Problem Statement

A problem overview explaining your implemented baselines to examine problems that can be addressed by regression classification or clustering.

I focused on predicting a patient's time in the hospital. This feature is measured in days from when a patient is checked in to when they are checked out. I used variables such as the number of medications, lab procedures, diagnoses, and procedures to predict this in machine learning algorithms.

I also focused on predicting readmission rates. The readmitted column is measured as either NO, admitted in more than 30 days, or less than 30 days.

Knowing the suspected time a patient will stay in the hospital or when they will be readmitted is very advantageous because it can be used to optimize medical resources. Hospitals face challenges when allocating beds, staff, and equipment. Through the accurate prediction of each patient's stay based on previous medical history, resources can be more effectively distributed, leading to more efficient hospital practice.

1. Exploratory Data Analysis (EDA)

```
import pandas as pd

# Load the dataset
file_path = 'diabetic_data.csv'
df = pd.read_csv(file_path)

# Display basic info
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 101766 entries, 0 to 101765
Data columns (total 50 columns):
 #   Column                    Non-Null Count   Dtype 
---  ------                    --------------   ----- 
 0   encounter_id              101766 non-null  int64 
 1   patient_nbr               101766 non-null  int64 
 2   race                      101766 non-null  object
 3   gender                    101766 non-null  object
 4   age                       101766 non-null  object
 5   weight                    101766 non-null  object
 6   admission_type_id         101766 non-null  int64 
 7   discharge_disposition_id  101766 non-null  int64 
 8   admission_source_id       101766 non-null  int64 
 9   time_in_hospital          101766 non-null  int64 
 10  payer_code                101766 non-null  object
 11  medical_specialty         101766 non-null  object
 12  num_lab_procedures        101766 non-null  int64 
 13  num_procedures            101766 non-null  int64 
 14  num_medications           101766 non-null  int64 
 15  number_outpatient         101766 non-null  int64 
 16  number_emergency          101766 non-null  int64 
 17  number_inpatient          101766 non-null  int64 
 18  diag_1                    101766 non-null  object
 19  diag_2                    101766 non-null  object
...
 49  readmitted                101766 non-null  object
dtypes: int64(13), object(37)
memory usage: 38.8+ MB
None
```

At first glance, there are 101,766 entries with 50 different features. This is a large data set consisting of numerical and categorical data. There are no null data points. I started to look at specific examples of the data. Specifically, the data contains such attributes as patient number, race, gender, age, admission type, time in hospital, medical specialty of admitting physician, number of lab tests performed, HbA1c test result, diagnosis, number of

medications, diabetic medications, number of outpatient, inpatient, and emergency visits in the year before the hospitalization, etc. There are no null values.

There is a lot of data and measurements with a lot of data missing. This is seen in the data as a question mark. However, the column with boolean values measuring if a person has diabetes is full.

The readmitted column is measured as either NO or admitted more than 30 or less than 30. A lot of the data on the payer code and weight are missing. These data points cannot be used.

Hypothesis: It would be interesting to see how the sex and age of a person change the likelihood of the prevalence of diabetes. I believe that HbA1c will have one of the most direct effects on the prevalence of diabetes. Insulin levels are also a direct comparison.

Potential relevance to hospital business metrics:

Increase: time in hospital, specialty doctors, procedures done, number of medications readmitted

To increase the cost; decrease patient risk, increase specialties and expertise in specific dimensions,

To reduce risk and pay from patients: reduce admittance and identify factors that cause readmittance such as hba1c to predict readmittance

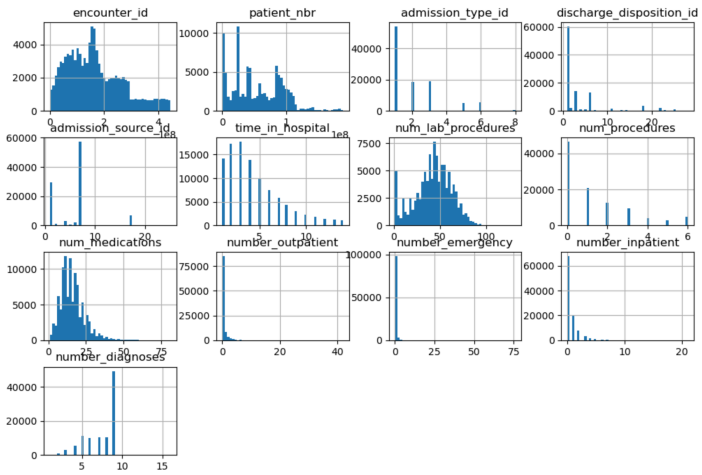
Missing data: The hospital does not monitor weight.

Diag_1- The primary diagnosis (coded as first three digits of ICD 9); 848 distinct values.

Values and levels are only tracked with steady, up, or down which is not very specific

Medical specialty is missing a lot of values.

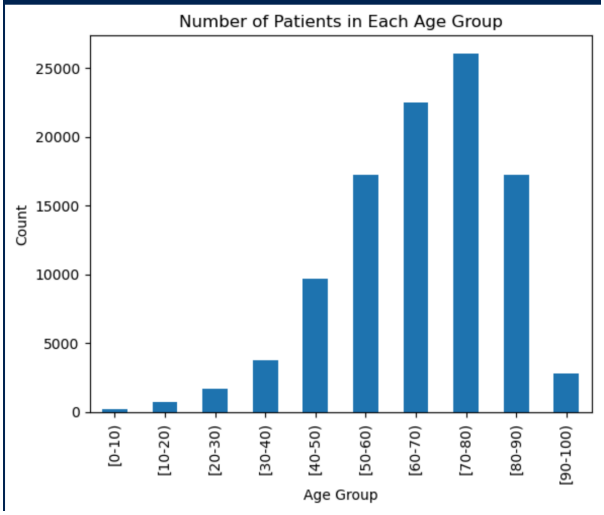
I used df.value_counts() to view example data points. I noticed some things right away that I will have to clean, but for now, I wanted to continue with EDA.



I used Matplotlib's .hist() to create basic histograms of some of the features in the data. From a first glance at this visualization, I saw that a large part of the data was skewed left, for example, encounter id, time in hospital, number of lab procedures, and number of medications. I thought that there might be some correlation between these variables and that I would discover more in the future. I also could see some features that didn't seem particularly useful such as number outpatient, number emergency, and possibly some others with stronger analysis.

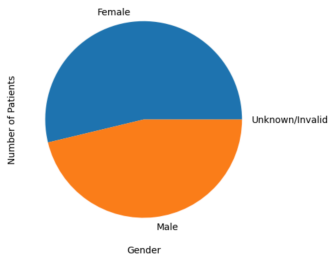
I wanted to discover the age column, and most of the data set was seniors. This was surprising to me at first. However, this makes sense because older people are much more susceptible to diabetes due to slower metabolism, reduced insulin sensitivity, and accumulative effects. Knowing this, the features that I am focusing on such as time in the hospital, number of medications, lab procedures,

```
data['age'].value_counts().sort_index().plot(kind='bar')
plt.title('Number of Patients in Each Age Group')
plt.xlabel('Age Group')
plt.ylabel('Count')
plt.show()
```



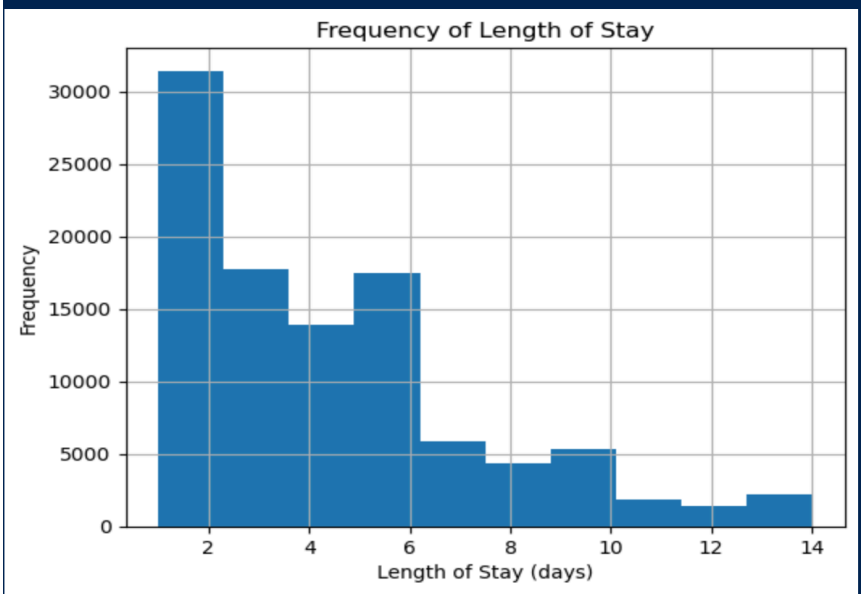
diagnoses, and procedures will be at higher rates compared to normal age distribution. This is because these people do not heal as quickly and have more healthcare experience.

```
data['gender'].value_counts().plot(kind='pie', color='purple')
plt.title('Gender Distribution')
plt.xlabel('Gender')
plt.ylabel('Number of Patients')
plt.show()
```



I looked at the distribution of gender. These were at normal levels, however, there are some unknown/invalid data points.

```
data['time_in_hospital'].hist(bins=10)
plt.title('Frequency of Length of Stay')
plt.xlabel('Length of Stay (days)')
plt.ylabel('Frequency')
plt.show()
```

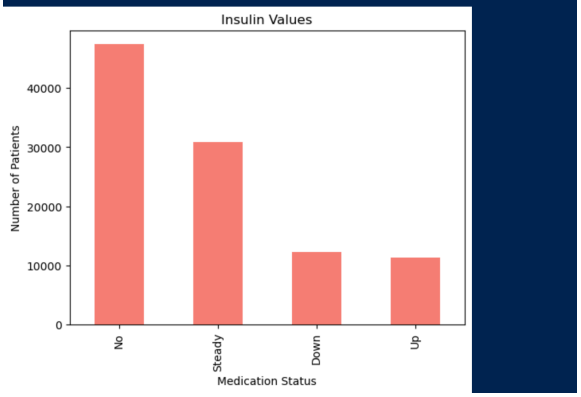


Next, I decided to examine time in the hospital. This is because I thought it was one of the key variables, and wanted to examine the correlations I saw in the hist() visual. Most of the data was within six days, with the most common being one, and there are some outliers. A person's length of stay is an important indicator of the severity of the condition, as well as the effectiveness of treatments compared to other patients.

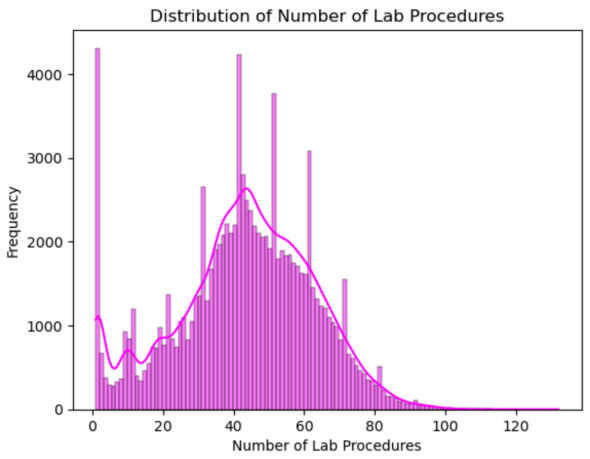
I then continued to look at the specific medication measurements. These measurements were only taken as no, steady, down, and up. A lot of these medications did not have strong data to be useful in analysis. One of the only medications that could be useful in further inspection was insulin values, which could be very useful as a feature in predicting something like diabetes.

```
key_meds = ['insulin', 'repaglinide', 'nateglinide', 'chlorpropamide', 'glimepiride',
            'acetohexamide', 'glipizide', 'glyburide', 'tolbutamide', 'pioglitazone',
            'rosiglitazone', 'acarbose', 'miglitol', 'troglitazone', 'tolazamide',
            'examide', 'citoglipton', 'metformin', 'glyburide-metformin', 'glipizide-metformin',
            'glimepiride-pioglitazone', 'metformin-rosiglitazone', 'metformin-pioglitazone']
allowed_values = ['Down', 'No', 'Steady', 'Up']

for med in key_meds:
    filtered_counts = data[med][data[med].isin(allowed_values)].value_counts()
    filtered_counts.plot(kind='bar', title=f"{med.capitalize()} Values", color='salmon')
    plt.xlabel('Medication Status')
    plt.ylabel('Number of Patients')
    plt.show()
```

✓ 6.2s




```
sns.histplot(data['num_lab_procedures'], kde=True, color='magenta')
plt.title('Distribution of Number of Lab Procedures')
plt.xlabel('Number of Lab Procedures')
plt.ylabel('Frequency')
plt.show()
```




```
average_lab_procedures = df['num_lab_procedures'].sum() / len(df['num_lab_procedures'])
print(f"Average number of lab procedures: {average_lab_procedures}")
```

✓ 0.0s

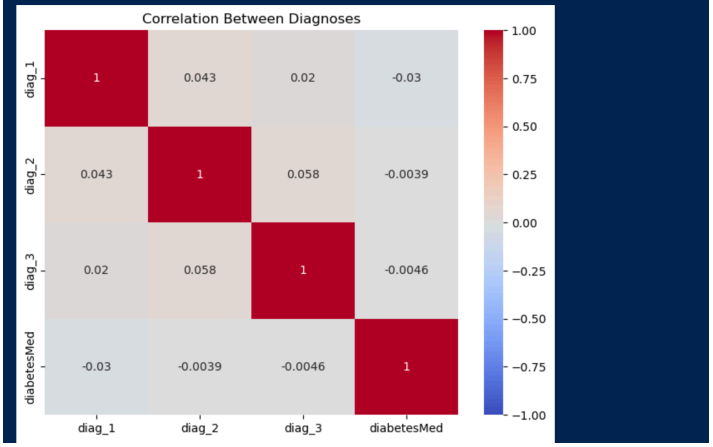
```
Average number of lab procedures: 43.09564098028811
```

The average number of lab procedures being 40 seemed very high to me (I calculated it below the graph). However, when dealing with patients with diabetes it is common to continually administer blood and urine tests. The number of procedures probably directly correlates with the many variables such as time in the hospital and insulin levels because a person is much more likely to have negative health levels if they are required to be tested frequently.

I wanted to inspect the diagnoses or if they received diabetes medication to see how similar these diagnoses are. Based on the correlation graph, these diagnoses are not very similar so it is unlikely that one patient has several of these diagnoses, or that they are related. Further inspection of the metadata would be useful to know what these specifically are.

```
df_encoded = df[['diag_1', 'diag_2', 'diag_3', 'diabetesMed']].apply(lambda x: x.astype('category').cat.codes)

plt.figure(figsize=(8, 6))
sns.heatmap(df_encoded.corr(), annot=True, cmap='coolwarm', vmin=-1, vmax=1)
plt.title('Correlation Between Diagnoses')
plt.show()
```

✓ 1.3s




```
sns.countplot(data=data, x='readmitted', palette='pastel')
plt.title('Count Plot of Readmission Status')
plt.xlabel('Readmitted')
plt.ylabel('Number of Patients')
plt.show()
```

✓ 0.9s

```
/var/folders/5f/4ph3f2jj253bcxmy2h_4qdym0000gn/T/ipykernel_36064/1095067521.py:1: F

assing `palette` without assigning `hue` is deprecated and will be removed in v0.1

 sns.countplot(data=data, x='readmitted', palette='pastel')
```

Lastly, I wanted to inspect the readmission rate. This is an important variable because if a patient is readmitted it is because they need more health care. A patient who is readmitted will see higher numbers in almost every category making it a key variable. Based on the graph, most patients are not readmitted but if they are, most are readmitted within 30 days.

Overall, performing EDA on this data set was very useful in understanding more about the variables within, and gave me insight into the type of data cleaning I will begin to implement.

1. Data Processing

Defining why missing values are detected, and why importing metadata is important, including justifications for using each visualization

Data cleaning-

1. First I dropped all duplicates

```
# Check for duplicate rows
#<add your code here>

print(f"Number of unique values in title column: ", df.nunique())

# Drop duplicates if any
df.drop_duplicates()
```

2. I replaced '?' with nan to begin to look at the null values. I found this data issue while performing EDA. I then found the columns with large amounts of missing values, and dropped them. After diag_1 no feature contains missing values

```
df.replace('?', np.nan, inplace=True)
```

```
# Calculate missing values in percentage
missing_percentage = df.isnull().mean() * 100

# Sort the percentages in descending order
missing_percentage_sorted = missing_percentage.sort_values(ascending=False)

# Display the sorted missing values
print("Missing values (in percentage) for each column:\n", missing_percentage_sorted)
```

```
Missing values (in percentage) for each column:
 weight                    96.858479
max_glu_serum              94.746772
A1Cresult                  83.277322
medical_specialty          49.082208
payer_code                 39.557416
race                        2.233555
diag_3                      1.398306
diag_2                      0.351787
diag_1                      0.020636
```

```
df = df.drop(columns=['patient_nbr', 'encounter_id', 'weight', 'payer_code', 'max_glu_serum', 'A1Cresult', 'medical_specialty'])
```

3. I changed the age data to map the age range to just the midpoint changing it to an integer.

```
# Mapping age ranges to midpoints
age_mapping = {
    '[0-10)': 5, '[10-20)': 15, '[20-30)': 25, '[30-40)': 35, '[40-50)': 45,
    '[50-60)': 55, '[60-70)': 65, '[70-80)': 75, '[80-90)': 85, '[90-100)': 95
}

df['age'] = df['age'].replace(age_mapping).astype(int)
print(df['age'].dtype)  # Should now be int
```

4. The diag 1, 2, and 3 had numerical values, but for some reason the data was categorical. I simply changed this to numeric data.

```
diag_cols = ['diag_1', 'diag_2', 'diag_3']
df[diag_cols] = df[diag_cols].apply(pd.to_numeric, errors='coerce')
```

5. I replaced the other data point within the race with unknown because this is much more appropriate wording. Knowing that there are null values within race from step one, I replaced these values with unknown and one hot encoded the feature.

```
df['race'] = df['race'].replace('Other', 'Unknown')
print(df['race'].isnull().sum())
df['race'] = df['race'].fillna('Unknown')
```

```
2273
```

```
df = pd.get_dummies(df, columns=['race'], drop_first=True)
```

6. I replaced all remaining null values with the median because I had not dealt with outliers so median was a more accurate way of dealing with those null values.

```
df = df.fillna(df.median())
```

7. I then encoded the boolean features to either 0 or 1, checking to see if there were any remaining categorical columns. There weren't any so I saved the cleaned data set and was ready for model implementation.

```
df = pd.get_dummies(df, columns=['race'], drop_first=True)

df['readmitted'] = df['readmitted'].map({'No': 0, 'Yes': 1})

df['gender'] = df['gender'].map({'Male': 1, 'Female': 0})

df['change'] = df['change'].map({'Ch': 1, 'No': 0})

df['diabetesMed'] = df['diabetesMed'].map({'Yes': 1, 'No': 0})

non_numeric_cols = df.select_dtypes(include=['object']).columns
print("Non-numeric columns:", non_numeric_cols)
```

```
Non-numeric columns: Index([], dtype='object')
```

```
df.to_csv('regression_diabetic_data.csv', index=False)
```

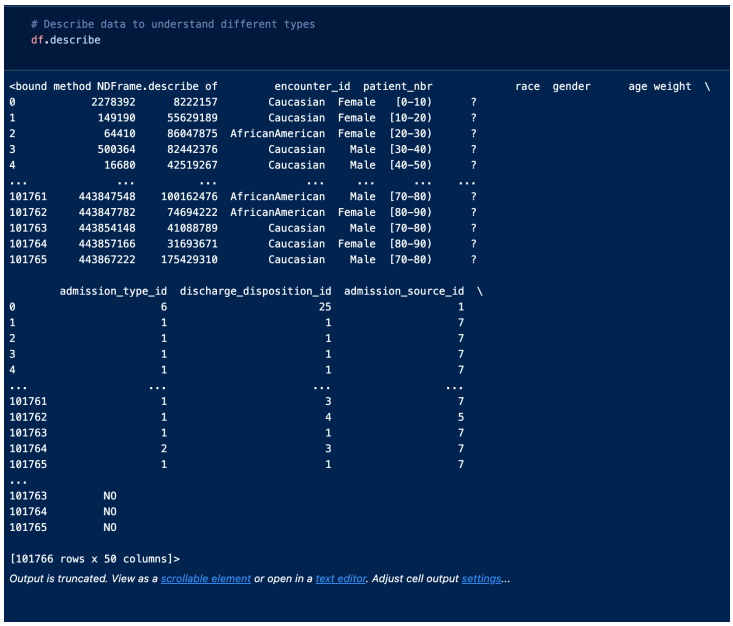
Metadata

```
META DATA
```

```
#meta data mapping
admission_type_mapping = {
    1: 'Emergency', 2: 'Urgent', 3: 'Elective', 4: 'Newborn', 5: 'Trauma Center',
    6: 'Not Available', 7: 'Unknown', 8: 'Other'
}

discharge_disposition_mapping = {
    1: 'Discharged to home', 2: 'Discharged to another short-term hospital',
    3: 'Discharged to SNF', 4: 'Discharged to ICF', 5: 'Discharged to another facility',
    6: 'Discharged to home under care', 7: 'Left AMA', 8: 'Discharged to home with IV',
    9: 'Admitted as inpatient to this hospital', 10: 'Neonate discharged to another hospital',
    11: 'Expired', 12: 'Discharged to psych hospital', 13: 'Discharged to rehab',
    14: 'Discharged to long-term care', 15: 'Discharged to another hospital outpatient facility',
    16: 'Unknown', 17: 'Discharged to federally qualified health center'
}

admission_source_mapping = {
    1: 'Physician Referral', 2: 'Clinic Referral', 3: 'HMO Referral', 4: 'Transfer from hospital',
    5: 'Transfer from SNF', 6: 'Transfer from another facility', 7: 'Emergency Room',
    8: 'Court/Law Enforcement', 9: 'Not Available', 10: 'Transfer from rehab',
    11: 'Unknown', 12: 'Other'
}

df['admission_type_id'] = df['admission_type_id'].map(admission_type_mapping)
df['discharge_disposition_id'] = df['discharge_disposition_id'].map(discharge_disposition_mapping)
df['admission_source_id'] = df['admission_source_id'].map(admission_source_mapping)
```

While it was important to make all data for the implementation of machine learning algorithms, I decided to go through the database codebook and map data points to their respective definitions. This will be useful in visualizations, and in understanding the data, drawing

meaning from the data more accurately.



```
# Describe data to understand different types
df.describe
```

```
<bound method NDFrame.describe of         encounter_id  patient_nbr            race  gender      age weight  \
0            2278392      8222157       Caucasian  Female   [0-10)      ?
1             149190     55629189       Caucasian  Female  [10-20)      ?
2              64410     86047875  AfricanAmerican  Female  [20-30)      ?
3             500364     82442376       Caucasian    Male  [30-40)      ?
4              16680     42519267       Caucasian    Male  [40-50)      ?
...              ...          ...             ...     ...     ...    ...
101761     443847548    100162476  AfricanAmerican    Male  [70-80)      ?
101762     443847782     74694222  AfricanAmerican  Female  [80-90)      ?
101763     443854148     41088789       Caucasian    Male  [70-80)      ?
101764     443857166     31693671       Caucasian  Female  [80-90)      ?
101765     443867222    175429310       Caucasian    Male  [70-80)      ?

        admission_type_id  discharge_disposition_id  admission_source_id  \
0                       6                        25                    1
1                       1                         1                    7
2                       1                         1                    7
3                       1                         1                    7
4                       1                         1                    7
...                   ...                       ...                  ...
101761                  1                         3                    7
101762                  1                         4                    5
101763                  1                         1                    7
101764                  2                         3                    7
101765                  1                         1                    7
...
101763         NO
101764         NO
101765         NO

[101766 rows x 50 columns]>
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

Metadata analysis notes from 11/28:

“The dataset represents ten years (1999-2008) of clinical care at 130 US hospitals and integrated delivery networks. It includes over 50 features representing patient and hospital outcomes. Information was extracted from the database for encounters that satisfied the following criteria. (1) It is an inpatient encounter (a hospital admission). (2) It is a diabetic encounter, that is, one during which any kind of diabetes was entered into the system as a diagnosis. (3) The length of stay was at least 1 day and at most 14 days. (4) Laboratory tests were performed during the encounter. (5) Medications were administered during the encounter”(BMC Health Services Research, 2014) Specifically, using the BMC health services I began to derive some potential business objectives I could derive from the data. The conclusions drawn from BMC illustrated the importance of some variables to me specifically HbA1c. HbA1c measures glucose levels in the blood and is directly related to the presence of diabetes. Due to missing data, however, this statistic would be very difficult to predict using machine learning algorithms.

Due to the missing values, I instead decided to focus on readmission rates and time in hospital measures. This is because predicting these values is very useful in hospital optimization,

measuring quality of care, and predicting hospital costs. These are all vital predictions that can create the most efficient hospitals, providing care to the most people.

1. Machine Learning Models

A model overview explaining the process of creating training and evaluating the implemented regression, classification, and clustering models and their iterations based on feature selections and fine-tuning the parameters.

- Regression Models

To begin, regression models predict a numerical continuous variable based on variables. There can be several different variables used that can assist in this prediction. Due to the fact that a numerical value is predicted, essential features are correctly scaled to ensure data is fairly incorporated. Specific algorithms such as linear regression, decision trees, and ensemble methods are used to solve these tasks, each effective based on data and business objectives.

Regression problem statements-

What is the most effective model that accurately forecasts a patient's stay based on clinical variables?

What can these forecasts predict?

What is an example patient and what do the results of the model show?

Feature Selection-

After performing EDA and metadata analysis there were some features that stood out to me. Primarily there was time in the hospital. This was a particularly interesting feature because being

able to predict a patient's time they will spend in the hospital can be very useful. Because this is a consistent numeric value with a strong correlation to other features.

```
Top correlations with target variable:
time_in_hospital              1.000000
num_medications               0.466135
num_lab_procedures            0.318450
number_diagnoses              0.220186
num_procedures                0.191472
```

These correlations were among the strongest observed in the dataset, which is why I chose to focus on these four features. The relationships between these variables made intuitive sense. For instance, a patient who has received more medications, undergone more procedures, and has been diagnosed with more conditions is likely to have spent more time in the hospital. This is because the administration of medications, procedures, and diagnoses inherently requires time. Any interaction with a patient in a hospital setting takes time, which is reflected in these variables.

I selected time in hospital as the target variable for linear regression not only because it is a consistent numeric value and has strong correlations with other variables, but also because it is an incredibly valuable metric for hospitals to predict. Specifically, predicting time in hospitals can enhance hospital efficiency. If, upon admission, a hospital can predict the accurate duration of a patient’s stay, it can better allocate resources. Hospitals operate with finite resources which they can optimize through this metric.

Additionally, this ability to predict patient stay durations benefits both patients and the hospital. Patients can receive faster and more efficient care and predictions can aid in cost planning. The time a patient stays in the hospital is directly correlated to the amount of money spent, allowing for the hospital to better manage and allocate these resources as well.

3.1.2: Start to implement Linear Regression as a baseline model

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error

# Target Value drop to avoid bias in data
X = df.drop(columns=(['num_medications', 'num_lab_procedures', 'number_diagnoses', 'num_procedures']))
y = df['time_in_hospital']

# Split Data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Feature Scaling (Standardization) could also use mid max scaling but range would be difficult in this data set
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

#  Linear Regression model
linear_model = LinearRegression()
linear_model.fit(X_train_scaled, y_train)

# Model Evaluation
y_pred = linear_model.predict(X_test_scaled)

mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)

print("Linear Regression Model Performance:")
print(f"Mean Absolute Error (MAE): {mae}")
print(f"Mean Squared Error (MSE): {mse}")
```

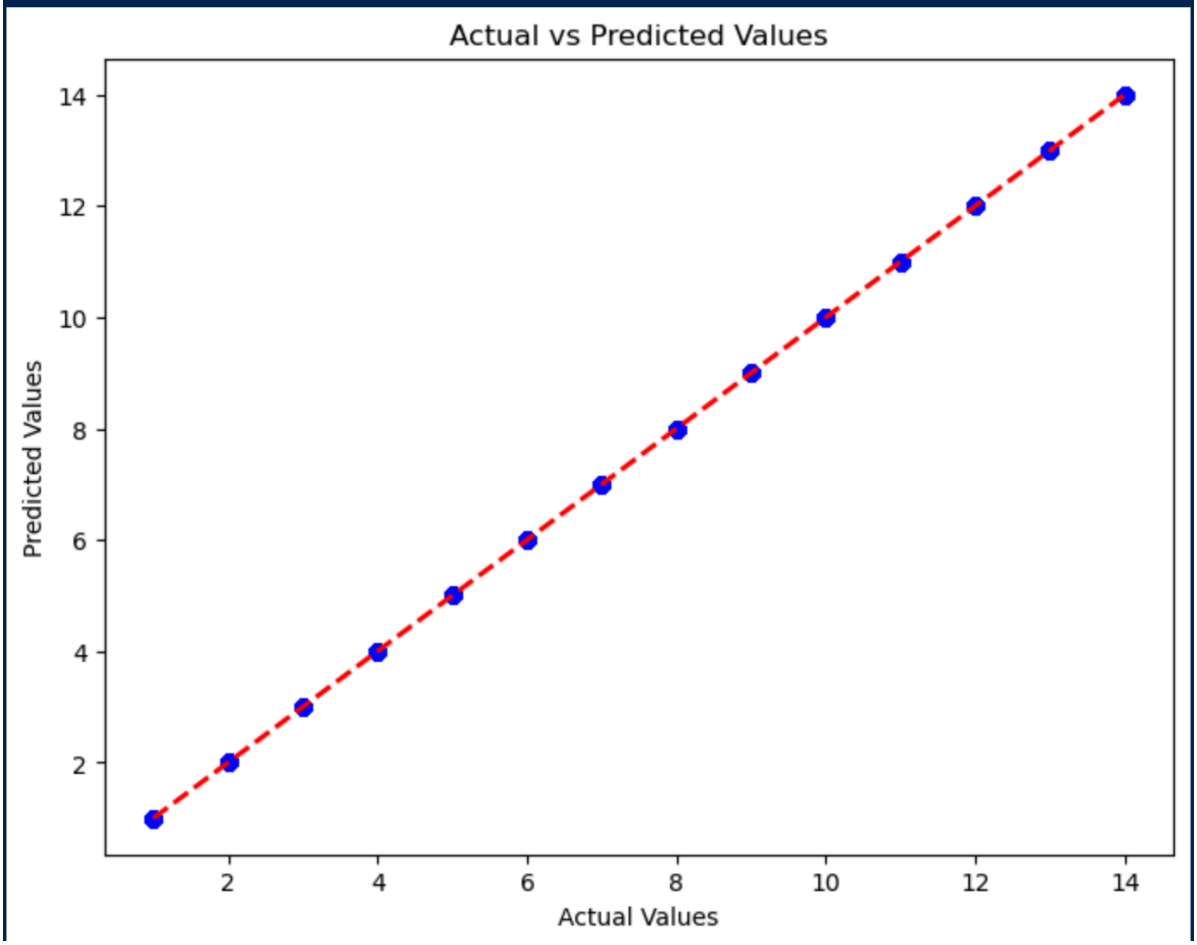
```
Linear Regression Model Performance:
Mean Absolute Error (MAE): 1.0822563571796515e-14
Mean Squared Error (MSE): 2.0753405714192698e-28
```

I implemented linear regression using 20% of the data as the test set, setting the test size equal to 0.2. Given the differences in the ranges of the features, it was essential to scale the data to ensure the model viewed all data fairly. I used StandardScaler over MinMaxScaler or

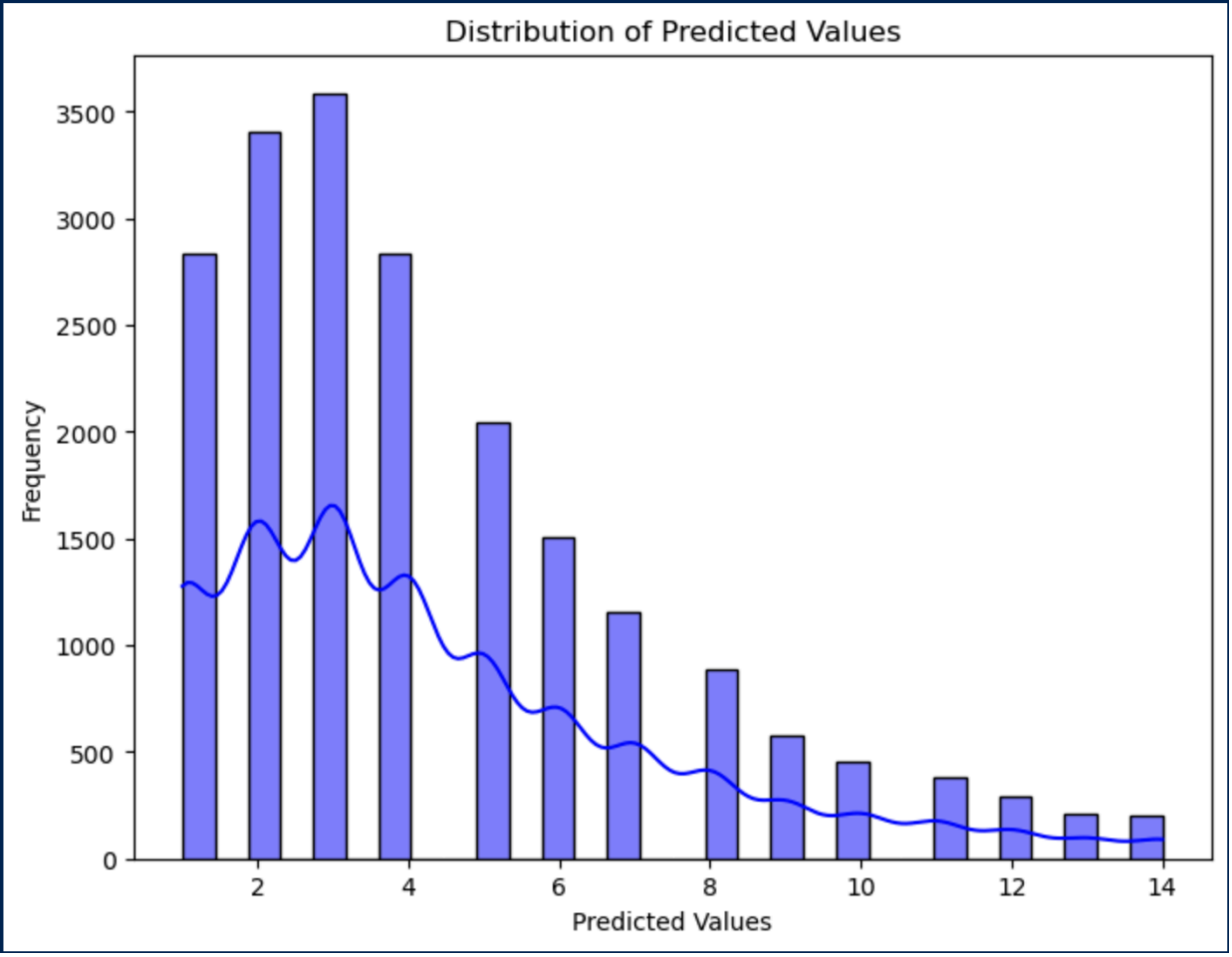
RobustScaler because the data was relatively normally distributed and positive. This was seen in the lack of significant outliers and negative values.

To ensure consistency across models and reproducibility of results, I set the random_state parameter to 42 in all implementations. This guaranteed that the same train-test splits were used throughout the modeling process.

```
# Plot actual vs predicted values
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, color='blue', alpha=0.5)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', lw=2)
plt.title('Actual vs Predicted Values')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.show()
```

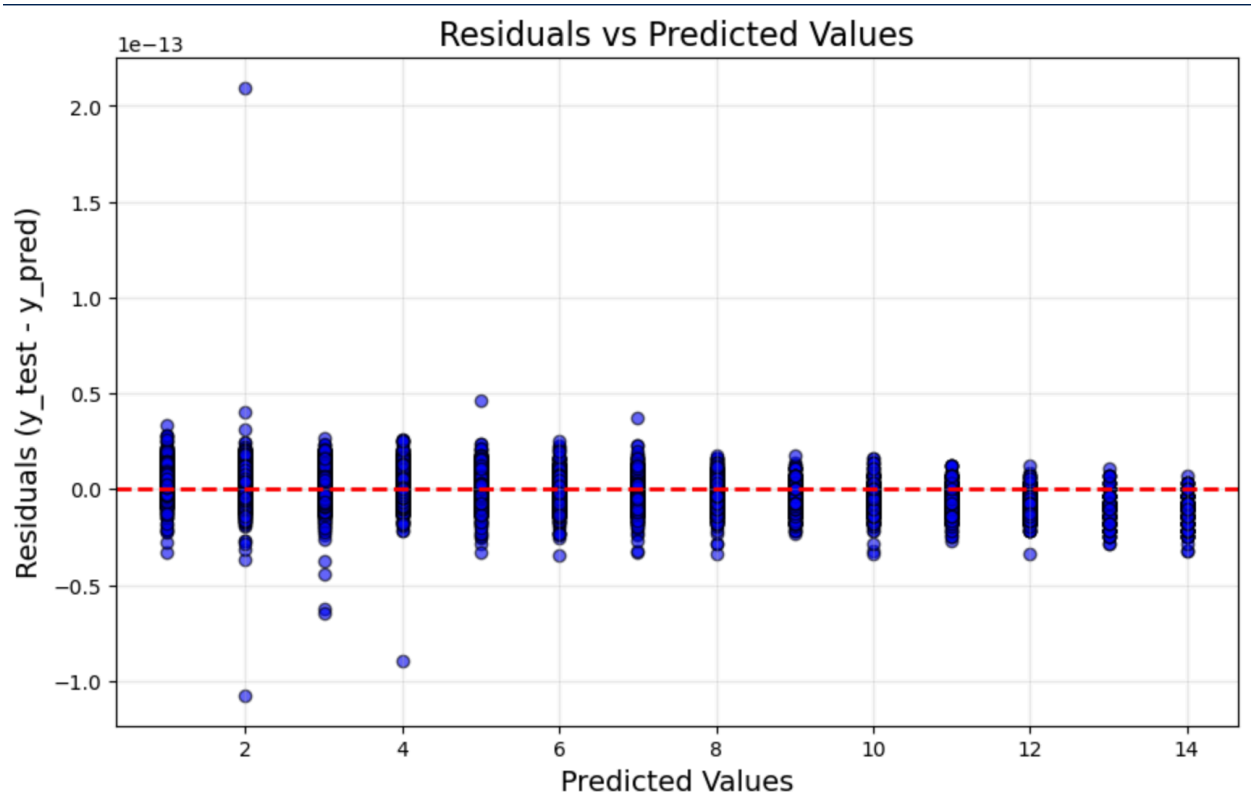


```python
# Plot distribution of predicted values
plt.figure(figsize=(8, 6))
sns.histplot(y_pred, kde=True, color='blue', bins=30)
plt.title('Distribution of Predicted Values')
plt.xlabel('Predicted Values')
plt.ylabel('Frequency')
plt.show()
```



This model was very accurate and overfit the training data. The predicted values were almost identical to the training data and the graph illustrating the predicted values was very similar to the time in-hospital visualization of the data set. I began to understand why the model was overfitting. The data was split correctly, the training data the model was run on was different

from the actual data set, and the features were correct and made sense with context.



As seen in the plot of the residuals vs predicted values there is a clear pattern illustrating the model overfitting the data, capturing noise and irrelevant features. I decided to increase the size of the testing set to increase the amount of data the model was training increasing bias.

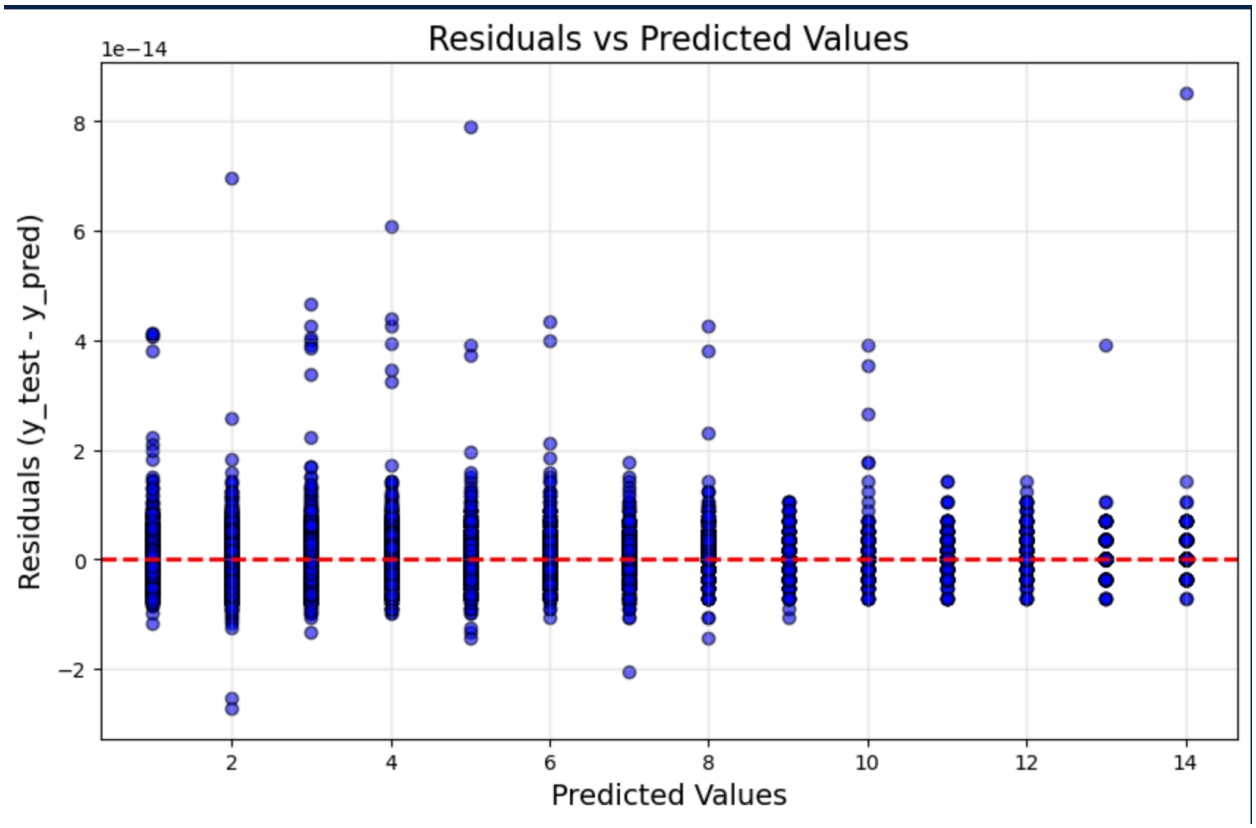
testsize=3

```
Linear Regression Model Performance:
Mean Absolute Error (MAE): 2.074433357194141e-15
Mean Squared Error (MSE): 9.991945292197043e-30
```

testsize=4

```
Linear Regression Model Performance:
Mean Absolute Error (MAE): 4.727929492569582e-15
Mean Squared Error (MSE): 4.0641796215195765e-29
```

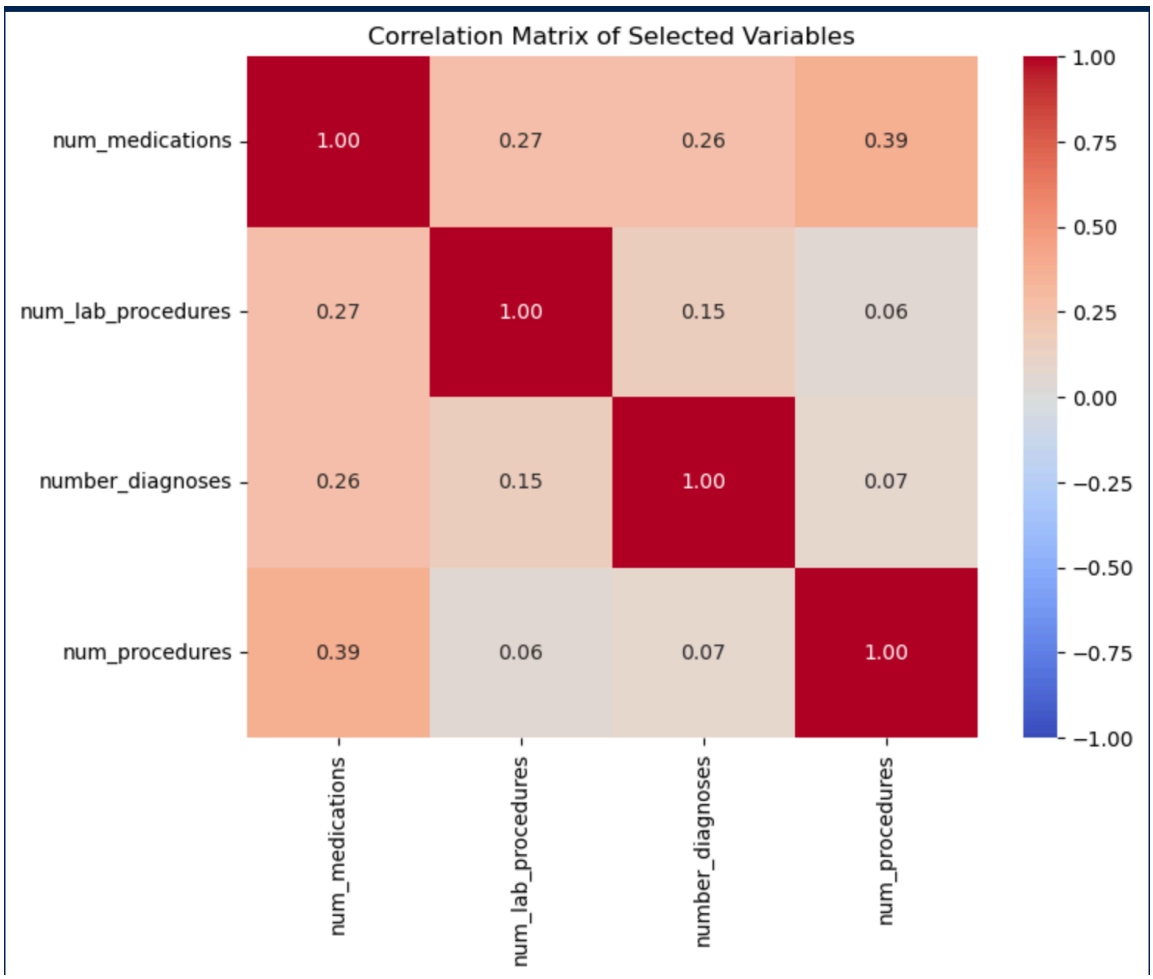
Setting the test size to 3 was the most effective, slightly increasing the variance in the model.



Due to the model still overfitting, I decided to implement regularized models such as Ridge, Lasso, and ElasticNet. These models add a penalty term to the loss function, which helps prevent the model from overfitting the training data, encouraging the model to generalize. Specifically, the implementation of ElasticNet worked best in increasing generalization.

ElasticNet combines the strengths of both Lasso (L1 regularization) and Ridge (L2 regularization), allowing for better control over the weight of the features. It can eliminate irrelevant features (increasing sparsity), while also ensuring that the model doesn't become overly sensitive to small changes or noise in the data (increasing stability).

In my case, I believed that the features used to predict the target variable were highly correlated, causing multicollinearity. This occurs when two or more predictor variables are highly correlated, making it difficult for the model to determine the weights of the features. As a result, the model's coefficients can become unstable, becoming overly sensitive to small changes in the data.



As seen in the visualization, num procedures and num medication had a very strong correlation, higher than all but one correlation between time in the hospital and its features.

Overall, ElasticNet was the best choice for addressing this issue because it balances both L1 and L2 regularization, which helps stabilize the model's coefficients while still allowing for feature selection.

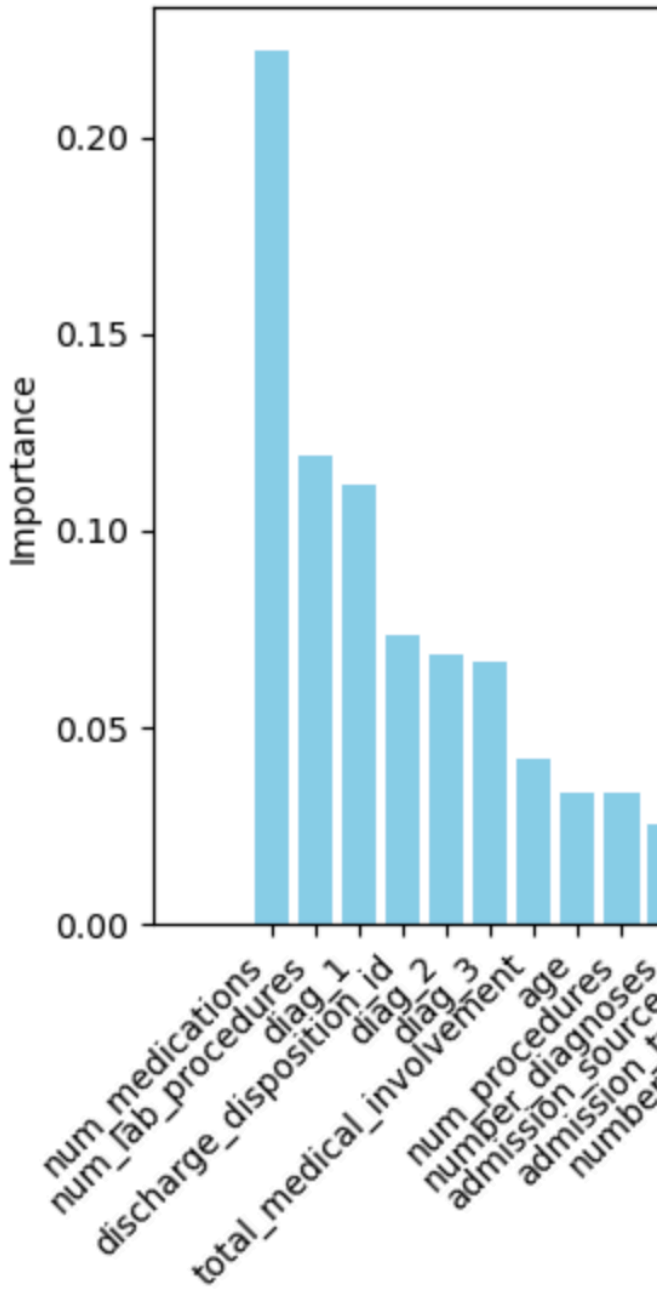
Implementation of ElasticNet

```
columns_to_sum = [
    'num_procedures', 'num_medications',
    'number_inpatient', 'number_outpatient', 'number_emergency'
]
# Create the total_medical_involvement column as the sum of the specified columns
df['total_medical_involvement'] = df[columns_to_sum].sum(axis=1)
```

First, I created a total_medical_involvement column by summing up features such as the number of procedures, number of medications, number of inpatient visits, number of outpatient visits, and number of emergency visits. These features together represent a patient's interaction with the hospital.

I intentionally did not include the time in the hospital in this feature, as I plan to use it as a target variable for prediction. Including it as a feature would have led to data leakage, where the target value is unintentionally present in the features, causing errors in the model.

I believe that this total_medical_involvement metric will be useful for understanding the extent of a patient's involvement with the hospital, providing a holistic view of their interactions.



I then used a Random Forest Regressor to illustrate feature importance in order to select the best features to train my ElasticNet model. A Random Forest Regressor evaluates feature importance by creating multiple decision trees, each trained on a random subset of the data with a random subset of features. During the training process, the model measures how much each feature helps to reduce the error in the model.

The feature importance is typically determined using metrics such as Gini impurity (for classification tasks) or Mean Squared Error (MSE) (for regression tasks). For example, a feature that results

in the largest reduction of these metrics when used to split the data will have a higher importance. This makes it a more influential feature for predicting the target value.

Based on the feature importance results of the random forest regressor model, I started to implement an ElasticNet model.

```
from sklearn.linear_model import ElasticNet
from sklearn.model_selection import GridSearchCV, KFold
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error, make_scorer
import numpy as np

# Initialize K-Fold
kf = KFold(n_splits=5, shuffle=True, random_state=42)

# Prepare data
X = df.drop(columns=(['num_medications', 'num_lab_procedures', 'number_diagnoses', 'num_procedures', 'total_medical_involvement'])).values
y = df['time_in_hospital'].values

# Define ElasticNet hyperparameter grid
elasticnet_grid = {
    'alpha': [0.01, 0.1, 1, 10],        # Regularization strength
    'l1_ratio': [0.1, 0.5, 0.9],        # Mix of L1 and L2 regularization
}

# Custom scoring function using mean absolute error
scorer = make_scorer(mean_absolute_error, greater_is_better=False)

# Initialize GridSearchCV
elasticnet = ElasticNet(random_state=42)
grid_search = GridSearchCV(estimator=elasticnet, param_grid=elasticnet_grid,
                           scoring=scorer, cv=kf, verbose=1)

# Apply feature scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Fit GridSearchCV
grid_search.fit(X_scaled, y)

# Best parameters and results
print("Best Parameters:", grid_search.best_params_)
print("Best Score (MAE):", -grid_search.best_score_)  # Negative because MAE scorer is negated
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
Best Parameters: {'alpha': 0.01, 'l1_ratio': 0.9}
Best Score (MAE): 0.009445487392410273
```

I used GridSearchCV and KFold to pick the best parameters and to split the data for cross-validation. GridSearchCV. To begin, in GridSearchCV the elasticnet_grid I set up sets a range of parameters that the algorithm runs KFold on, accessing MAE for each parameter

combination and choosing the best final hyperparameter tuning.

```
best_model = ElasticNet(alpha=0.01, l1_ratio=0.9, random_state=42)
best_model.fit(X_scaled, y)
```

```
ElasticNet
ElasticNet(alpha=0.01, l1_ratio=0.9, random_state=42)
```

```
from sklearn.model_selection import train_test_split

# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Scale the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)  # Fit and transform on training data
X_test_scaled = scaler.transform(X_test)    # Transform test data only
```

```
# Use the best parameters from GridSearchCV
best_params = grid_search.best_params_
best_model = ElasticNet(alpha=best_params['alpha'],
                        l1_ratio=best_params['l1_ratio'],
                        random_state=42)
best_model.fit(X_train_scaled, y_train)
```

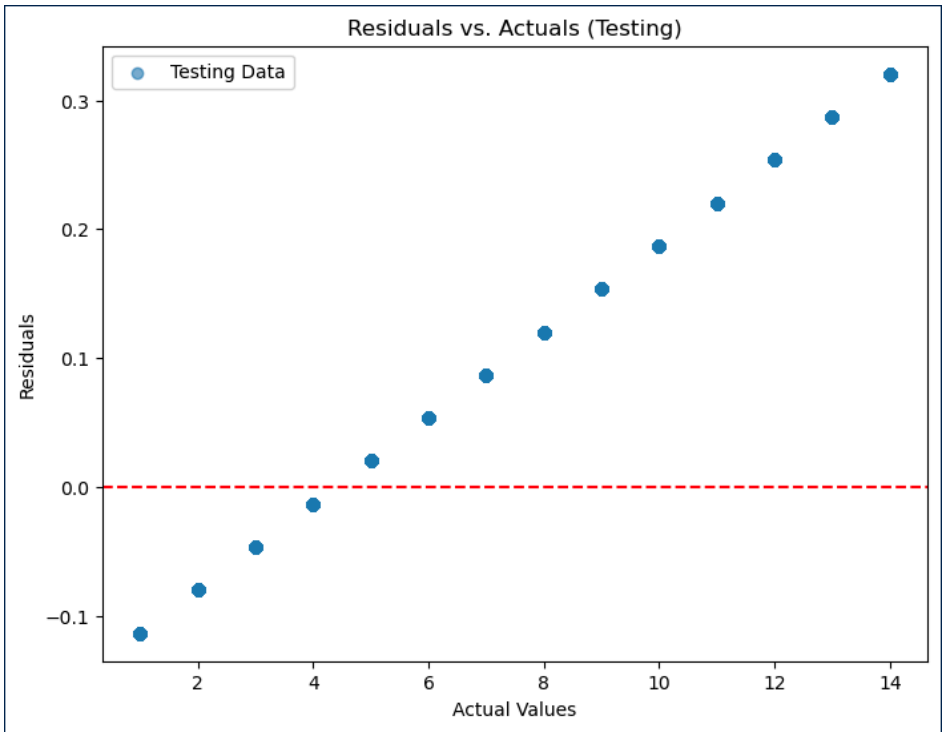
```
ElasticNet
ElasticNet(alpha=0.01, l1_ratio=0.9, random_state=42)
```

KFold is a type of cross-validation in which the model splits the data into n_splits folds. The model is trained on these K-1 folds and tested on the remaining folds. This process repeats how many times n_splits is set to. The model's performance is then averaged over the iterations to get a more accurate score on how well the algorithm applies to all data. Through maximizing the use of the data set the chances of overfitting are reduced.

Tuning the parameters

Specifically, I chose five-folds, moving to ten folds to reduce overfitting. When the model is trained on, the impact each iteration has on the final score. This creates a more regularized result based on the data. I then increased the parameters to include higher regularization strength to 100. I also increased the range for the L1_ratio, attempting to find a stronger balance between sparsity and stability with smaller values.

Model Selection Linear Regression

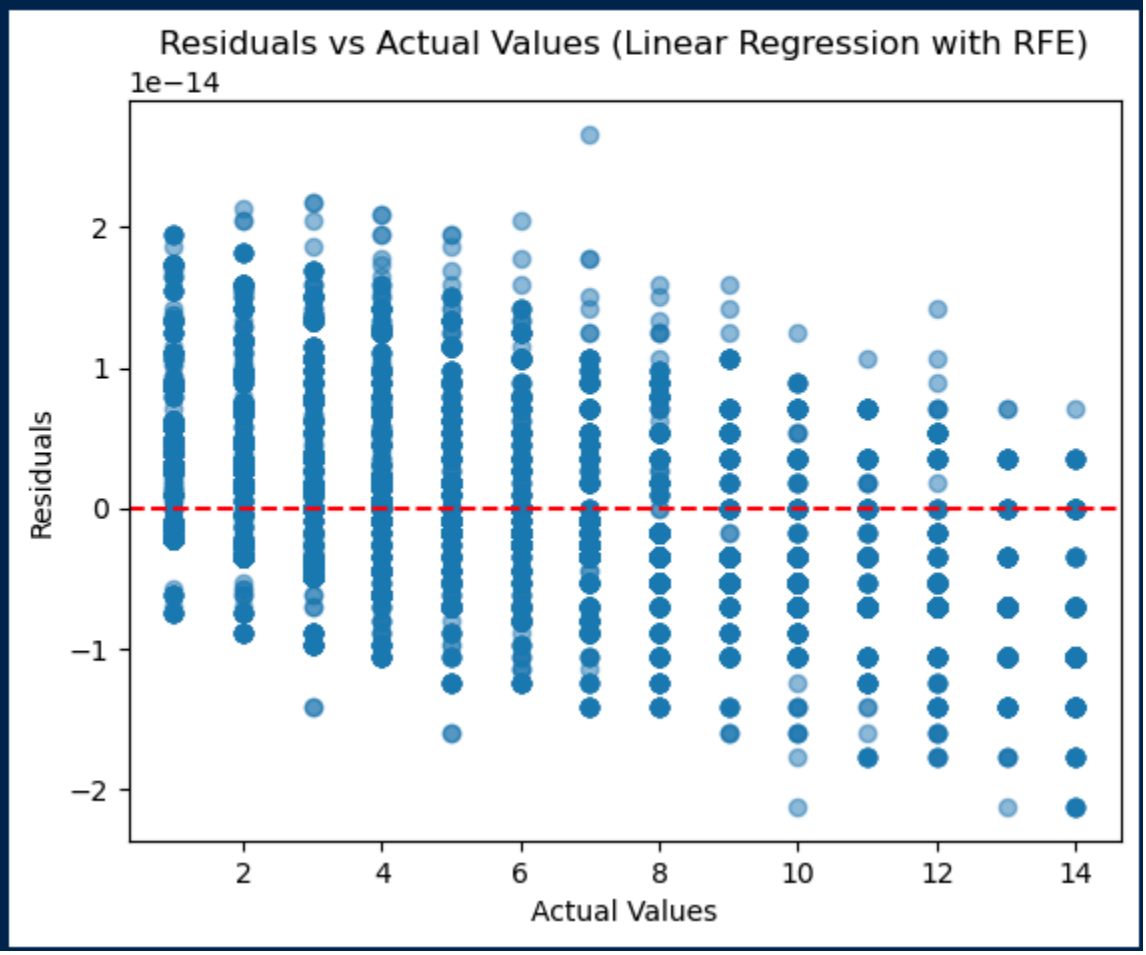



```
Test MAE: 0.0093
Test MSE: 0.0001
```

Unfortunately, initially, I chose to focus on this model to decrease overfitting, due to the complexity of the model overfitting increased. After continuous testing, the best model that predicted the most useful models I could create was my original Linear Regression Model. This model was simple and fit the data the best. I attempted models such as ridge regression, linear

regression with cross-validation, and feature selection using RFE (choosing the top 5 features) to reduce overfitting. (RFE selects these features by training the data on each feature, ranking its importance by its impact on the prediction).

```
Linear Regression Model Performance:
Cross-Validation MAE (Average): 4.752508244064836e-15
Mean Absolute Error (MAE): 3.290973055084745e-15
Mean Squared Error (MSE): 2.1488578809883366e-29
```



These models began to underfit the data slightly. In conclusion, the best model that fit the data the best and resulted in the most applicable predictions was the original linear regression model.

- Classification Models

Classification models are used to classify data into predefined categories. A classification model requires the target value to be categorical. Examples of classification models are logistic regression, random forest, and K nearest neighbor.

Logistic regression is used for binary or multi class classification tasks predicting the probability that an input belongs to a specific category. It uses the sigmoid function using logs to map probabilities between zero and one. A set threshold(usually 0.5) determines classification.

Random forests are an ensemble model built on random trees with unique subsets and features of the data. Each tree asks a question to separate the data, with the majority voting among trees determining the final prediction. This randomness is great for finding unique patterns in the data as well as feature selection.

KNN nearest neighbors is a non-parametric algorithm meaning it classifies data points based on distance between other points. This maps the data set and classifies based on location on the map.

Business Objective:

Being able to predict if a patient will be readmitted, specifically given a timeframe is very useful because this can associate certain practices that that patient received to having to be readmitted. This is very useful in optimizing hospital practices because knowing if someone will need to be readmitted will plan ahead to provide for this patient. Also, medical attention can be shown as ineffective due to this prediction, making sure the hospital gives helpful treatment, improving

patient care. This also can be used for hospital cost reduction by reducing unnecessary readmission, saving money on both ends.

Potential Use Cases:

Patients can be segmented based on their likelihood of being readmitted. Also, hospitals' efficiency can be benchmarked and compared for specific needs to be readmitted.

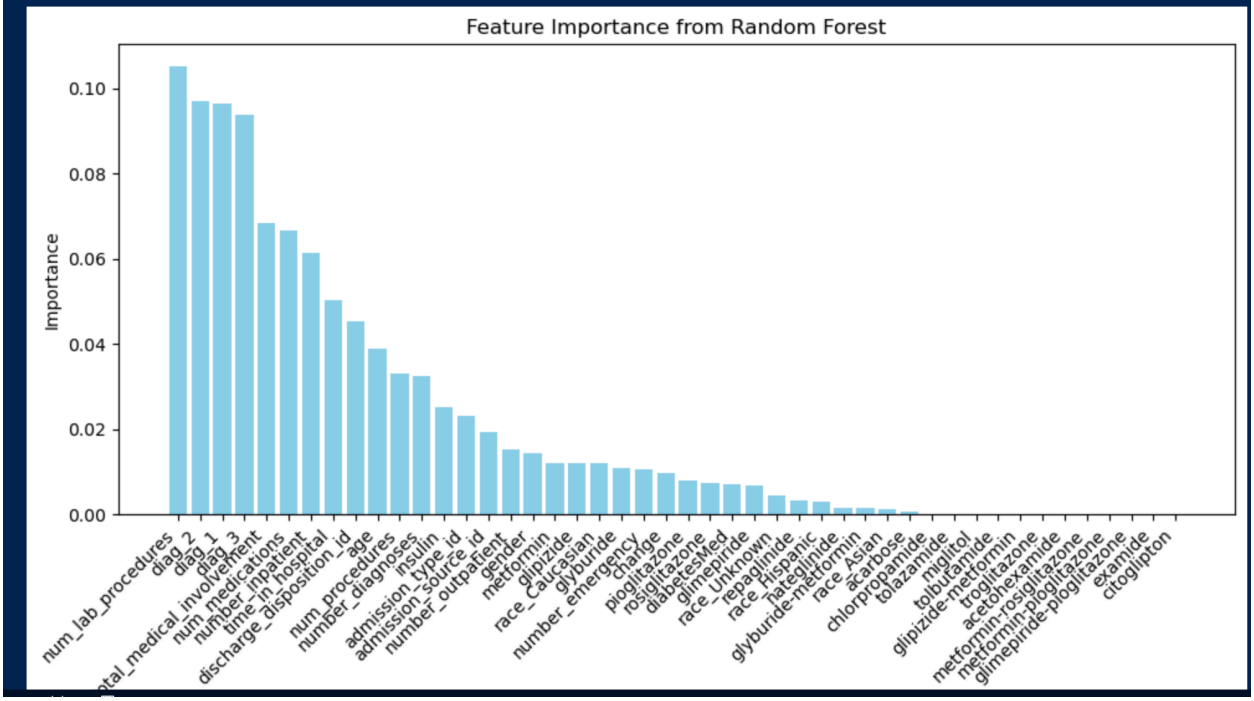
```
# Importing required sklearn packages
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split

label_encoder = LabelEncoder()
df['readmitted'] = label_encoder.fit_transform(df['readmitted'])

# Defining target and features
X = df[['age', 'time_in_hospital', 'admission_type_id', 'diag_1', 'diag_2', 'diag_3']]
y = df['readmitted']

# Standardizing the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Splitting data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# Print shapes to verify
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

✓ 0.2s

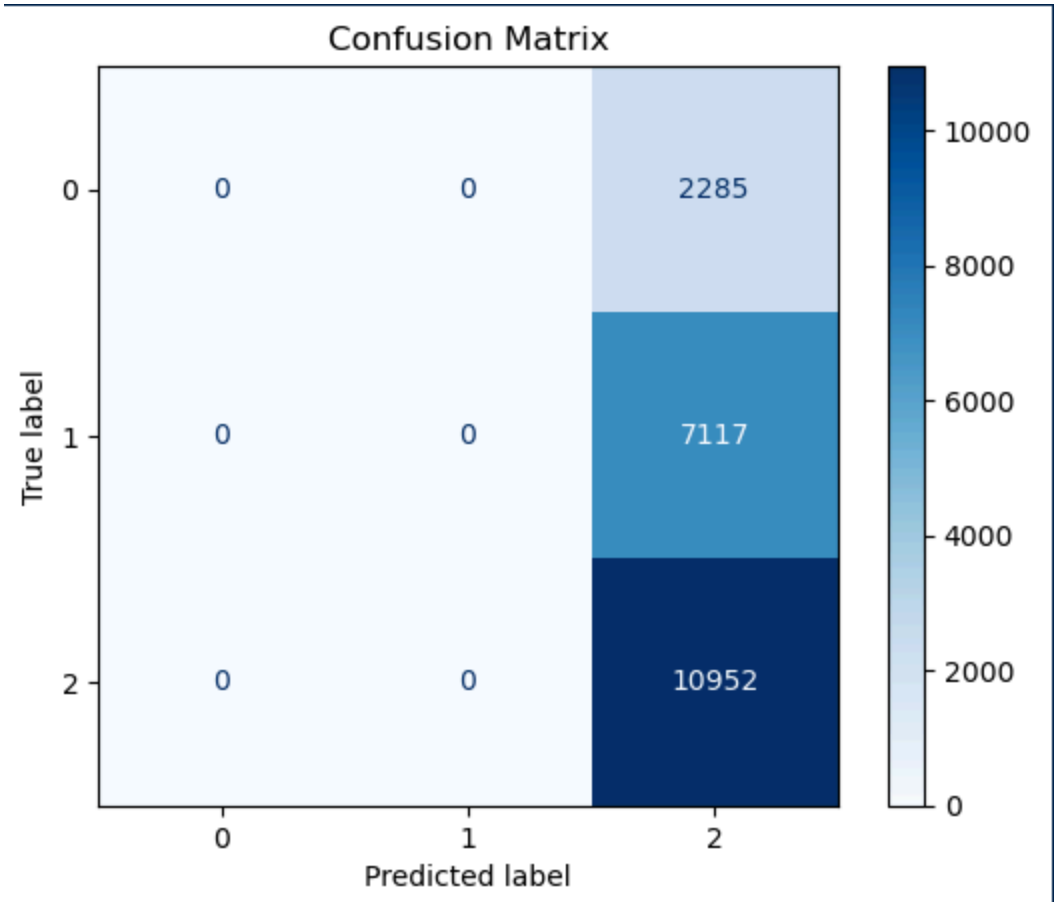
```
X_train shape: (81412, 6)
X_test shape: (20354, 6)
y_train shape: (81412,)
y_test shape: (20354,)
```

To begin, I set the data label encoding readmitted to make sure that the readmitted target value was categorical, and Label encoding was effective based on the data within the readmitted category. The readmitted column is measured as either NO or admitted more than 30 or less than 30. In picking the features, I used the same random forest regressor.

```python
importances = model.feature_importances_
feature_names = X.columns

# Create a DataFrame for visualization
importance_df = pd.DataFrame({
    'Feature': feature_names,
    'Importance': importances
}).sort_values(by='Importance', ascending=False)

# Plot feature importance
plt.figure(figsize=(10, 6))
plt.bar(importance_df['Feature'], importance_df['Importance'], color='skyblue')
plt.title('Feature Importance from Random Forest')
plt.xlabel('Features')
plt.ylabel('Importance')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```



Based on the results, I decided to change the features of the data set to num_lab_procedures, diag_1, diag_2, and diag_3. These variables make sense intuitively as to why they would be effective features to predict readmission. Specifically, diag_1 is the diagnosis that initially caused the patient to be admitted and diag_2 and diag_3 are subsequent diagnoses based on that initial diagnosis. These are likely to predict readmission because it is likely a person will have to be readmitted to experience multiple procedures and diagnoses tests.



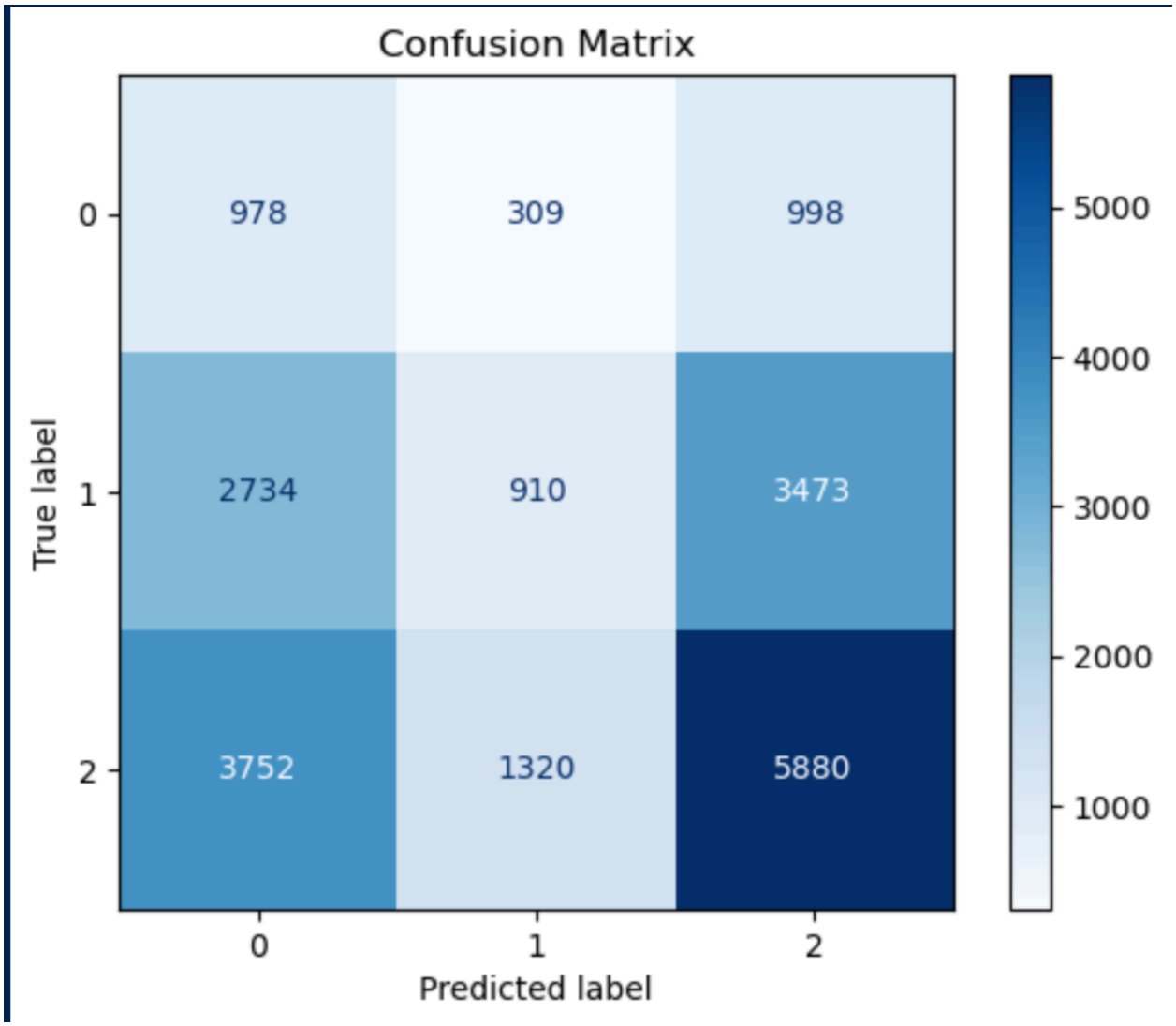
After first testing the model the confusing matric result showed that the model was misclassifying the data This is likely due to an imbalanced distribution of the data.

```
print(df['readmitted'].value_counts())
```

✓ 0.0s

```
readmitted
2    54864
1    35545
0    11357
Name: count, dtype: int64
```

I used.value_counts to see this distribution and it was clear that more than half of the data in the readmitted column belonged to one instance. To fix this issue, I set class_weight='balanced'. This sets the weights of the model inversely proportional to their frequency. So less frequent instances will have a higher weight, creating a balanced target feature.



This change created a much more accurate model. I applied cross-validation to the model, however it did not significantly impact the results.

```
from sklearn.model_selection import cross_val_score

# Cross-validation for Logistic Regression
from sklearn.linear_model import LogisticRegression

log_model = LogisticRegression(max_iter=1000, class_weight='balanced', random_state=42)

# Evaluate with 5-fold cross-validation
cv_scores = cross_val_score(log_model, X_train, y_train, cv=5, scoring='f1_macro')  # Use F1 for imbalanced data
print(f"Logistic Regression Cross-Validation F1 Score: {cv_scores.mean():.2f}")
```

✓ 1.7s

```
Logistic Regression Cross-Validation F1 Score: 0.29
```

I decided to focus on creating a strong Random Forest Classifier. This model was useful in the implementation of this data set to identify complex relationships in the data. Due to the model being highly dimensional, and having over 50 features, this implementation might identify important insights. Also, in my first implementation of the logistic regression, random forest classifier, and KNN models, the Random Forest Regressor earned the highest accuracy scores.

	Model	Accuracy	Precision	Recall	f1 Score
0	Logistic Regression	0.357325	0.352931	0.357351	0.302571
1	Random Forest	0.500295	0.377942	0.363918	0.357627
2	K-Nearest Neighbors	0.448806	0.355312	0.351966	0.349027

Through the implementation of the Random Forest Classifier, key risk factors can also be identified. Through random feature selection, the model finding important relationships can

illustrate important features being able to optimize specific treatment plans.

```python
# Importing required sklearn packages
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, classification_report
import matplotlib.pyplot as plt

# Initializing and training Random Forest Classifier
forest_clf = RandomForestClassifier(random_state=42)
forest_clf.fit(X_train, y_train)

# Predictions
predForest = forest_clf.predict(X_test)

# Previewing Predictions vs True Values
print("Predicted values (first 3):", predForest[:3])
print("True values (first 3):", y_test[:3].values)

# Evaluation
accuracy_rf = accuracy_score(y_test, predForest)
f1_rf = f1_score(y_test, predForest, average='macro')  # Use 'macro' for multiclass
precision_rf = precision_score(y_test, predForest, average='macro')
recall_rf = recall_score(y_test, predForest, average='macro')

print("Random Forest Metrics:")
print(f"Accuracy: {accuracy_rf:.2f}")
print(f"Precision: {precision_rf:.2f}")
print(f"Recall: {recall_rf:.2f}")
print(f"F1 Score: {f1_rf:.2f}")
print("\nClassification Report:\n", classification_report(y_test, predForest))

# Example code for encoding categorical variables if needed
# from sklearn.preprocessing import LabelEncoder
# le_sex = LabelEncoder()
# le_sex.fit(['F', 'M'])
# df['sex_encoded'] = le_sex.transform(df['sex'])
```

✓ 29.3s

```
Predicted values (first 3): [2 2 2]
True values (first 3): [2 2 2]
Random Forest Metrics:
Accuracy: 0.50
Precision: 0.38
Recall: 0.36
F1 Score: 0.36
```

This was my first model implementation, I added several parameters to improve the model. I started by setting class_weight='balanced' to handle the imbalanced data, n_estimators=100 to increase the number of trees without significantly increasing run time. I further increased n_estimators=200 and this improved performance even more. I also set max_depth=10. This reduces overfitting because it limits the amount of splitting the model can do based on the data.

This model could be further improved through GridSearchCV finding the best parameter combination and adding cross-validation.

```
# Importing required sklearn packages
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, classification_report
from sklearn.preprocessing import StandardScaler

# Standardizing the features for KNN
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Initializing and training the KNN model
k = 4
neigh6 = KNeighborsClassifier(n_neighbors=k)
neigh6.fit(X_train_scaled, y_train)

# Predictions
yhat6 = neigh6.predict(X_test_scaled)

# Evaluation Metrics
accuracy_knn = accuracy_score(y_test, yhat6)
f1_knn = f1_score(y_test, yhat6, average='macro')  # Use 'macro' for multiclass
precision_knn = precision_score(y_test, yhat6, average='macro')
recall_knn = recall_score(y_test, yhat6, average='macro')

# Printing results
print("KNN Model Performance:")
print(f"Train set Accuracy: {accuracy_score(y_train, neigh6.predict(X_train_scaled)):.2f}")
print(f"Test set Accuracy: {accuracy_knn:.2f}")
print(f"Precision: {precision_knn:.2f}")
print(f"Recall: {recall_knn:.2f}")
print(f"F1 Score: {f1_knn:.2f}")
print("\nClassification Report:\n", classification_report(y_test, yhat6))
```

✓ 8.0s

```
KNN Model Performance:
Train set Accuracy: 0.66
Test set Accuracy: 0.45
Precision: 0.36
Recall: 0.35
F1 Score: 0.35
```

I improved my KNN model by finding the best k value. This is the number of clusters formed that the data is classified for. I also added class_weight='balanced' to control the distribution of data. I chose the k value based on trial and error, but it would be most successful to form an elbow visualization, and when the graph starts flattening out at the elbow, this is the best k value.

Model Selection Voting Ensemble:

I decided to create an ensemble model combining these three models. This combines the effectiveness and differences of each model. This is seen with the linear relationships seen in the logistic regression model, feature selection identifying complex relationships in the Random Forest, and KNN illustrating local patterns. I used a voting ensemble with soft voting. This averages the predicted possibilities and chooses the class with the highest probability. This means the ensemble model chooses the most effective probabilities, taking the best from each model. I chose this method because the models vary in their confidence levels for each metric significantly, balancing the imbalance nature of the readmission variable.

```
Fitting 5 folds for each of 8 candidates, totalling 40 fits
Best Parameters for Random Forest: {'class_weight': 'balanced', 'max_depth': 10, 'min_samples_split': 2, 'n_estimators': 200}
Logistic Regression Cross-Validation F1 Score: 0.29
KNN Cross-Validation F1 Score: 0.34
Classification Report for Ensemble Model:
              precision    recall  f1-score   support

           0       0.16      0.09      0.11      2285
           1       0.38      0.36      0.37      7117
           2       0.56      0.63      0.59     10952

    accuracy                           0.47     20354
   macro avg       0.36      0.36      0.36     20354
weighted avg       0.45      0.47      0.46     20354
```

In creating the model using GridSearchCV and cross-validation for each of the three models combined, the scoring was not that much improved from each model individually.

“Class 0 (No Readmission):

Low Precision (0.16): The model has a high rate of false positives for this class, meaning many patients predicted to have no readmission actually belong to another class.

Low Recall (0.09): The model struggles to correctly identify patients who truly belong to this class, missing many cases.

Low F1-score (0.11): Indicates poor overall performance for this class, with the model performing poorly in balancing precision and recall.

Class 1 (Readmitted <30 Days):

Moderate Precision (0.38): The model does better at correctly predicting this class, but still has significant false positives.

Moderate Recall (0.36): It identifies some patients correctly but still misses a lot of cases.

Moderate F1-score (0.37): The model shows better but suboptimal performance for this critical class.

Class 2 (Readmitted >30 Days):

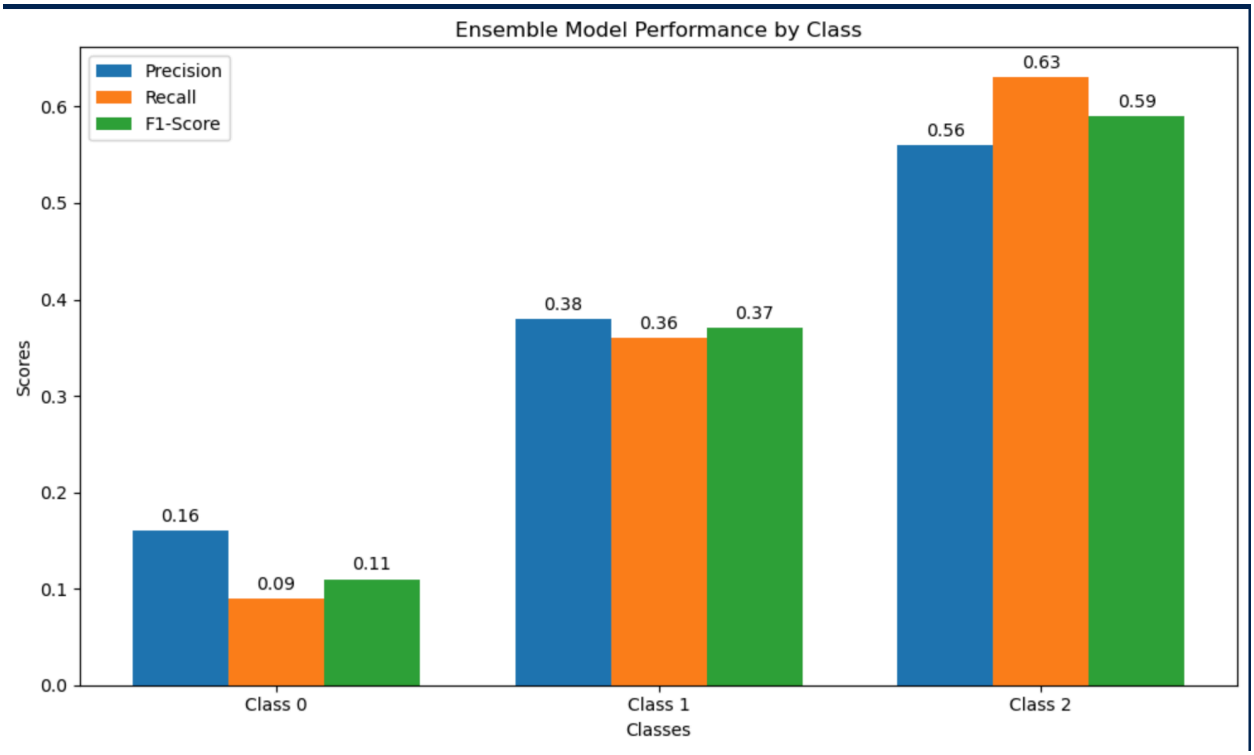
High Precision (0.56): The model is relatively good at identifying patients in this class with fewer false positives.

High Recall (0.63): The model captures most of the true positives for this class.

High F1-score (0.59): Strong performance in balancing precision and recall, making this the best-performing class in the model." (chat GPT, 2025)

Business Objective: It is most important to make sure of the accuracy for class one. This is because if a patient is readmitted within a 30-day window it is likely due to an immediate error in treatment. This also means procedures are commonly related to the initial admission. The class ii accuracy is sufficient. Class zero needs significant improvement. There is a large percentage of false positives in this class which can result in missed opportunities for preventive care for

patients. This is also essential to increase accuracy.



As shown in the visualization and accuracy scores, it is clear there are still issues with balancing the data set. To improve the data set it would be useful to apply oversampling techniques like SMOTE or undersampling. Oversampling techniques add more data points to the minority instances, making them comparable to the majority instances. This creates a more balanced data set helping the model more accurately classify the minority instances.

- Clustering Models

I focused on predicting time_in_hospital with the features of num_medications, num_lab_procedures, and num_procedures to show the intensity of care, and number_diagnoses to indicate patient complexity. This was based on the same business objective of resource utilization. I could provide additional features admission_type_id; capturing reasoning of the

care, enabling the model to group by importance, discharge_disposition_id; identifying groups that would need to be readmitted again, or age; and identifying patterns here.

Clustering helps uncover latent structures that are useful for segmentation. This is done by identifying groups of patients with similar characteristics, which could inform personalized care plans and optimize hospital workflows. While clustering models are unsupervised, powerful insights can be drawn using context. (chatGPT, 2025)

Overall, numerous different features can be used in clustering to identify patterns and to group the data. I decided to focus on these specific features and target variables to try and group based on tailoring needs for patients based on these clusters, optimizing hospital practice. I stuck with these original models because of the required computational complexity to run an effective model on a sufficient-size of the data set. Also, I believe the other models were worth spending more time to reveal the insights from them. Clustering this data made less sense from a business approach, as it was more productive to predict the length a patient would stay in the hospital through linear regression, and group patients based on whether they would be readmitted through classification.

Implementation:

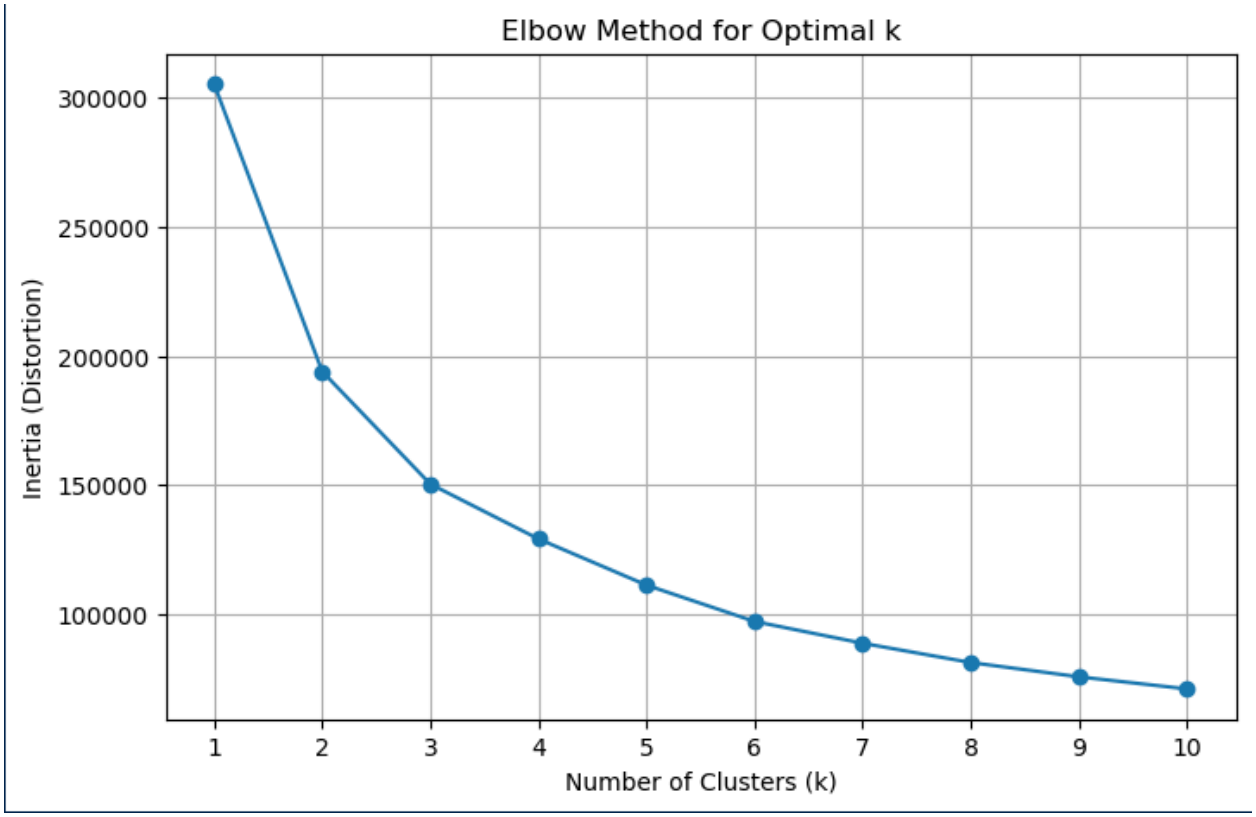
I started by splitting and standardizing the features. “The silhouette score is a metric used to evaluate the quality of clusters. It quantifies how well each data point fits within its assigned cluster compared to other clusters” (chatGPT, 2025). The silhouette score is calculated by dividing an (instance subtracted by its cluster) by (that instance and all points in the nearest neighboring cluster). A score of one identifies well-matched clusters while a score of negative one shows the data is in the wrong cluster.

```
import random
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
import numpy.testing as testing
from sklearn.metrics import silhouette_score


# Convert DataFrame to NumPy array for clustering
selected_features = ['time_in_hospital', 'num_medications', 'num_lab_procedures']
data_array = data[selected_features].values

scaler = StandardScaler()
data_array = scaler.fit_transform(data[selected_features])

# Initialising and training K-Means model
kmeans = KMeans(init = "k-means++", n_clusters = 4, n_init = 12, random_state=42)
kmeans.fit(data_array)
# Predict clusters and evaluate
predicted_labels = kmeans.predict(data_array)

#print(f"K-Means Silhouette Score: {}")
kmeans_silhouette = silhouette_score(data_array, predicted_labels)
print(f"K-Means Silhouette Score: {kmeans_silhouette:.2f}")
```

✓ 4m 31.4s

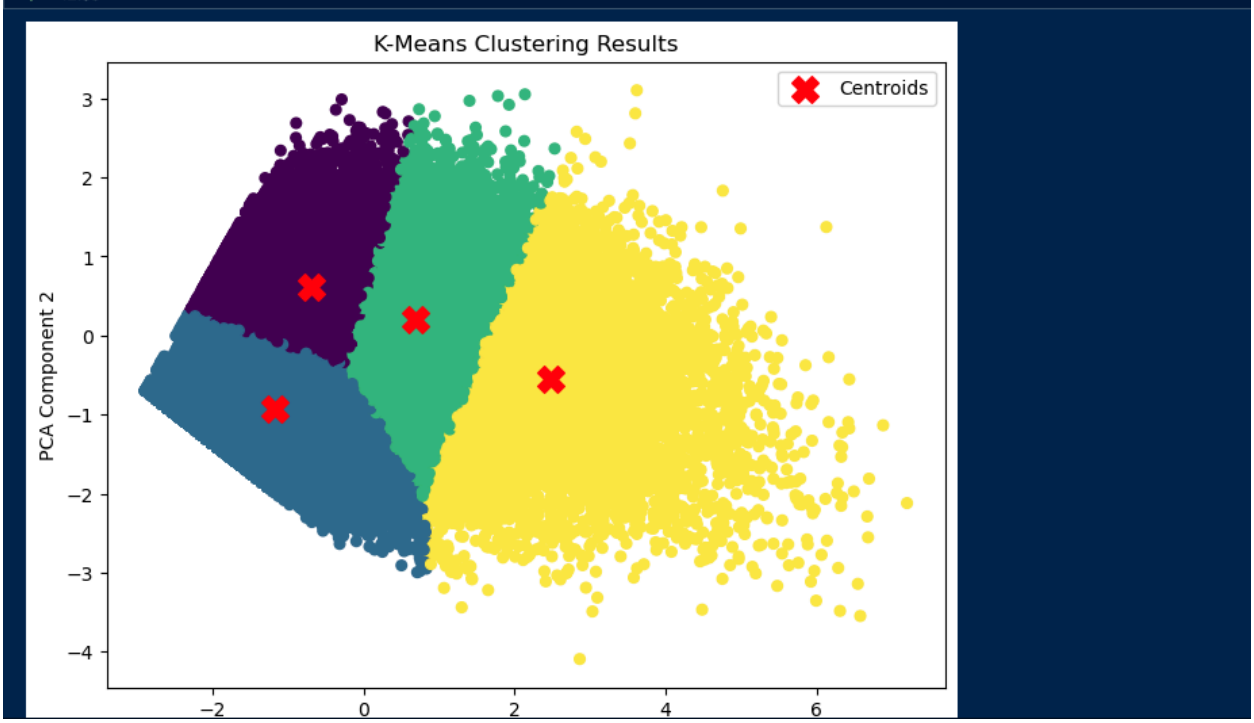
```
K-Means Silhouette Score: 0.25
```

I used an elbow visualization to predict if the n_clusters = 4 was correct. Because the line begins to flatten out around three or four this is a correct selection of this parameter.



PCA for dimension reduction:

```
from sklearn.decomposition import PCA

scaler = StandardScaler()
scaled_data = scaler.fit_transform(data_array)

# Applying PCA to reduce dimensions
pca = PCA(n_components=2)
pca_data = pca.fit_transform(scaled_data)

final_labels = kmeans.fit_predict(pca_data)

# Visualising clusters for K-Means in 2D
# Hint: Set n_components=2 for 2D visualisation or n_components=3 for 3D visualisation.
plt.figure(figsize=(8, 6))
plt.scatter(pca_data[:, 0], pca_data[:, 1], c=final_labels, cmap='viridis', s=30)
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], c='red', marker='X', s=200, label='Centroids')
plt.title("K-Means Clustering Results")
plt.xlabel("PCA Component 1")
plt.ylabel("PCA Component 2")
plt.legend()
plt.show()
✓ 12.5s
```



Overlap between the blue and purple clusters suggests the need for better feature separation or a different clustering algorithm. I decided to implement other models to better cluster the data.

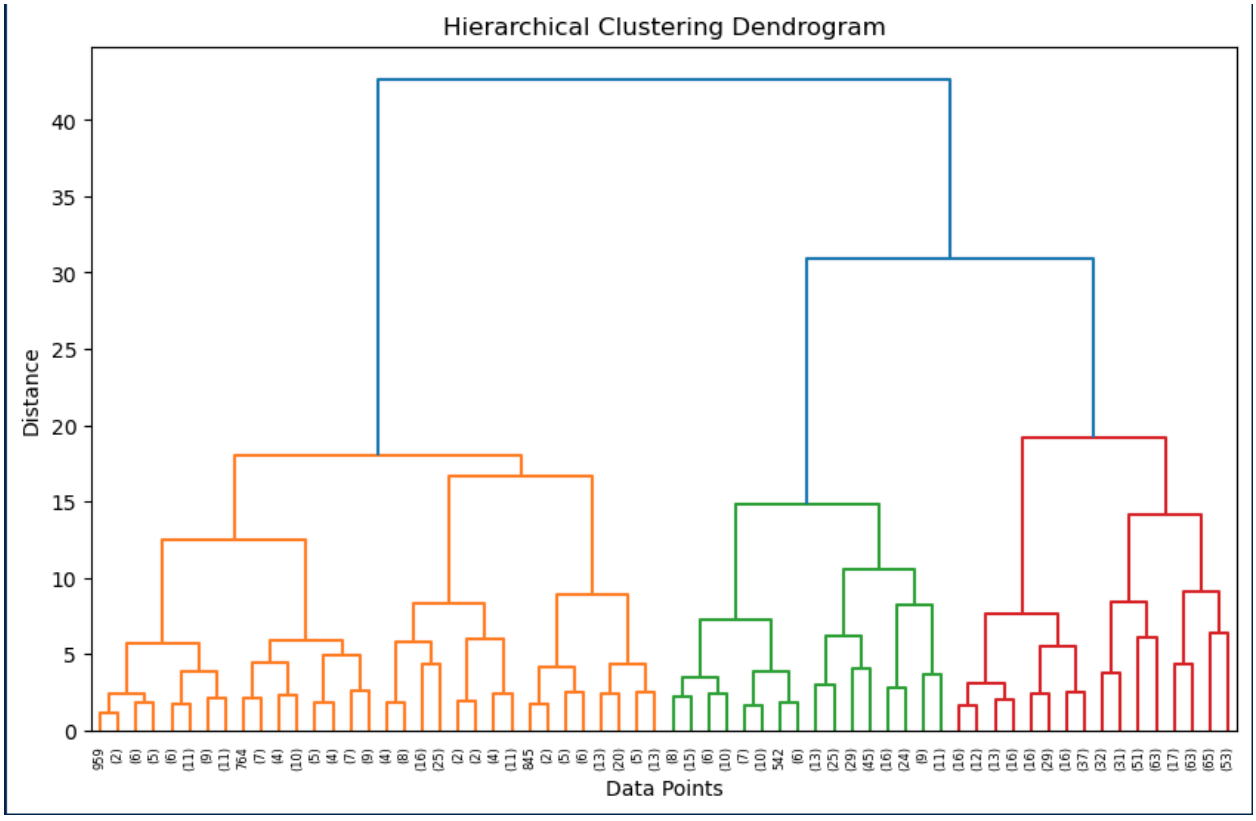
The better results shown in this visualization are representative of overfitting because the data is only trained on a few models. Dimension reduction, which focuses the model on more valuable features is not useful because the features are already set.

```
#does not run/takes too long
from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics import silhouette_score
from sklearn.preprocessing import StandardScaler

# Select a subset of relevant features to speed up clustering
selected_features = ['time_in_hospital', 'num_medications', 'num_lab_procedures']  # Focused on few fe
X = data[selected_features]

# Reduce sample size for faster computation
sample_size = 1000
X_sample = X.sample(n=sample_size, random_state=42)

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_sample)


# Initialising and fitting hierarchical clustering
n_clusters = 4
hierarchical = AgglomerativeClustering(n_clusters=n_clusters, metric='euclidean', linkage='single')
hierarchical_labels = hierarchical.fit_predict(X)

# Evaluating with Silhouette Score
hierarchical_silhouette = silhouette_score(X, hierarchical_labels)
print(f"Hierarchical Clustering Silhouette Score: {hierarchical_silhouette:.2f}")
```

```
Hierarchical Clustering Silhouette Score: 0.46
```

Scores can be improved by focusing on more features such as age, and number_diagnoses, and admission_type_id. However, due to run time, I selected only these three features and a small sample size. Training the model on an increased number of features and a larger amount of data would significantly increase the accuracy of the model. This model is much more effective because Hierarchical Clustering “does not require pre-specifying the number of clusters (you can choose k by visually inspecting the dendrogram)”(chatGPT, 2025). This means this model works better for non-spherical clusters which are representative of this data set (shown in the PCA

visualization).



This dendrogram chart shows the “hierarchical relationship between data points and clusters.

Each vertical line represents a cluster merger as the algorithm moves up the hierarchy.

The height at which two clusters merge indicates their similarity (lower height = more similar).”(chatGPT, 2025) This illustrates that one of the clusters is very unique while the other is fairly similar. Most of the data is able to be clustered within three splits of data.

DBSCAN for Density-Based Clustering

```
from sklearn.preprocessing import StandardScaler

# Normalize the data
X_scaled = StandardScaler().fit_transform(X_sample)

# Adjust eps and min_samples for more clusters
dbscan = DBSCAN(eps=0.3, min_samples=5).fit(X_scaled)

# Assign clusters to the dataset
DBSCAN_dataset = X_sample.copy()
DBSCAN_dataset['Cluster'] = dbscan.labels_

# Filtering noise points (labeled as -1)
cluster_counts = DBSCAN_dataset['Cluster'].value_counts().to_frame()
print("Cluster counts including noise:\n", cluster_counts)

DBSCAN_filtered = DBSCAN_dataset[DBSCAN_dataset['Cluster'] != -1]

# Check if there are at least 2 clusters (excluding noise)
if len(DBSCAN_filtered['Cluster'].unique()) > 1:
    dbscan_silhouette = silhouette_score(
        DBSCAN_filtered[selected_features], DBSCAN_filtered['Cluster']
    )
    print(f"DBSCAN Silhouette Score (excluding noise): {dbscan_silhouette:.2f}")
else:
    print("Silhouette Score cannot be calculated (less than 2 clusters).")


# changing eps greatlehy changes the silhoutte score because it controls the dista
# With a eps around 0.1, all data largely fits into one category with very few it
# I settled for 0.6 because this the first eps where more than 2 clusters form.
```

```
Cluster counts including noise:
         count
Cluster
-1         970
 1          12
 2           8
 0           5
 3           5
DBSCAN Silhouette Score (excluding noise): 0.29
```

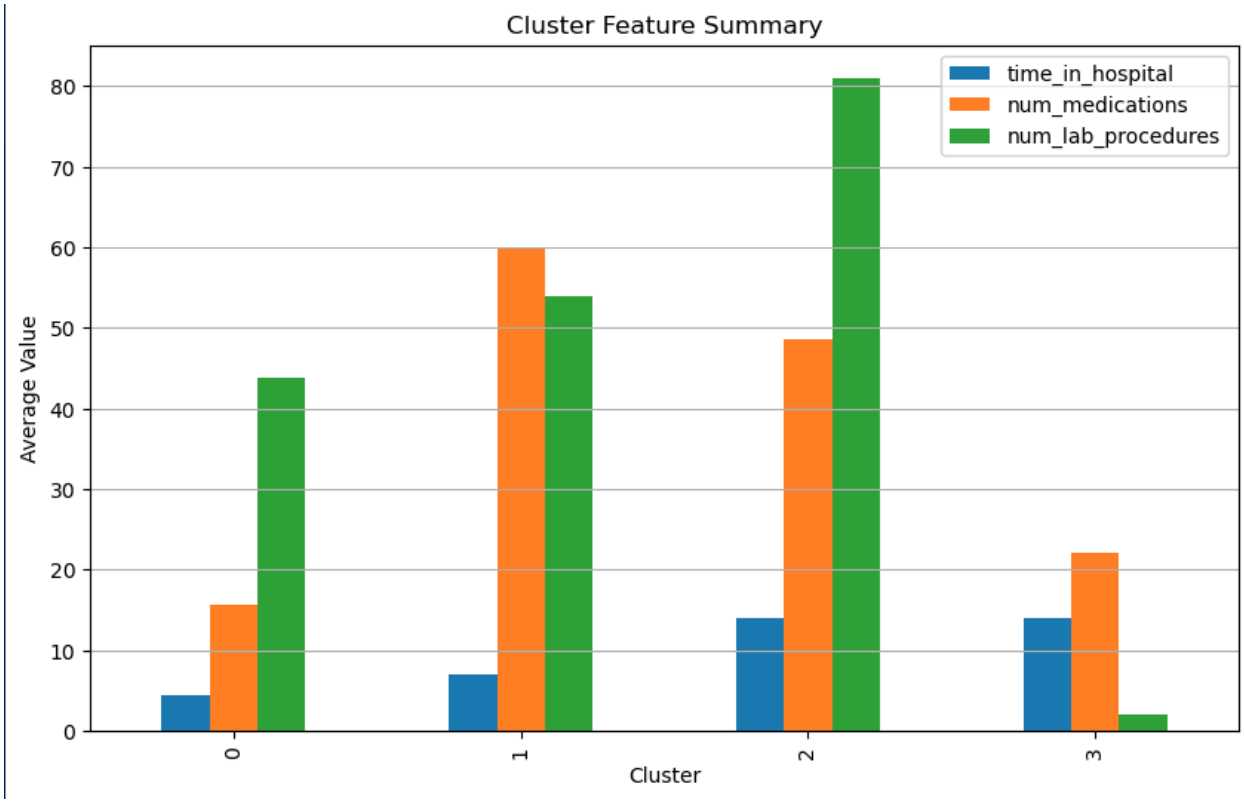
Changing the parameter eps greatly changes the silhouette score because it controls the distance between points to form clusters. With an eps of around 0.1, all data largely fits into one category with very few others. This means a high silhouette score is not always ideal. I settled for 0.6 because this is the first episode where more than 2 clusters form. On the other hand, min_samples=5 is used to set the minimum of data points to form a cluster. Most of the data fits into one cluster so this is not an effective model at representing this data set.

“DBSCAN is a density-based clustering algorithm that groups together points that are closely packed (points with high density) and marks points in low-density regions as outliers or noise. Unlike K-Means, DBSCAN does not require you to predefine the number of clusters and is particularly effective at identifying clusters of arbitrary shapes.” (chatGPT, 2025)

The data set clusters are not arbitrary shapes with similar shapes to the clusters, but they are smooth borders, not arbitrary shapes.

Clustering Final Model

Based on the extensive computational demands, and lack of derivable meaning from clustering, the best model was the Hierarchical clustering model. This is because the model was best for the non-spherical similar shape of clustering of the data set and can capture nested relationships. “Nested relationships refer to a hierarchical structure within data where groups (clusters) contain smaller, more specific subgroups (chatGPT, 2025). Based on the dendrogram, the nested relationships in the data set are shown. Specifically, on the right side of the structure, it is clear that there are small, specific subgroups. This is also shown in the DBSCAN results where there is one large cluster and several very small clusters that are very similar.



Business Objective

From this visualization, it's clear that Cluster Two represents patients requiring significant attention and resources, including specialized staff for lab work and pharmacy services due to the high level of number lab procedures followed by the number of medications, with higher time in the hospital relative to other groups.

On the other hand, Cluster Zero includes patients who experience routine care and could be managed more efficiently by general practitioners or outpatient services. This cluster indicates groups that have experienced several important procedures due to the high level of procedures but relatively low time in the hospital.

Cluster three stands out with patients who have longer hospital stays but minimal medical needs, suggesting the need to investigate reasons for these prolonged stays, such as administrative delays or recovery issues. Developing targeted intervention plans for this group could help reduce stay durations and improve turnover as the lack of procedures, but relatively high medication illustrates that these patients could be assisted outside of the hospital.

Lastly, Cluster One could identify the base group, requiring lots of attention to resources, without a significant requirement of hospital space.

Overall, the hierarchical clustering structure provides a valuable way to organize patients. It allows for broad categorization based on resource intensity and enables further analysis of subgroups to develop specific strategies for optimization. (chatGPT, 2025)

1. Conclusion and Reflection

Overall, I cleaned and prepared the diabetic data set to implement Regression, Classification, and Clustering Models to optimize hospital resource utilization by predicting time in hospital and patient readmission.

To begin, my final Linear Regression Model is useful in predicting a specific value for time in the hospital so the hospital can predict how long a patient will stay in the hospital based on medical history.

Additionally, this ability to predict patient stay durations benefits both patients and the hospital. Patients can receive faster and more efficient care and predictions can aid in cost planning. The time a patient stays in the hospital is directly correlated to the amount of money spent, allowing for the hospital to better manage and allocate resources.

I selected time in hospital due to the strong correlations observed in the dataset and the intuitive sense of relationships between features. For instance, a patient who has received more medications, undergone more procedures, and has been diagnosed with more conditions is likely to have spent more time in the hospital. This is because the administration of medications, procedures, and diagnoses inherently requires time. Any interaction with a patient in a hospital setting takes time, which is reflected in these variables.

I selected time in hospital as the target variable for linear regression not only because it is a consistent numeric value and has strong correlations with other variables, but also because it is an incredibly valuable metric for hospitals to predict. Specifically, predicting time in hospitals can enhance hospital efficiency. If, upon admission, a hospital can predict the accurate duration

of a patient's stay, it can better allocate resources. Hospitals operate with finite resources which they can optimize through this metric.

Overall, this model is not strong enough to be used in a business objective. This is because the predictions can yield variance within days with an overfitting model producing MAE and MSE scores around one. The predictive capabilities of this model are not strong enough to currently be used by a hospital. Also, I believe a model that would predict accurately enough to be used is not feasible. This is due to the lack of strong correlations and features in the data set. In order to predict strong values for a linear regression model, a new data set must be used.

Second, my final Voting Ensemble combining Logistic Regression, Random Forest, and KNN using soft voting combines the best qualities of each classification model to identify patients most at risk for readmission, creating possible interventions to produce better medical attention, and reducing readmission rates.

It is most important to make sure the accuracy for class one is high. This is because if a patient is readmitted within a 30-day window it is likely due to an immediate error in treatment. As shown in the visualization and accuracy scores, it is clear there are still issues with balancing the data set. To improve the data set it would be useful to apply oversampling techniques like SMOTE or undersampling.

Overall, the ability of this model to classify if a patient will be readmitted based on medical history is currently not accurate. If the accuracy of class one was improved, this model would be very useful for hospitals. Being able to predict if a patient will be readmitted, specifically given a timeframe of less than 30 days is very useful because this can associate certain practices that that patient received to having to be readmitted. This is very useful in optimizing hospital practices because knowing if someone will need to be readmitted will plan ahead to provide for this

patient. Also, medical attention can be shown as ineffective due to this prediction, making sure the hospital gives helpful treatment, and improving patient care. This also can be used for hospital cost reduction by reducing unnecessary readmission, saving money on both ends.

Patients can be segmented based on their likelihood of being readmitted. Also, hospitals' efficiency can be benchmarked and compared for specific needs to be readmitted.

```
Fitting 5 folds for each of 8 candidates, totalling 40 fits
Best Parameters for Random Forest: {'class_weight': 'balanced', 'max_depth': 10, 'min_samples_split': 2, 'n_estimators': 200}
Logistic Regression Cross-Validation F1 Score: 0.29
KNN Cross-Validation F1 Score: 0.34
Classification Report for Ensemble Model:
              precision    recall  f1-score   support

           0       0.16      0.09      0.11      2285
           1       0.38      0.36      0.37      7117
           2       0.56      0.63      0.59     10952

    accuracy                           0.47     20354
   macro avg       0.36      0.36      0.36     20354
weighted avg       0.45      0.47      0.46     20354
```

While these predictors are very strong, the lack of balance and high dimensionality of the data set make this classifier model unfeasible. The scores of the model illustrate a need for a more balanced data set, and longer training using a Voting Ensemble.

Lastly, my final Random Hierarchical Clustering was very useful in finding and creating unique clusters as a valuable way to organize patients based on hospital interactions, to better understand the required resource intensity of each cluster of patients.

Overall, this model is extremely feasible for hospitals to gain insights into the groupings of their patients. By understanding how much medical attention a specific patient will require, and sorting these patients, the hospital can best optimize medical practices.

Improvements in this model could be made by training the model on a larger amount of features and data. Through this training, stronger patterns could be identified within groups to apply business objectives.

Last of all, both the Random Hierarchical Clustering and Voting Ensemble models would be very useful in optimizing medical practices. Improvements in regulating the balance of the data set for the ensemble method with larger training time and using more features to identify patterns in the Random Hierarchical Clustering would create effective models.

[References]

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. BMC Health Services Research, 14(1), 228.

ChatGPT. (2025). Explanation of oversampling and undersampling techniques for imbalanced datasets. OpenAI. Retrieved January 23, 2025, from https://chat.openai.com