PART I: RESEARCH QUESTION

A1. PROPOSAL OF QUESTION

What are the major predictors influencing the initial administration among patients?

A2. DEFINE GOAL

The primary goal of this analysis is to identify key predictors. The aim is to determine the variables that significantly influence in predicting the initial administration method. This analysis also aims to understand the relationship between various demographic, medical history and medical conditions and the initial administration of patients.

PART II: METHOD OF JUSTIFICATION

B1. EXPLANATION OF CLASSIFICATION METHOD

For this analysis, I am using K-Nearest Neighbor (KNN). KNN is an intuitive classification algorithm that operates based on the principle of similarity, and those similar things are close to each other. It assumes that data points with similar features tend to belong to the same class or have similar output values.

For a given data point, KNN finds the K nearest neighbors to that point based on a distance metric. In classification tasks, the class of a new data point is determined by a majority vote among its K nearest neighbors. The expected outcome of my analysis is to leverage the similarity of data points to predict new and unseen data points.

B2. SUMMARY OF METHOD ASSUMPTION

KNN assumes that proximity in the feature space corresponds to similarity in the target variable. This implies that if two data points are close to each other in the feature space, they are likely to belong to the same class.

B3. PACKAGES OR LIBRARIES LIST

Packages & Libraries	Usage
csv	For reading and writing CSV files
pandas	For data manipulation, provides data structure like DataFrame and Series
warnings	For controlling warning messages by suppressing certain warning messages that may not be critical
platform	Provides information about the platform and/or version of the Python running
numpy	For numerical computing, also for mathematical functions that enables arrays to efficiently operate
matplotlib.pylot	A plotting library; used to create histograms, bar charts and ROC curves
seaborn	A statistical data visualization based on matplotlib; used to create heatmaps and correlation matrices

sklearn (scikit-learn)	A machine learning library that provides tools for data mining and data analysis.
train_test_split	For splitting the dataset into training and testing sets
GridSearchCV	For performing hyperparameter tuning using grid search
KFold	For cross validation
StandardScaler	For standardizing features by removing the mean and scaling to unit variance
LabelEncoder	For encoding categorical variables into numerical format
KNeighborsClassifier	For building the K-Nearest Neighbors classification model
accuracy_score	For computing the accuracy of the classification model
classification_report	For generating a text report showing the main classification metrics
roc_auc_score	For computing the ROC AUC score to evaluate model
roc_curve	For generating the ROC curve
SelectKBest	For selecting the top K features based on univariate statistical test
Statsmodels import variance_inflation_factor	For calculating the variance inflation factor to help detect multicollinearity among predictor variables

PART III: DATA PREPARATION

C1. DATA PREPROCESSING

One relevant data preprocessing goal for KNN is feature scaling. Since KNN relies on calculating the distance between data points to determine similarity, the scale of features affects the distance computations. Features with larger scales can dominate the distance calculation, thus leading to biased results. By scaling features to the same range, feature scaling ensures that each feature contributes proportionally to the distance calculation.

C2. DATA SET VARIABLES

After Feature Selection using SelectKBest

Numeric	Categorical
TotalCharge	Allergic_rhinitis
Additional_charges	Complication_risk
VitD_levels	Gender
vitD_supp	Soft_drink
Income	

C3. STEPS FOR ANALYSIS

The preprocessing steps for this analysis:

1. Initialize all libraries and packages to be used

```
# Data Handling
import csv
import pandas as pd

# Ignore Warnings
import warnings
warnings.filterwarnings("ignore")

# System Information
import platform

# Numeric Computation and Array Handling
import numpy as np

# Data Visualization
import matplotlib.pyplot as plt
import seaborn as sns

# Machine Learning
from sklearn.model_selection import train_test_split, GridSearchCV, KFold
from sklearn.preprocessing import LabelEncoder, StandardScaler, label_binarize
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, roc_auc_score, roc_curve
from sklearn.feature_selection import SelectKBest, f_classif

# Statistical Analysis
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

2. Print Python version

```
# Print Python version
print("Python Version:", platform.python_version())
```

```
Python Version: 3.11.5
```

3. Load the dataset into Pandas DataFrame

```
# Specifiy CSV file path
file_path = r'C:\Users\kolgi\OneDrive - Western Governors University\d209\medical_clean.csv'

# Open the CSV file and read it using DictReader
with open(file_path, 'r') as csvfile:
    csvreader = csv.DictReader(csvfile)

# Read the CSV file into a pandas DataFrame then open head
df = pd.read_csv(file_path)
```

4. View the first 5 rows of the DataFrame
   df.head()
5. View the index, column names, non-null count, and data types
   df.info()
6. Check for null/missing values in the dataset, then count the null values for each column
   df.isnull().sum()
7. Check for duplicates in the data
   df.duplicated()

8. Drop irrelevant columns for this analysis

```
columns_to_drop = ['CaseOrder', 'Customer_id', 'Interaction', 'UID', 'City', 'State',
                   'County', 'Zip', 'Lat', 'Lng', 'Population', 'TimeZone', 'Job',
                  'Item1', 'Item2', 'Item3', 'Item4', 'Item5', 'Item5', 'Item6', 'Item7', 'Item8']

# Drop the specified columns
df = df.drop(columns=columns_to_drop)

# Display the updated DataFrame
print(df.info)
```

9. Visualize categorical variables with a histogram

```
# Visualization only for categorical variables
bar_graphs = ['Initial_admin', 'Area', 'Marital', 'Gender', 'ReAdmis', 'Soft_drink',
              'Complication_risk', 'Services', 'HighBlood', 'Stroke', 'Overweight',
              'Arthritis', 'Diabetes', 'Hyperlipidemia', 'BackPain', 'Anxiety',
              'Allergic_rhinitis', 'Reflux_esophagitis', 'Asthma']

# Calculate the number of rows and columns for the subplots
num_rows = len(bar_graphs) // 3 + (1 if len(bar_graphs) % 3 != 0 else 0)
num_cols = min(3, len(bar_graphs))

fig, axes = plt.subplots(nrows=num_rows, ncols=num_cols, figsize=(15, 5*num_rows))

for i, col in enumerate(bar_graphs):
    row_index = i // num_cols
    col_index = i % num_cols
    df[col].value_counts().plot(kind='bar', ax=axes[row_index, col_index])
    axes[row_index, col_index].set_title(col)

# Remove any unused subplots
for i in range(len(bar_graphs), num_rows * num_cols):
    fig.delaxes(axes.flatten()[i])

plt.tight_layout()
plt.show()
```

10. Convert categorical variables to numerical using Label Encoding

```
# List of categorical columns to be encoded
categorical_columns = [
    'Area', 'Marital', 'Gender', 'ReAdmis',
    'Soft_drink', 'Initial_admin', 'HighBlood', 'Stroke', 'Complication_risk',
    'Overweight', 'Arthritis', 'Diabetes', 'Hyperlipidemia', 'BackPain',
    'Anxiety', 'Allergic_rhinitis', 'Reflux_esophagitis', 'Asthma', 'Services'
]

# Initialize LabelEncoder
encoder = LabelEncoder()

# Encode categorical columns
for column in categorical_columns:
    df[column] = encoder.fit_transform(df[column])

# Display the updated DataFrame
pd.set_option('display.max_columns', None)
print(df)
```

11. Define the features (X) and target (y) for feature selection and print their shape

```
# From Elleh (2023)

# Assign to X all the predictor features
X = df.drop(["Initial_admin"], axis=1)
print(X.shape)

# Assign to y to the target variable
y = df["Initial_admin"]
print(y.shape)
```

```
(10000, 28)
(10000,)
```

12. Initialize SelectKBest with the f_classif scoring function and print shape of the transformed feature matrix. SelectKBest is a feature selection method. 'k' indicates that it will select all features based on the scoring function.

```
# From Elleh (2023)

# Initialize SKBest
skbest = SelectKBest(score_func=f_classif, k='all')
X_new = skbest.fit_transform(X, y)
print(X_new.shape)
```

```
(10000, 28)
```

13. Calculate p-values for each feature. Then, select only significant features with p-values of <0.05. Finally, print the significant features and their corresponding values and the p-values of all features.

```
# From Elleh (2023)

# Calculate P-values for X
p_values = pd.DataFrame({'Feature': X.columns, 'p-value': skbest.pvalues_}).sort_values('p-value')
significant_features = p_values[p_values['p-value'] < 0.05]

# Print Features to keep and all P-values
features_to_keep = significant_features['Feature']
print("Features to keep:")
print(features_to_keep)
print("\nP-values:")
print(p_values)
```

14. Create a new DataFrame with the selected features

```
# Create a new dataset with the selected features
X_new = X[features_to_keep]
```

15. Check for VIF

```
# Check VIF for multicollinearity issues amongst these features

# Create a new DataFrame with the selected features
X_new = X[features_to_keep]
print(X_new)

# Calculate the VIF for each feature
vif = pd.DataFrame()
vif["Feature"] = X_new.columns
vif["VIF"] = [variance_inflation_factor(X_new.values, i) for i in range(X_new.shape[1])]

# Print the VIFs
print(vif)
```

```
              Feature        VIF
0         TotalCharge   6.546100
1  Additional_charges   4.743249
2         VitD_levels  14.279640
3   Allergic_rhinitis   1.641234
4   Complication_risk   2.570350
5              Gender   1.905121
6           vitD_supp   1.398429
7          Soft_drink   1.343340
8              Income   2.938075
```

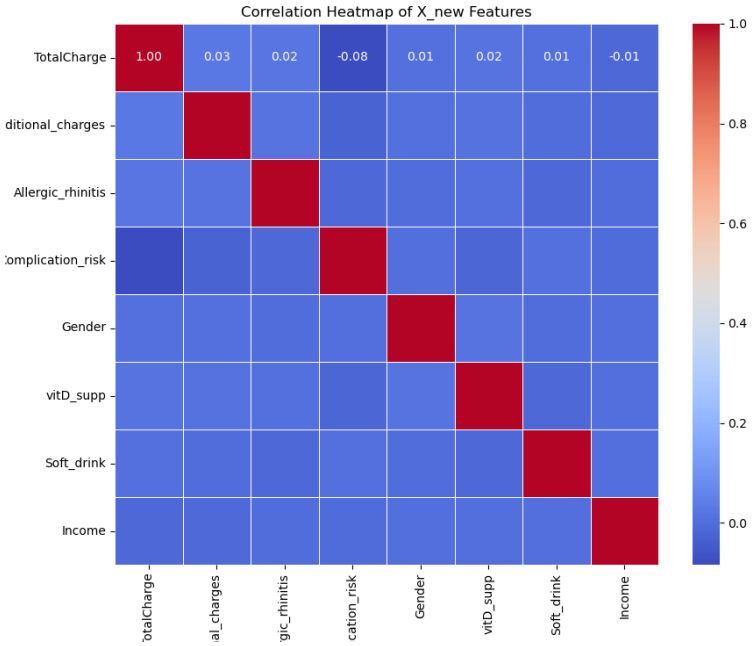
16. Drop the VitD_levels since it has a high multicollinearity then print the updated VIF

```
# Drop the 'VitD_levels' column from your dataset since it has a high multicollinearity
X_new = X_new.drop(columns=['VitD_levels'])

# Recalculate the VIF for the updated dataset
vif = pd.DataFrame()
vif["Feature"] = X_new.columns
vif["VIF"] = [variance_inflation_factor(X_new.values, i) for i in range(X_new.shape[1])]

# Print the updated VIF values
print(vif)
```

```
              Feature       VIF
0         TotalCharge  4.479221
1  Additional_charges  3.847659
2   Allergic_rhinitis  1.596953
3   Complication_risk  2.264100
4              Gender  1.831346
5           vitD_supp  1.376514
6          Soft_drink  1.320585
7              Income  2.633425
```

17. Calculate the correlation between X_new. Create a correlation matrix heatmap for visualization.

```
# Calculate the correlation matrix
correlation_matrix = X_new.corr()

# Plot the heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)
plt.title('Correlation Heatmap of X_new Features')
plt.savefig("corr_matrix.png")
plt.show()
```



18. Save cleaned dataset into a .csv file

```
# Save the filtered DataFrame to a CSV file
X_new.to_csv('filtered_med.csv', index=False)
```

19. Scale the feature matrix (X_new). Since only numeric variables need to be scaled, separate the feature variables by numeric and categorical. Then, concatenate the scaled numeric and categorical variables. Scaling features is essential to ensure that all features contribute equally to the analysis by bringing them to a similar scale. Since KNN is a distance-based algorithm, features with larger scales can dominate the distance calculation.

```
# Scale features but only the numeric columns

# Select only the numeric features
numeric_features = ['TotalCharge', 'Additional_charges', 'vitD_supp', 'Income']
X_numeric = X_new[numeric_features]

# Initialize the StandardScaler
scaler = StandardScaler()

# Fit and transform the numeric features
X_scaled_numeric = scaler.fit_transform(X_numeric)

# Combine the scaled numeric features with the categorical features
X_categorical = X_new[['Allergic_rhinitis', 'Complication_risk', 'Gender', 'Soft_drink']]
X_scaled = np.concatenate((X_scaled_numeric, X_categorical), axis=1)

print(X_scaled)
```

C4. CLEANED DATA SET

.csv attached as filtered_med.csv

PART IV: ANALYSIS

D1. SPLITTING THE DATA

Split the scaled feature data (X_scaled) and the target variable (y) into training and testing sets. Use the package scikit-learn with train_test_split to do this task. Use an 80/20 training split. The random_state parameter ensures reproducibility by fixing the random seed. Stratify = y ensures that the class distribution is preserved in both training and testing sets.

```
# Split the scaled data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, train_size=0.8, test_size=0.2, random_state=15, stratify=y)
```

Export the training and testing files to .csv

```
#Save the training and testing sets as csv files
pd.DataFrame(X_train).to_csv('X_train.csv')
pd.DataFrame(X_test).to_csv('X_test.csv')
pd.DataFrame(y_train).to_csv('y_train.csv')
pd.DataFrame(y_test).to_csv('y_test.csv')
```

X_train.csv, X_test.csv, y_train.csv and y_test.csv files attached.

D2. OUTPUT & INTERMEDIATE CALCULATIONS

1.

	Feature	VIF
0	TotalCharge	4.479221
1	Additional_charges	3.847659
2	Allergic_rhinitis	1.596953
3	Complication_risk	2.264100
4	Gender	1.831346
5	vitD_supp	1.376514
6	Soft_drink	1.320585
7	Income	2.633425

VIF measures the extent to which the variance of an estimated regression coefficient increased due to collinearity among the predictor values. VIF values greater than 10 indicates high multicollinearity. This suggests that the predictor value may be highly correlated with other variables in the model. VIF values between 5 to 10 suggest moderate collinearity. VIF values below 5 indicate acceptable levels of multicollinearity.

2.

```
Best k: KNeighborsClassifier(n_neighbors=29)
Best Accuracy: 0.501875
```

This is the output which is the best value of “k” (number of neighbors) and the corresponding best accuracy score. These values provide insights into the optimal configuration of the KNN model for the given data set.

This was achieved through hyperparameter tuning for the KNN classifier using grid search with cross-validation. It involves defining a parameter grid with a range of 1 to 30 for the number of neighbors, initializing the KNN classifier, and setting up a grid search with cross-validation to find the optimal value of “k” that maximizes the classification accuracy.

3.

```
The predicted classes for the instances in the test set:
[1 1 1 ... 1 1 1]

The predicted probabilities of the positive class for the instances in the test set:
[0.51724138 0.44827586 0.5862069  ... 0.62068966 0.34482759 0.48275862]
```

Predictions are made on the test data using predict(), which assigns each instance in the test set to the nearest class based on the trained model. Additionally, the predict.proba() is used to obtain the predicted probabilities of class 1 for each instance in the test set, and specifically the probabilities of class 1 are extracted using [:, 1].

4.

```
Classification Report
              precision    recall  f1-score   support

           0       0.36      0.10      0.16       501
           1       0.53      0.93      0.68      1012
           2       0.35      0.06      0.10       487

    accuracy                           0.51      2000
   macro avg       0.41      0.37      0.31      2000
weighted avg       0.44      0.51      0.41      2000
```

- Precision is the ratio of correctly predicted positive observations to the total predicted positives. For class 0, precision is 0.36. This indicates that out of all instances predicted as class 0, only 36% were actually class 0. For class 1, precision is 0.53. This indicates that 53% of instances predicted as class were actually class 1. For class 2, precision is 0.35. This indicates that that 35% of instances predicted as class 2 were actually class 2.

- Recall is the ratio of correctly predictive positive observations to all observations in actual class. For class 0, recall is 0.10. this indicates that only 10% of actual class 0 instances were correctly predicted as class 0. For class 1, recall is 0.93. This indicates that 93% of all actual class 1 instances were correctly predicted as class 1. For class 2, recall is 0.06. This indicates that only 6% of all actual class 2 instances were correctly predicted as class 2.
- F1-score is the harmonic mean of precision and recall. It is balance between precision and recall. For class 0, F1-score is 0.16. For class 1, F1-score is 0.68. For class 2, F1-score is 0.10. Compared to class 0 and class 2, class 1 has the most significant number among the classes.
- Support is the number of actual occurrences of the class in the specified dataset.
- Accuracy means the overall accuracy of the model across all classes. In this analysis, it's 0.51. This indicates that 51% of the predictions were correct.
- Macro average is the average of precision, recall, and F1-score across all classes, without considering class imbalance.
- Weighted average is the average of precision, recall, and F1-score across all classes, considering class imbalance.
- The classification report suggests that the model performs well in predicting class 1, as indicated by the high precision, recall, and F1-score. However, it struggles with classes 0 and 2, as indicated by lower precision, recall, and F1-score values for these classes.

5.

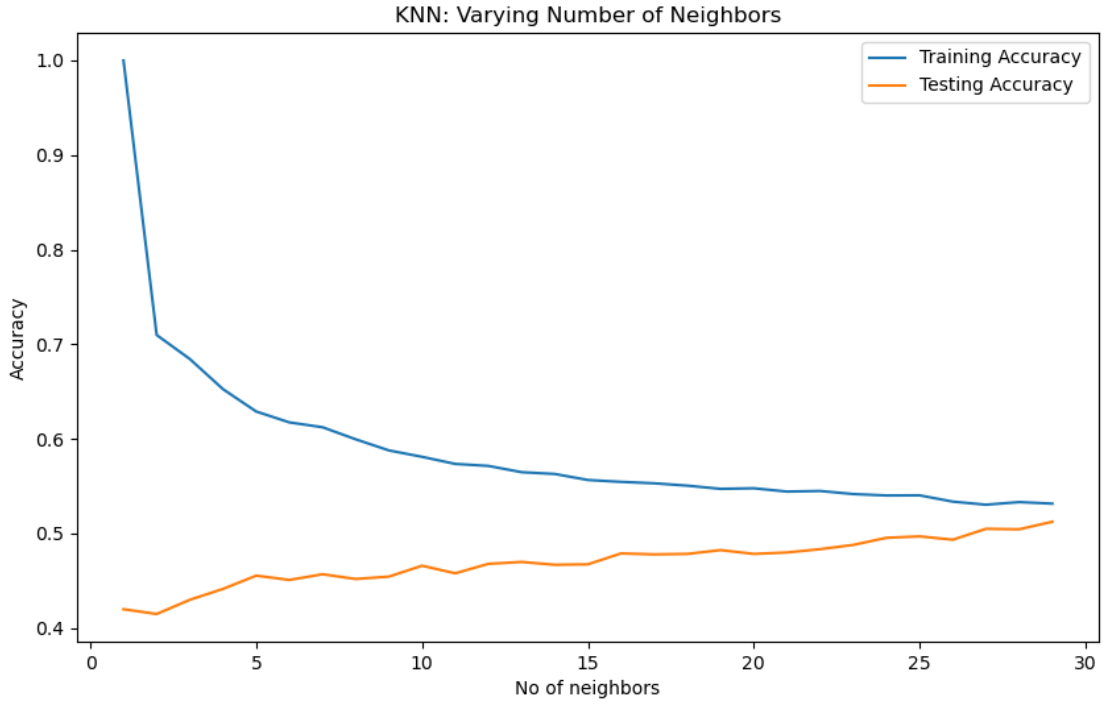


Figure 1 KNN Line Graph

The graph shows the training and testing accuracy of the KNN model.

- The training accuracy line shows that as the number of neighbors increases, the training accuracy decreases. This is because a larger number of neighbors smooths out the decisions boundary, leading to a simpler model.

- The testing accuracy line shows an initial increase with more neighbors, indicating better generalization. However, after a certain point, it starts to decrease due to the model becoming too simple and underfitting the data.

6.
```
AUC: 0.559237233200672
ROC AUC Score (OvR): 0.4387961901413984
```

- Area Under the Roc Curve (AUC) measures the ability of the model to distinguish between positive and negative classes across all possible threshold values. To calculate the AUC, use the best_knn to output the predicted probabilities of the classes. Then, use predict_proba() to get the predicted probabilities for the test data (X_test). Finally, use the roc_auc_score() to calculate the AUC score for each class individually using the one-vs-rest (OVR) strategy since there are three unique values in Initial_admin.
- ROC AUC Score is the average ROC AUC Score across all classes in a multi-class classification problem. It represents the overall performance of the classifier in distinguishing between different classes. To find the ROC AUC Score, convert the true labels into binary format. This conversion makes it easier for multiclass classification evaluation. It is also important to reshape y_pred_prob array to a two-dimensional array with a single column. This is necessary so roc_auc_score function to handle the predicted probabilities correctly. Finally, the roc_auc_score() computes the ROC AUC score for each class. In the macro averaging strategy, the ROC AUC score for each class is computed independently, and then the unweighted mean of these scores to get the final ROC AUC score.

7.
```
Training Accuracy: 0.531625
Testing Accuracy: 0.5125
```

Calculate the training and testing accuracy by finding the y_train_pred and y_test_pred using the trained KNN model. Then, use the accuracy_score() to calculate the accuracy of the model predictions compared to the true labels. It computes the accuracy, which is the proportion of correctly classified instances among all instances, for both the training and testing sets.
The training accuracy of the KNN model is 53.16%, while the testing accuracy is 51.25%. This indicates that the model performs slightly better on the training data.

D3. CODE EXECUTION

1. Code to find the VIF of the final features to keep

```
# Drop the 'VitD_levels' column from your dataset since it has a high multicollinearity
X_new = X_new.drop(columns=['VitD_levels'])

# Recalculate the VIF for the updated dataset
vif = pd.DataFrame()
vif["Feature"] = X_new.columns
vif["VIF"] = [variance_inflation_factor(X_new.values, i) for i in range(X_new.shape[1])]

# Print the updated VIF values
print(vif)
```

```
              Feature       VIF
0         TotalCharge  4.479221
1  Additional_charges  3.847659
2   Allergic_rhinitis  1.596953
3   Complication_risk  2.264100
4              Gender  1.831346
5           vitD_supp  1.376514
6          Soft_drink  1.320585
7              Income  2.633425
```

2. Code to find the best value of “k”. 29 as best k means the model considered the 29 nearest neighbors when making predictions.

```
# Define the parameter grid
param_grid = {'n_neighbors': range(1, 30)}

# Initialize the KNN classifier
knn = KNeighborsClassifier()

# Initialize GridSearchCV with the KNN classifier, the parameter grid, cross-validation (KFold), and the scoring metric (accuracy)
grid_search = GridSearchCV(knn, param_grid, cv=KFold(n_splits=5), scoring='accuracy')

# Fit the grid search to the data
grid_search.fit(X_train, y_train)

# Get the best parameters and best score
best_k = grid_search.best_params_['n_neighbors']
best_score = grid_search.best_score_

# Get the best KNN model
best_knn = grid_search.best_estimator_

print("Best k:", best_knn)
print("Best Accuracy:", best_score)
```

```
Best k: KNeighborsClassifier(n_neighbors=29)
Best Accuracy: 0.501875
```

3. Code to find the predicted class and the predicted probabilities

```
# Initialize KNeighborsClassifier with the best k value
best_knn = KNeighborsClassifier(n_neighbors=29)

# Fit the best_knn model to the training data
best_knn.fit(X_train, y_train)

# Make predictions on the test data using the best_knn model
y_pred = best_knn.predict(X_test)
y_pred_prob = best_knn.predict_proba(X_test)[:, 1]

print("The predicted classes for the instances in the test set:")
print(y_pred)
print("\nThe predicted probabilities of the positive class for the instances in the test set:")
print(y_pred_prob)
```

```
The predicted classes for the instances in the test set:
[1 1 1 ... 1 1 1]

The predicted probabilities of the positive class for the instances in the test set:
[0.51724138 0.44827586 0.5862069  ... 0.62068966 0.34482759 0.48275862]
```

4. Code for classification report

```
# Print classification report
print("Classification Report")
print(classification_report(y_test, y_pred))
```

```
Classification Report
              precision    recall  f1-score   support

           0       0.36      0.10      0.16       501
           1       0.53      0.93      0.68      1012
           2       0.35      0.06      0.10       487

    accuracy                           0.51      2000
   macro avg       0.41      0.37      0.31      2000
weighted avg       0.44      0.51      0.41      2000
```

5. Code for the KNN line graph

```
# from Boorman(n.d.)

train_accuracies = {}
test_accuracies = {}
neighbors = np.arange(1, 30)
for neighbor in neighbors:
    knn = KNeighborsClassifier(n_neighbors=neighbor)
    knn.fit(X_train, y_train)
    train_accuracies[neighbor] = knn.score(X_train, y_train)
    test_accuracies[neighbor] = knn.score(X_test, y_test)

plt.figure(figsize=(10, 6))
plt.title("KNN: Varying Number of Neighbors")
plt.plot(neighbors, list(train_accuracies.values()), label="Training Accuracy")
plt.plot(neighbors, list(test_accuracies.values()), label="Testing Accuracy")
plt.legend()
plt.xlabel("No of neighbors")
plt.ylabel("Accuracy")
plt.savefig("knn.png")
plt.show()
```

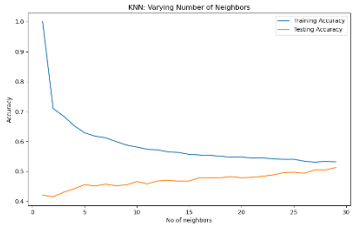


Figure 1 KNN Line Graph

6. Code for AUC and ROC AUC Score

```
# Predict the class probabilities for each sample in the test set using the trained KNN model
y_scores = best_knn.predict_proba(X_test)

# Calculate the AUC score for each class individually using the one-vs-rest (OvR) strategy
auc = roc_auc_score(y_test, y_scores, multi_class='ovr')
print("AUC:", auc)

# Reshape y_pred_prob to make it two-dimensional
y_true_binary = label_binarize(y_test, classes=[0, 1, 2])
y_pred_prob_reshaped = y_pred_prob.reshape(-1, 1)

# Calculate the ROC AUC score for each class
auc_roc = roc_auc_score(y_true_binary, y_pred_prob_reshaped, average='macro')
print("ROC AUC Score (OvR):", auc_roc)
```

```
AUC: 0.559237233200672
ROC AUC Score (OvR): 0.4387961901413984
```

7. Code to find the training and testing accuracy

```
# Predict the target variable using the fitted KNN model
y_train_pred = best_knn.predict(X_train)
y_test_pred = best_knn.predict(X_test)

# Calculate accuracy on training set
train_accuracy = accuracy_score(y_train, y_train_pred)
print("Training Accuracy:", train_accuracy)

# Calculate accuracy on testing set
test_accuracy = accuracy_score(y_test, y_test_pred)
print("Testing Accuracy:", test_accuracy)
```

```
Training Accuracy: 0.531625
Testing Accuracy: 0.5125
```

PART V: DATA SUMMARY & IMPLICATIONS

E1. ACCURACY AND AUC

Metric	Percentage Score
Accuracy	55.92%
Training Accuracy	53.16%
Testing Accuracy	51.25%
Area Under the Curve	55.92%

Accuracy measures the overall correctness of predictions. In this analysis, the accuracy is relatively low, suggesting that the model's predictions are not highly reliable. For the research question, a low accuracy means that the model might not effectively predict the initial administration among patients based on the provided features. An overall accuracy of 55.92% implies that approximately half the instances are classified correctly.

Training accuracy measures the model's performance on the data it was trained on. The training accuracy is 53.16% which is slightly higher than the testing accuracy. This suggests that the model might be overfitting to the training data. Overfitting occurs when the model learns too much from the noise in the training data. For the research question, overfitting could lead to overly favorable performance estimates and a lack of generalization to new patients.

Testing accuracy measures the model's performance on new and unseen data. It indicates how well the model generalizes to data it hasn't seen during training. The low testing accuracy of 51.25% suggests that the model's performance decreases when applied to new patients. For the research question, low testing accuracy means that the model might not be reliable when making predictions for new patients.

Area Under the Curve (AUC) measures the ability of the model to distinguish between different classes. For the research question, the low AUC of 55.92% indicates that the model's predictive power is limited. It might not be effective at differentiating between patients with different initial administration based on the provided features.

E2. RESULTS & IMPLICATIONS

The classification analysis provides valuable insights into the relationship between patient demographics and initial administration. However, the poor performance of the model underscores the need for continued research to develop more accurate predictive models.

The classification model exhibits a relatively low accuracy, training accuracy, testing accuracy, and AUC. This indicates that the model's ability to predict the initial administration among patients based on the provided features is limited. These metrics suggest that it struggles to effectively distinguish between different classes and generalize well to new and unseen data.

The suboptimal performance of the classification model highlights the need for further model refinement. This may involve refining the feature set, exploring different modeling techniques, or

incorporating domain expertise to better capture the complexities of patient demographics and medical conditions. Additionally, collecting additional relevant features or refining existing features could improve the model's ability to discriminate between different classes.

The limitations of the classification model have important clinical implications. Healthcare professionals rely on accurate predictive models to make informed decisions about patient care. A model with poor predictive performance could lead to incorrect decisions.

E3. LIMITATION

One limitation of the analysis is the class imbalance in the target variable, specifically the distribution of initial administration classes. In this analysis, Class 0 has 2504 instances, Class 1 has 5060 instances, and Class 2 has 2436. This imbalance can pose challenges during training and evaluation.

The imbalanced class distribution can lead to be biased toward the majority class (Class 1). As a result, the model may prioritize accuracy on the majority class while neglecting the minority classes, leading to poor predictive performance for those classes.

Imbalanced data can also hinder the model's ability to generalize well, as seen in this analysis. the model may struggle to learn meaningful patterns associated with minority classes due to their limited representation in the dataset.

E4. COURSE OF ACTION

Based on the results and implications of the classification analysis, I would not recommend this model to identify the key predictors for initial administration for patients.

My recommendation for the organization is to first address the significant class imbalance in the dataset. The organization should collect more data for the minority classes to achieve a more balanced distribution. This can involve targeted data collection effects to increase the representation of minority classes.

Moreover, the organization should implement a system for continuous monitoring and improvement of the classification model. It is essential to regularly update the model with new data, reevaluate its performance, and refine its features as needed to ensure its effectiveness.

PART VI. PANOPTO DEMONSTRATION

https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=4ad3f067-9b25-4e0d-a89f-b14a0167c761

SOURCES OF THIRD-PARTY CODES

https://westerngovernorsuniversity.sharepoint.com/:p:/r/sites/DataScienceTeam/_layouts/15/Doc.aspx?sourcedoc=%7B945F58A7-B99E-4D7A-BEC0-9BB216B4D2BD%7D&file=D209%20Data%20Mining%201%20Task%201%20Cohort.pptx&action=edit&mobileredirect=true

https://campus.datacamp.com/courses/supervised-learning-with-scikit-learn/classification-1?ex=1

SOURCES

No sources cited