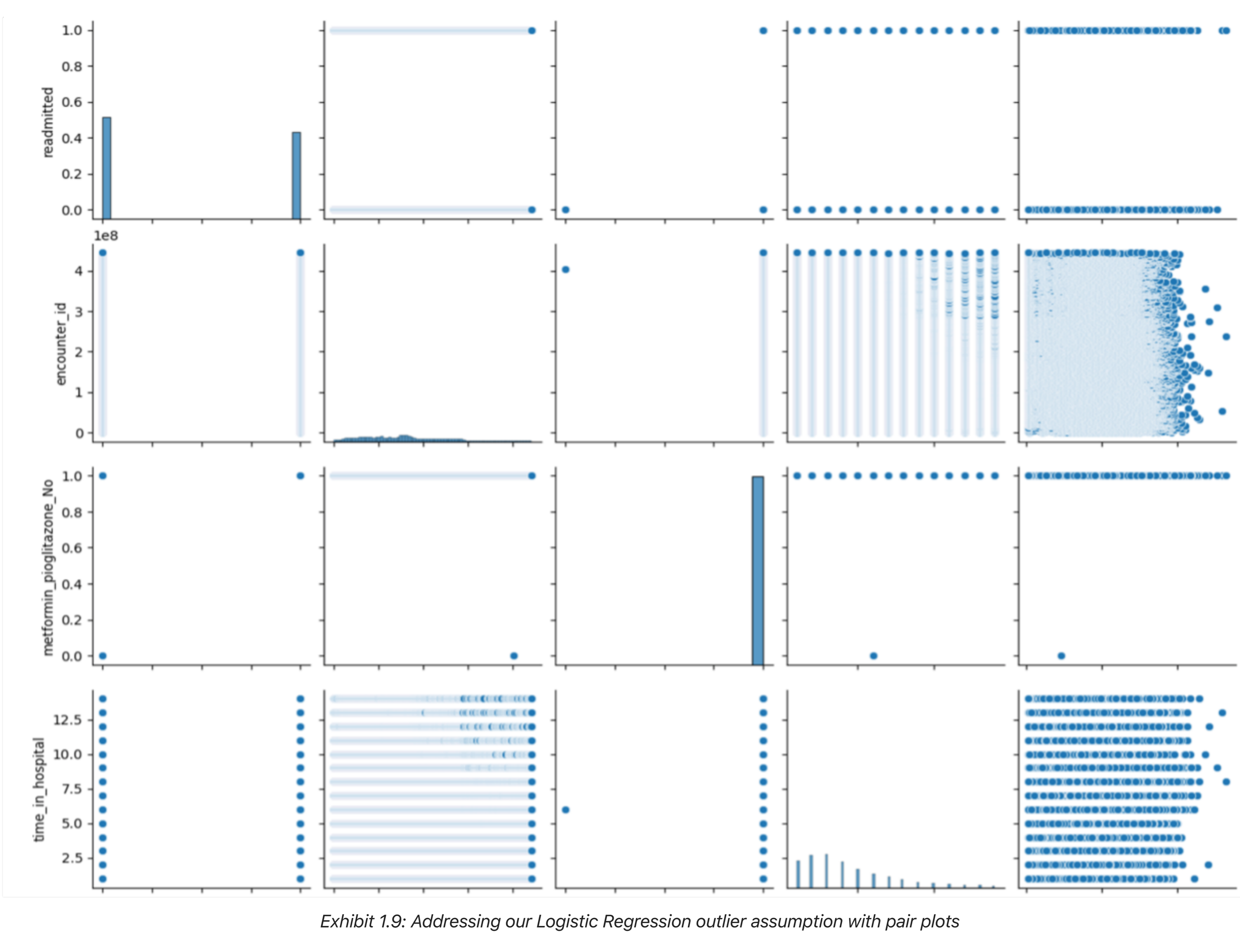

Exhibit 1.9: Addressing our Logistic Regression outlier assumption with pair plots

3. MODEL BUILDING & RESULTS

The use of Sklearn's `LogisticRegression` was used to model the data for this case study.

Models use: 10-fold Cross validation (`Kfold`), `random_seed = 0`, and `max_iter = 50000`, and scoring metric of `Negative Mean Absolute Error`

Model HalvingRandomSearchCV:

After preprocessing, EDA, and scaling the data, modeling was able to begin. To determine the best hyperparameters that we should use, we needed to iterate through several of `sklearn's` modules. We began with utilizing `GridSearchCV`, but due to the shape of our data and inability to scale our CPU, GPU, and Memory for the needs of this project, `GridSearchCV` was unable to complete and we needed to try other methods of tuning hyperparameters. With the use of Skelearn's `experimental` and `model_selection` packages we were able to utilize `HalvingRandomSearchCV` to obtain good, but potentially not the best hyperparameters for this model. `HalvingRandomSearchCV` combines the idea of `HalvingSearchCV` and `RandomizedSearchCV`. HalvingSearchCV works by modeliong all potential candidates with less data and selects half of the best performing models to add additional resources and data until a "best" model is output. `RandomizedSearchCV` randomly picks candidate modles from the grid to model.

We passed the below parameters into `HalvingRandomSearchCV` and the best model outputs were the following:

Halving Random Search CV Parameters:

```
"C":            np.logspace(-3,3,7),
"l1_ratio":     np.arange(0.0,1.0,0.1),
'solver':       ['saga'],
'penalty':      ['elasticnet'],
"tol":          [1e-9,1e-8,1e-7,1e-6,1e-5, 1e-4, 1e-3, 1e-2, 1e-1]
```

Best Model Output:

```
"C":            1.0
"l1_ratio":     0.2
"n_jobs":       -1
"penalty":      'elasticnet'
"solver":       'saga'
"tol":          1e-09
```

ElasticNetCV with GridSearchCV Tuned Parameters:

After performing `HalvingRandomSearchCV` to tune the model parameters, Sklearn's `cross_validate` was used to validate the model and determine final performance. The results of all 10 folds are below with a MAE score of 0.431234. We modeled several different metrics, but determined that MAE was the best metric for determine the best model. Model results for the 10 fold Cross Validation are below in Exhibit 2.0.

CV Run	fit_time	score_time	estimator	test_score	train_score
0	25.5363	0.00609469	LogisticRegression(l1_ratio=0.2, n_jobs=-1, penalty='elasticnet',random_state=0, solver='saga', tol=1e-09)	-0.431463	-0.431449
1	24.9823	0.00657201	LogisticRegression(l1_ratio=0.2, n_jobs=-1, penalty='elasticnet',random_state=0, solver='saga', tol=1e-09)	-0.431561	-0.431384
2	25.6763	0.00533915	LogisticRegression(l1_ratio=0.2, n_jobs=-1, penalty='elasticnet',random_state=0, solver='saga', tol=1e-09)	-0.432544	-0.431187
3	25.1806	0.00650811	LogisticRegression(l1_ratio=0.2, n_jobs=-1, penalty='elasticnet',random_state=0, solver='saga', tol=1e-09)	-0.433821	-0.431002
4	25.3891	0.00669408	LogisticRegression(l1_ratio=0.2, n_jobs=-1, penalty='elasticnet',random_state=0, solver='saga', tol=1e-09)	-0.432642	-0.431231
5	25.3641	0.00693393	LogisticRegression(l1_ratio=0.2, n_jobs=-1, penalty='elasticnet',random_state=0, solver='saga', tol=1e-09)	-0.427729	-0.431613
6	25.4331	0.00782394	LogisticRegression(l1_ratio=0.2, n_jobs=-1, penalty='elasticnet',random_state=0, solver='saga', tol=1e-09)	-0.425806	-0.432023
7	19.1147	0.00559735	LogisticRegression(l1_ratio=0.2, n_jobs=-1, penalty='elasticnet',random_state=0, solver='saga', tol=1e-09)	-0.426985	-0.431848
8	25.2208	0.008039	LogisticRegression(l1_ratio=0.2, n_jobs=-1, penalty='elasticnet',random_state=0, solver='saga', tol=1e-09)	-0.437991	-0.430757
9	25.2794	0.006809	LogisticRegression(l1_ratio=0.2, n_jobs=-1, penalty='elasticnet',random_state=0, solver='saga', tol=1e-09)	-0.4318	-0.431226
MEAN	24.7177	0.00664113		-0.431234	-0.431372

Exhibit 2.0: 10 Fold Cross Validation Model Results

4. CONCLUSION

In conclusion, after significant updates to thresholds and hyperparameters, we have determined that logistic regression does not properly model this data due to inablilty for the coefficient's to converge. Potential models that woudl be better for this data set would include decision trees or any sort of gradient boosting.

5. CODE:

Attached in file CS2_CODE.ipynb