Task 1: Classification Data Mining Models

Vulnerabilities of Web Servers

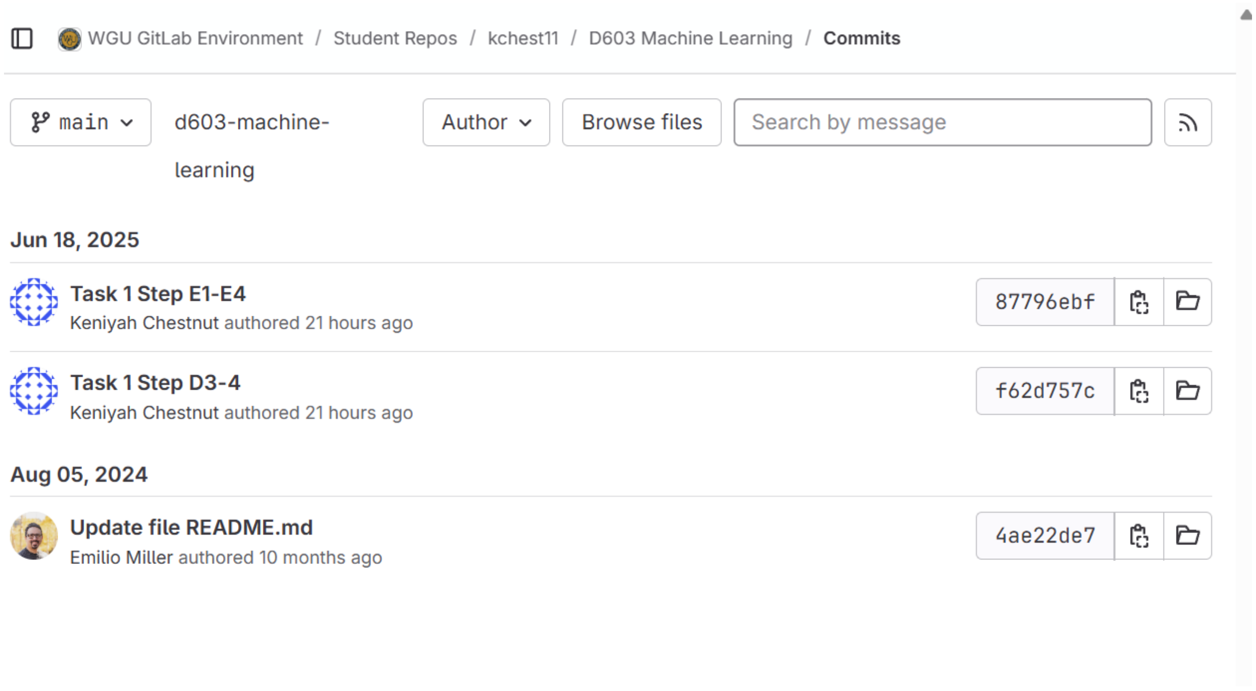
Keniyah Chestnut

Machine Learning — D603

SID: 012601305

A: GITLAB REPOSITORY

I created a subgroup and project in GitLab using the provided link. I cloned the repo to my computer, worked in it locally, and pushed my changes as I completed each part of the task.



WGU GitLab Environment / Student Repos / kchest11 / D603 Machine Learning / Commits

Jun 18, 2025

Task 1 Step E1-E4
Keniyah Chestnut authored 21 hours ago
87796ebf

Task 1 Step D3-4
Keniyah Chestnut authored 21 hours ago
f62d757c

Aug 05, 2024

Update file README.md
Emilio Miller authored 10 months ago
4ae22de7

B1:PROPOSAL OF QUESTION

Is it possible to predict whether a patient will be readmitted to the hospital based on their demographic information and medical history using the Random Forest classification method?

B2:DEFINED GOAL

The goal of this analysis is to use the Random Forest classification method to identify key factors in predicting whether or not a patient will be readmitted to the hospital. While this insight could be used to help reduce costs or increase efficiency, the primary goal is to improve patient care. By predicting the likelihood of readmittance, the hospital can take proactive steps to support at-risk patients and reduce unnecessary returns.

C1:EXPLANATION OF CLASSIFICATION METHOD

I chose the Random Forest classification method because it is a powerful ensemble learning algorithm that builds multiple decision trees during training. Each tree is trained on a different random subset of the data, and at each split, it considers a random subset of features. This randomness helps reduce overfitting and makes the model more accurate and reliable.

In my analysis, each tree in the forest makes a prediction, and the model takes a majority vote to decide the final classification. I found this approach to be well-suited for my dataset, which includes a mix of categorical and continuous variables such as age, gender, and medical history. Random Forest can handle these types of variables effectively without much preprocessing.

I expect this model to accurately predict patient readmission and help identify key factors that contribute to it. I also like that Random Forest provides feature importance scores, which will allow me to see which variables have the most impact on the outcome and guide recommendations for healthcare decision-makers.

C2:PACKAGES OR LIBRARIES LIST

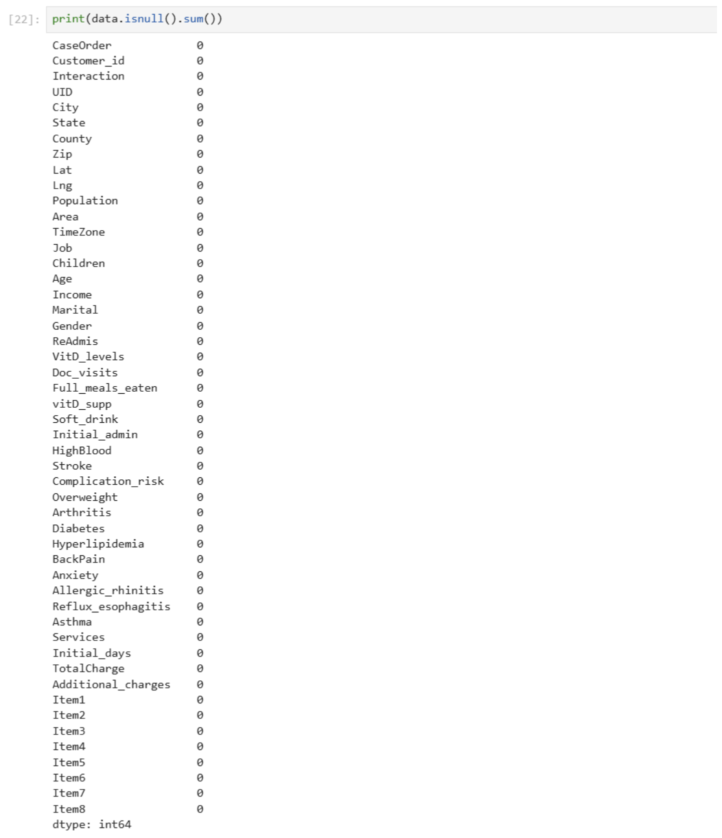
To complete this analysis, I used several Python libraries that made it easier to clean the data, build the model, and evaluate the results.

1. **Pandas** – This library was used to load and clean the dataset. It helped organize the information into a format that could be used for machine learning and made it easy to filter, replace, and preprocess the data.
2. **NumPy** – NumPy was used for working with numerical data and performing calculations behind the scenes. It works well with Pandas and is a standard part of most machine learning projects.
3. **Scikit-learn (sklearn)** – This was the main machine learning library used in the project. Specific tools from this library included:
    - train_test_split to divide the dataset into training, validation, and test sets
    - RandomForestClassifier to build and train the prediction model
    - GridSearchCV to tune hyperparameters and improve the model
    - classification_report, confusion_matrix, and roc_auc_score to evaluate how well the model performed

These libraries worked together to allow for a smooth and efficient analysis. Each one played an important role in helping to clean the data, train the model, and make accurate predictions.

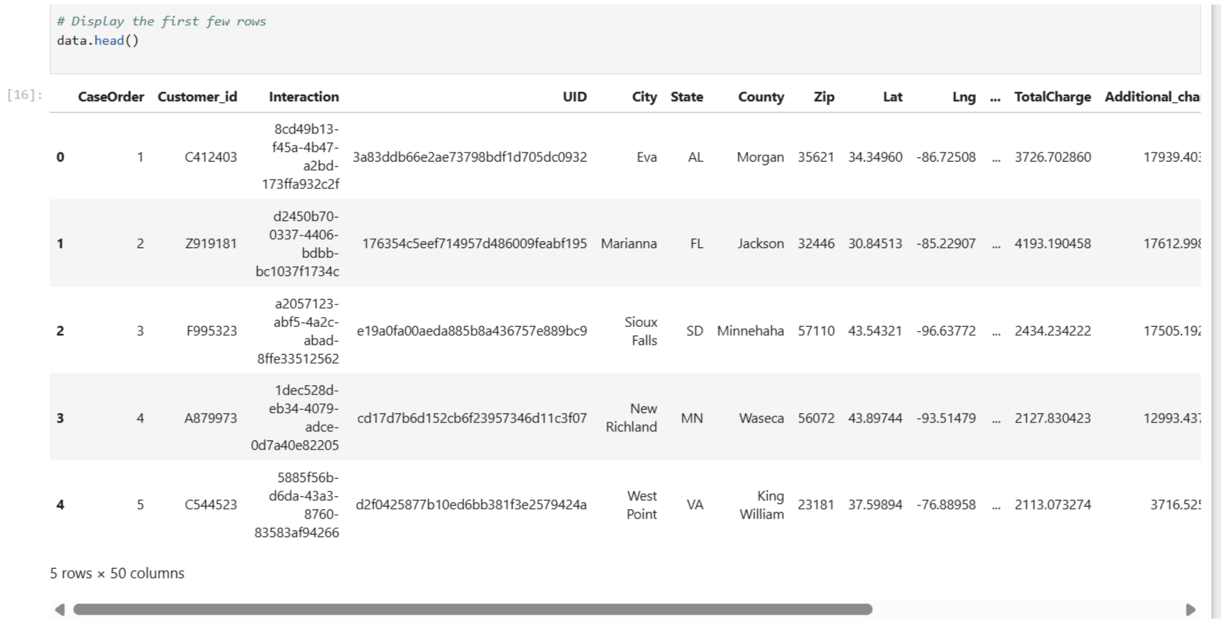
D1:DATA PREPROCESSING

To get the data ready for analysis, I started by checking for any missing values and confirming that the dataset was clean. I also removed any extra spaces in the column headers to prevent errors when referencing column names in the code.



```
[22]: print(data.isnull().sum())

CaseOrder             0
Customer_id           0
Interaction           0
UID                   0
City                  0
State                 0
County                0
Zip                   0
Lat                   0
Lng                   0
Population            0
Area                  0
TimeZone              0
Job                   0
Children              0
Age                   0
Income                0
Marital               0
Gender                0
ReAdmis               0
VitD_levels           0
Doc_visits            0
Full_meals_eaten      0
vitD_supp             0
Soft_drink            0
Initial_admin         0
HighBlood             0
Stroke                0
Complication_risk     0
Overweight            0
Arthritis             0
Diabetes              0
Hyperlipidemia        0
BackPain              0
Anxiety               0
Allergic_rhinitis     0
Reflux_esophagitis    0
Asthma                0
Services              0
Initial_days          0
TotalCharge           0
Additional_charges    0
Item1                 0
Item2                 0
Item3                 0
Item4                 0
Item5                 0
Item6                 0
Item7                 0
Item8                 0
dtype: int64
```

Next, I reviewed the dataset and removed columns that were not directly related to patient demographics or medical history, since they would not help with predicting readmission.



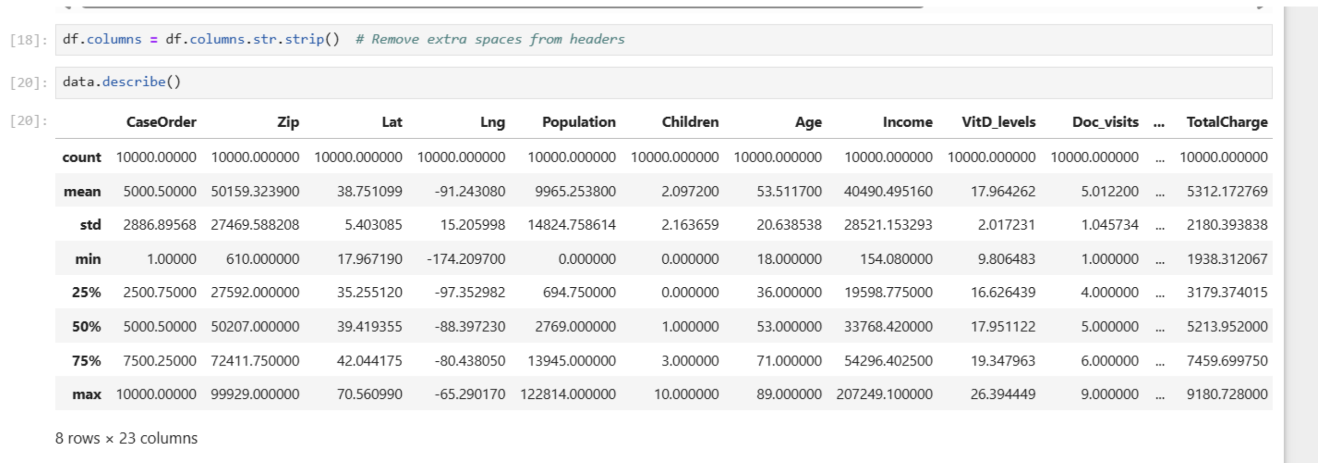
```
# Display the first few rows
data.head()
```

[16]:

	CaseOrder	Customer_id	Interaction	UID	City	State	County	Zip	Lat	Lng	...	TotalCharge	Additional_cha
0	1	C412403	8cd49b13-f45a-4b47-a2bd-173ffa932c2f	3a83ddb66e2ae73798bdf1d705dc0932	Eva	AL	Morgan	35621	34.34960	-86.72508	...	3726.702860	17939.40
1	2	Z919181	d2450b70-0337-4406-bdbb-bc1037f1734c	176354c5eef714957d486009feabf195	Marianna	FL	Jackson	32446	30.84513	-85.22907	...	4193.190458	17612.99
2	3	F995323	a2057123-abf5-4a2c-abad-8ffe33512562	e19a0fa00aeda885b8a436757e889bc9	Sioux Falls	SD	Minnehaha	57110	43.54321	-96.63772	...	2434.234222	17505.19
3	4	A879973	1dec528d-eb34-4079-adce-0d7a40e82205	cd17d7b6d152cb6f23957346d11c3f07	New Richland	MN	Waseca	56072	43.89744	-93.51479	...	2127.830423	12993.43
4	5	C544523	5885f56b-d6da-43a3-8760-83583af94266	d2f0425877b10ed6bb381f3e2579424a	West Point	VA	King William	23181	37.59894	-76.88958	...	2113.073274	3716.52

5 rows × 50 columns

After narrowing down the relevant variables, I encoded all categorical values into numerical format because the Random Forest model requires numeric input. The details and code for these steps are explained in sections D2 and D3.



```
[18]: df.columns = df.columns.str.strip()  # Remove extra spaces from headers
```

```
[20]: data.describe()
```

[20]:

	CaseOrder	Zip	Lat	Lng	Population	Children	Age	Income	VitD_levels	Doc_visits	...	TotalCharge
count	10000.00000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	...	10000.000000
mean	5000.50000	50159.323900	38.751099	-91.243080	9965.253800	2.097200	53.511700	40490.495160	17.964262	5.012200	...	5312.172769
std	2886.89568	27469.588208	5.403085	15.205998	14824.758614	2.163659	20.638538	28521.153293	2.017231	1.045734	...	2180.393838
min	1.00000	610.000000	17.967190	-174.209700	0.000000	0.000000	18.000000	154.080000	9.806483	1.000000	...	1938.312067
25%	2500.75000	27592.000000	35.255120	-97.352982	694.750000	0.000000	36.000000	19598.775000	16.626439	4.000000	...	3179.374015
50%	5000.50000	50207.000000	39.419355	-88.397230	2769.000000	1.000000	53.000000	33768.420000	17.951122	5.000000	...	5213.952000
75%	7500.25000	72411.750000	42.044175	-80.438050	13945.000000	3.000000	71.000000	54296.402500	19.347963	6.000000	...	7459.699750
max	10000.00000	99929.000000	70.560990	-65.290170	122814.000000	10.000000	89.000000	207249.100000	26.394449	9.000000	...	9180.728000

8 rows × 23 columns

D2:DATASET VARIABLES

To focus my analysis, I removed any columns that weren't directly related to patient demographics or medical history. I then organized the remaining variables into two categories: categorical and continuous, based on how the data is structured and how I plan to use it in the model.

I initially considered the survey response variables (Item1 through Item8) as continuous since they were numeric, but upon review, I recognize these are better classified as ordinal categorical variables. They represent ordered preferences rather than truly continuous measures.

Categorical (non-numerical):

- Gender
- ReAdmis
- Soft_drink
- Initial_admin
- HighBlood
- Stroke
- Overweight
- Arthritis
- Diabetes
- Hyperlipidemia
- BackPain
- Anxiety
- Allergic_rhinitis
- Reflux_esophagitis
- Asthma
- Services
- Complication_risk
- Item1 through Item8 (ordinal categorical)

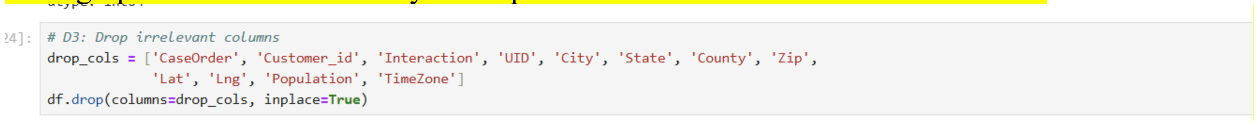
Continuous Variables:

- Income
- VitD_levels
- Doc_visits
- Initial_days
- TotalCharge

- Additional_charges
- Full_meals_eaten
- vitD_supp
- Children
- Age

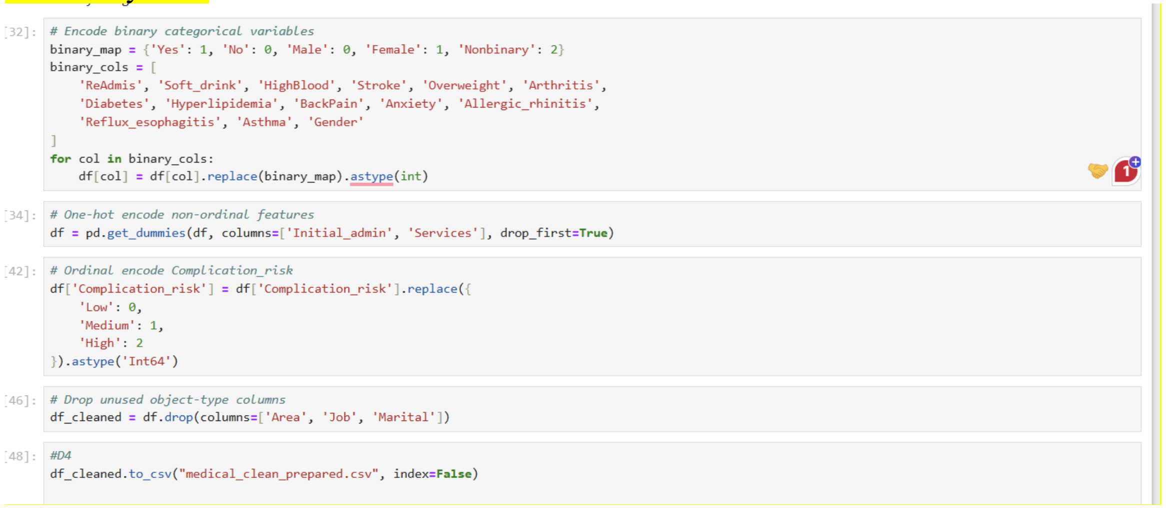
D3:STEPS FOR ANALYSIS

To prepare the dataset for analysis, I began by checking for missing values and cleaning up the column names to remove extra spaces. I then dropped columns that were unrelated to patient demographics or medical history to keep the dataset focused on relevant features.



```
# D3: Drop irrelevant columns
drop_cols = ['CaseOrder', 'Customer_id', 'Interaction', 'UID', 'City', 'State', 'County', 'Zip',
             'Lat', 'Lng', 'Population', 'TimeZone']
df.drop(columns=drop_cols, inplace=True)
```

Next, I encoded the categorical variables to prepare them for the Random Forest model, which requires all inputs to be numerical. I started by converting all Yes/No responses into binary values (1 for "Yes", 0 for "No"). For the Gender column, I encoded Male as 0, Female as 1, and Nonbinary as 2.



```
# Encode binary categorical variables
binary_map = {'Yes': 1, 'No': 0, 'Male': 0, 'Female': 1, 'Nonbinary': 2}
binary_cols = [
    'ReAdmis', 'Soft_drink', 'HighBlood', 'Stroke', 'Overweight', 'Arthritis',
    'Diabetes', 'Hyperlipidemia', 'BackPain', 'Anxiety', 'Allergic_rhinitis',
    'Reflux_esophagitis', 'Asthma', 'Gender'
]
for col in binary_cols:
    df[col] = df[col].replace(binary_map).astype(int)
```

```
# One-hot encode non-ordinal features
df = pd.get_dummies(df, columns=['Initial_admin', 'Services'], drop_first=True)
```

```
# Ordinal encode Complication_risk
df['Complication_risk'] = df['Complication_risk'].replace({
    'Low': 0,
    'Medium': 1,
    'High': 2
}).astype('Int64')
```

```
# Drop unused object-type columns
df_cleaned = df.drop(columns=['Area', 'Job', 'Marital'])
```

```
#D4
df_cleaned.to_csv("medical_clean_prepared.csv", index=False)
```

For categorical variables without a natural order like Initial_admin and Services, I applied one-hot encoding without dropping any columns. Since Random Forest is not affected by multicollinearity, I avoided using the k−1 rule (dropping one dummy column), which is typically done for linear models.

For Complication_risk, which has ordered categories (Low, Medium, High), I used ordinal encoding to preserve the natural rank of the values.

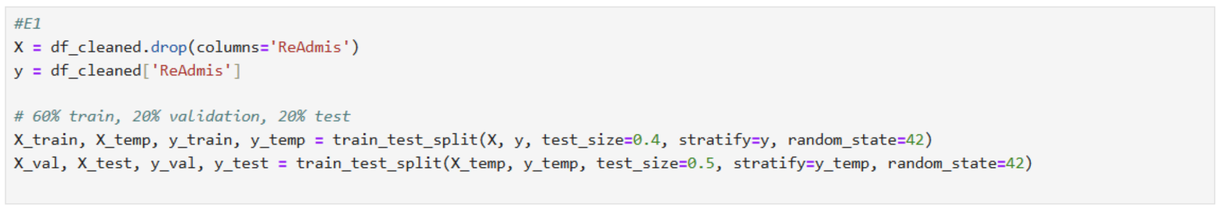
The final cleaned dataset was saved, and from there I proceeded to split the data to begin model training. The requirement for **D4: CLEANED DATASET** is satisfied with the submission of the processed file.

E1:SPLITTING THE DATA

To prepare the dataset for training and evaluation, I split the cleaned data into three separate sets: training, validation, and test sets. This allows the model to be trained on one portion of the data, tuned using another portion, and finally evaluated on completely unseen data to assess how well it generalizes.

I used a 60/20/20 split. The training set contains 60% of the data and is used to train the initial Random Forest model. The validation set, which contains 20%, is used for tuning hyperparameters during Grid Search. The final 20% is held out as the test set and used only to evaluate the final optimized model.

The code used to perform this split is shown below:



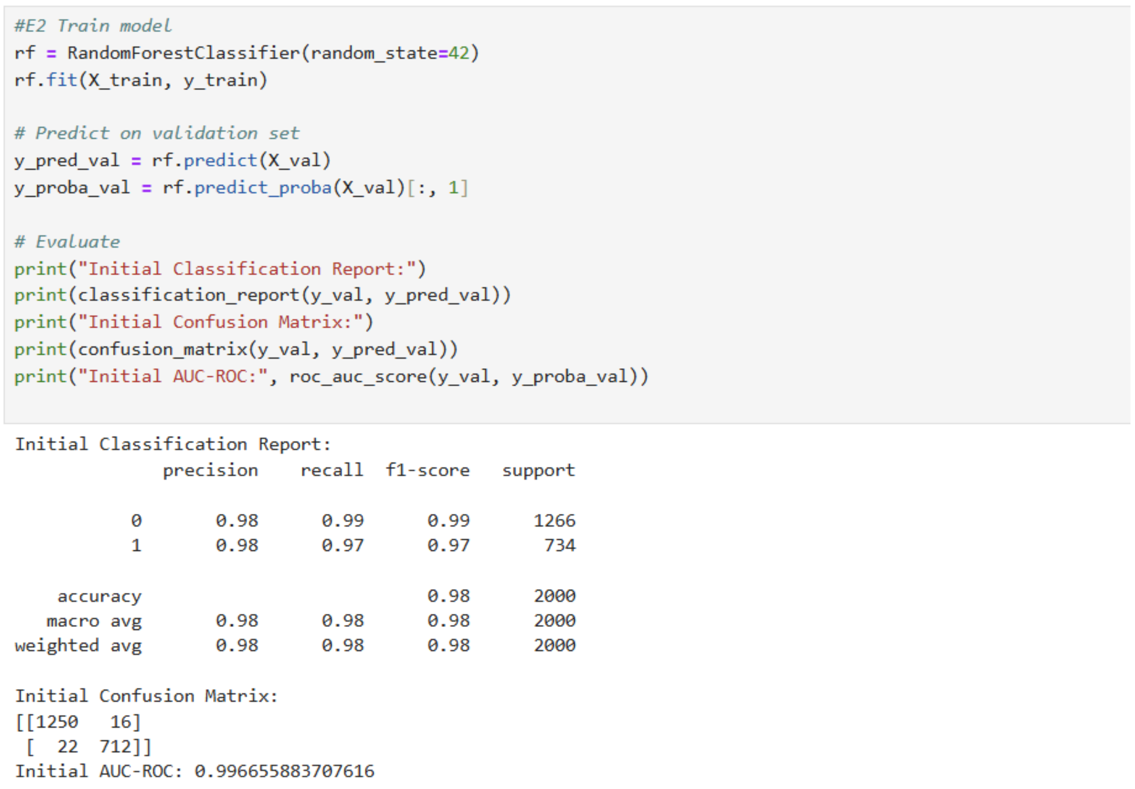
```
#E1
X = df_cleaned.drop(columns='ReAdmis')
y = df_cleaned['ReAdmis']

# 60% train, 20% validation, 20% test
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.4, stratify=y, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, stratify=y_temp, random_state=42)
```

E2:INITIAL MODEL CREATION

With the data split and preprocessed, I began by creating the initial Random Forest model using the training dataset. This model was built without any hyperparameter tuning to establish a performance baseline.

I trained the model using the default settings provided by the RandomForestClassifier in scikit-learn. After training, I evaluated the model on the validation set using several key classification metrics, including accuracy, precision, recall, F1 score, AUC-ROC, and the confusion matrix.

The code for training and evaluating the initial model is shown below:



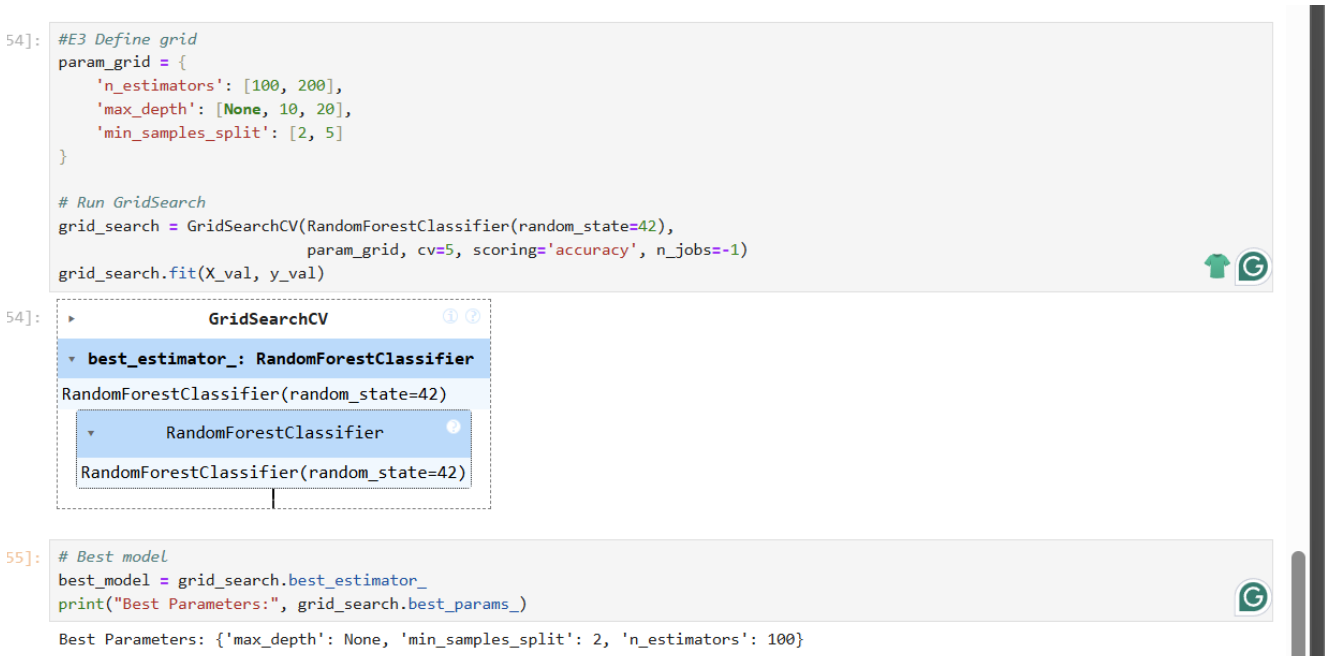
```python
#E2 Train model
rf = RandomForestClassifier(random_state=42)
rf.fit(X_train, y_train)

# Predict on validation set
y_pred_val = rf.predict(X_val)
y_proba_val = rf.predict_proba(X_val)[:, 1]

# Evaluate
print("Initial Classification Report:")
print(classification_report(y_val, y_pred_val))
print("Initial Confusion Matrix:")
print(confusion_matrix(y_val, y_pred_val))
print("Initial AUC-ROC:", roc_auc_score(y_val, y_proba_val))
```

```
Initial Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.99      0.99      1266
           1       0.98      0.97      0.97       734

    accuracy                           0.98      2000
   macro avg       0.98      0.98      0.98      2000
weighted avg       0.98      0.98      0.98      2000

Initial Confusion Matrix:
[[1250   16]
 [  22  712]]
Initial AUC-ROC: 0.996655883707616
```

E3:HYPERPARAMETER TUNING

To tune the Random Forest model, I used GridSearchCV to test combinations of key hyperparameters. The goal was to find a configuration that improved performance without overfitting.

The grid of parameters tested is shown below:



```
54]: #E3 Define grid
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5]
}

# Run GridSearch
grid_search = GridSearchCV(RandomForestClassifier(random_state=42),
                           param_grid, cv=5, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_val, y_val)
```

```
54]:  GridSearchCV
      best_estimator_: RandomForestClassifier
      RandomForestClassifier(random_state=42)
        RandomForestClassifier
        RandomForestClassifier(random_state=42)
```

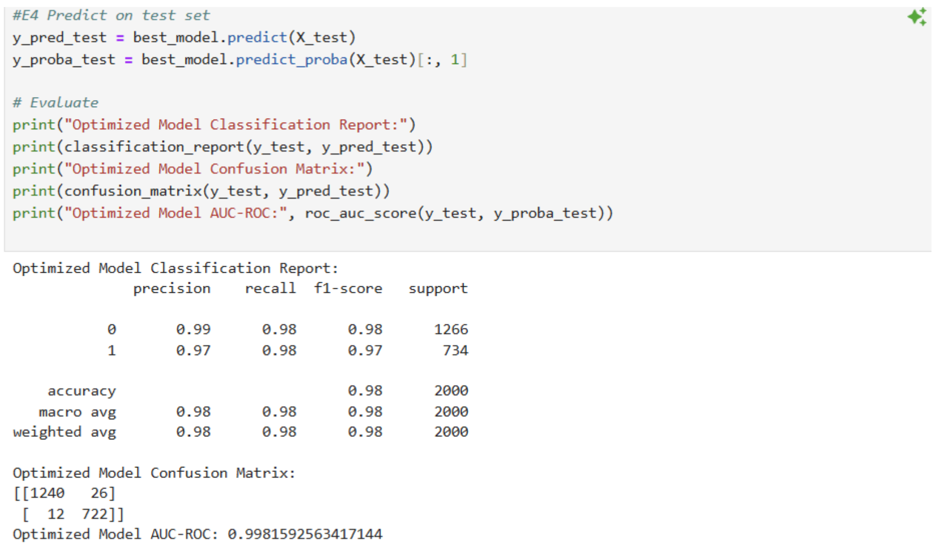
```
55]: # Best model
best_model = grid_search.best_estimator_
print("Best Parameters:", grid_search.best_params_)

Best Parameters: {'max_depth': None, 'min_samples_split': 2, 'n_estimators': 100}
```

E4:PREDICTIONS

After running the optimized model using the best hyperparameters found in Grid Search, I evaluated its performance on the test dataset. This final step allowed me to measure how well the model performs on completely unseen data, ensuring the results are realistic and not overfitted.

The same evaluation metrics were used: accuracy, precision, recall, F1 score, AUC-ROC, and the confusion matrix.

Here is the code used for making predictions and evaluating the final mode



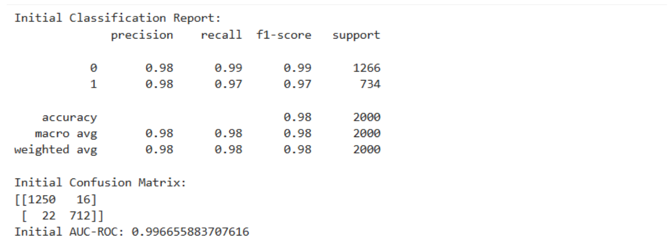
```python
#E4 Predict on test set
y_pred_test = best_model.predict(X_test)
y_proba_test = best_model.predict_proba(X_test)[:, 1]

# Evaluate
print("Optimized Model Classification Report:")
print(classification_report(y_test, y_pred_test))
print("Optimized Model Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_test))
print("Optimized Model AUC-ROC:", roc_auc_score(y_test, y_proba_test))
```

```
Optimized Model Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.98      0.98      1266
           1       0.97      0.98      0.97       734

    accuracy                           0.98      2000
   macro avg       0.98      0.98      0.98      2000
weighted avg       0.98      0.98      0.98      2000

Optimized Model Confusion Matrix:
[[1240   26]
 [  12  722]]
Optimized Model AUC-ROC: 0.9981592563417144
```

F1:MODEL EVALUATION

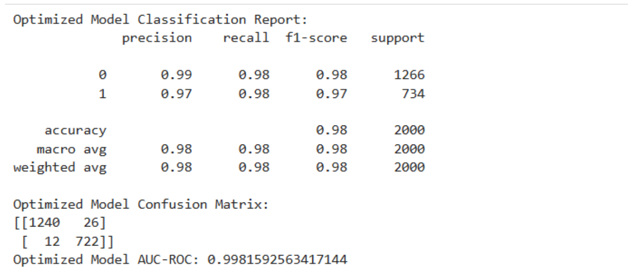
Here is a side-by-side comparison of the key evaluation metrics between the initial model and the optimized model. These include accuracy, precision, recall, F1 score, and AUC-ROC.

Initial:



```
Initial Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.99      0.99      1266
           1       0.98      0.97      0.97       734

    accuracy                           0.98      2000
   macro avg       0.98      0.98      0.98      2000
weighted avg       0.98      0.98      0.98      2000

Initial Confusion Matrix:
[[1250   16]
 [  22  712]]
Initial AUC-ROC: 0.996655883707616
```

Optimized:



```
Optimized Model Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.98      0.98      1266
           1       0.97      0.98      0.97       734

    accuracy                           0.98      2000
   macro avg       0.98      0.98      0.98      2000
weighted avg       0.98      0.98      0.98      2000

Optimized Model Confusion Matrix:
[[1240   26]
 [  12  722]]
Optimized Model AUC-ROC: 0.9981592563417144
```

1. To look at the actual changes in Accuracy did not notably improve, but I calculated the improvement at 0.05 percent since it is less than the rounding value.

2. Precision, recall, and F1 score are all unchanged in the weighted average. There are some small variations in the positive and negative class, but the changes are not significant overall.

3. AUC-ROC improved by 0.15 percent, increasing from 0.9967 to 0.9982.

the model and avoid making conclusions based on rounding, we need to observe the confusion matrices. In this case, the optimization was successful in moving one data point from the false positives to the true positives. That is the extent of the improvement. In other words, out of 50 misattributed data points, the optimized model corrected one. Because the model was already highly accurate, the improvements are fairly minimal, but still represent a slight gain in performance.

F2:RESULTS AND IMPLICATIONS

The results of this analysis are twofold. First, the model developed is able to predict hospital readmittance with an impressive 98 percent accuracy. Given a new patient's demographic and medical data, we can now make highly reliable predictions about whether they are likely to be readmitted. This allows the hospital to offer better care and help patients manage expectations about their recovery and long-term treatment.

In addition, the hospital could use this information to identify the most influential factors that contribute to readmittance and design targeted healthcare plans to reduce risk. Second, the results suggest that hospital readmittance is not random. Instead, there are strong and predictable patterns in the data, meaning that with the right information, future readmittance can be anticipated and managed.

F3:LIMITATION

While the model is highly accurate, there are a few limitations to keep in mind. One concern is whether the dataset is truly representative of all patients. If the original data is not diverse or comprehensive enough, the model's performance may drop when applied to new populations. Although the model currently shows 98 percent accuracy, it is important to test it on new data to ensure that it has not been overfit.

Another concern is the quality and potential bias in the data. If certain groups were over- or underrepresented, the model could inherit those biases. To maintain reliability, ongoing evaluation and retraining with new, diverse data will be necessary.

F4:COURSE OF ACTION

The goal of this project was to develop a model that could predict patient readmittance based on demographic and medical history data. Now that the model has proven to be highly accurate, it should be used to directly benefit patients and healthcare planning.

This model can be used to inform patients of their likelihood of being readmitted. That can help manage expectations and improve transparency in their care. Additionally, by identifying the strongest predictors of readmittance, healthcare providers can work with patients to address those risk factors. With this insight, hospitals can implement personalized intervention strategies to reduce readmittance and improve patient outcomes.

References

All data and supporting materials were provided by Western Governors University (WGU) for the purposes of academic analysis and student assessment. No external sources were used.