CSC108_HarmonyHealthcare_Final

December 14, 2025

Mike and Spencer

0.0.1 Beyond the Emergency Department: Predictive Analytics for 6-Month Patient Admission in Harmony Healthcare

0.0.2 Problem Statement

Our goal is to build a predictive model that identifies whether a patient will experience at least one ED admission within the next six months. Based on instructor feedback, we removed columns directly tied to emergency department data, forcing the model to learn broader risk signals rather than simply reproducing ED activity. This notebook documents data cleaning, exploratory visualization, and an initial predictive model using forward feature selection with logistic regression.

[ ]:
```
# from google.colab import drive
import pandas as pd
# drive.mount('/content/drive')
# Then, update the path in pd.read_excel, for example:
# df = pd.read_excel('/content/drive/MyDrive/Data Science CSC 108/
 ↪HarmonyHealthcareOneWeek_9_2025.xlsx')

df = pd.read_excel('HarmonyHealthcareOneWeek_9_2025.xlsx')

# We want to keep the 'ED Episode Admit Last-6-Mths' column and remove all␣
 ↪other columns with ED in the name
ed_columns = [col for col in df.columns if 'ED' in col and col != 'ED Episode␣
 ↪Admit Last-6-Mths']
df = df.drop(columns=ed_columns)

# Assuming NaN in 'ED Episode Admit Last-6-Mths' means no admission, fill with␣
 ↪0 early
target = 'ED Episode Admit Last-6-Mths'
df[target] = df[target].fillna(0)

# Now we can remove any columns that are just empty to shrink the data further
df = df.dropna(axis=1, how='all')
df.head()
```

[ ]:
```
target = 'ED Episode Admit Last-6-Mths' # Store our target var so we can use it later

# Check target distribution
print(df[target].value_counts())
print(df[target].value_counts(normalize=True) * 100)

# Check missing percentages
missing_pct = (df.isna().mean() * 100).sort_values(ascending=False)
print(missing_pct[missing_pct > 0])
```

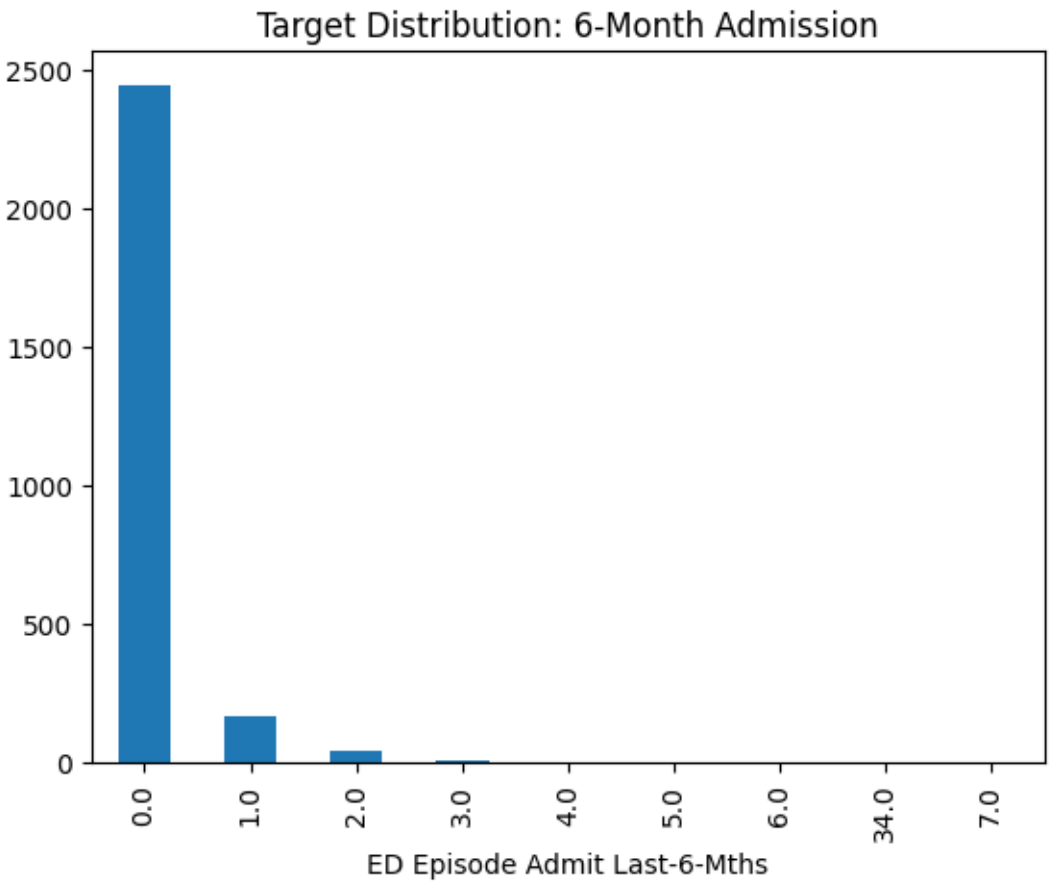
[18]:
```
# Drop columns with >70% missing, except target
columns_to_drop = [c for c in df.columns if c != target and df[c].isna().mean() * 100 > 70]
print("Number of columns to drop:", len(columns_to_drop))
df = df.drop(columns=columns_to_drop)
# df.head()
```

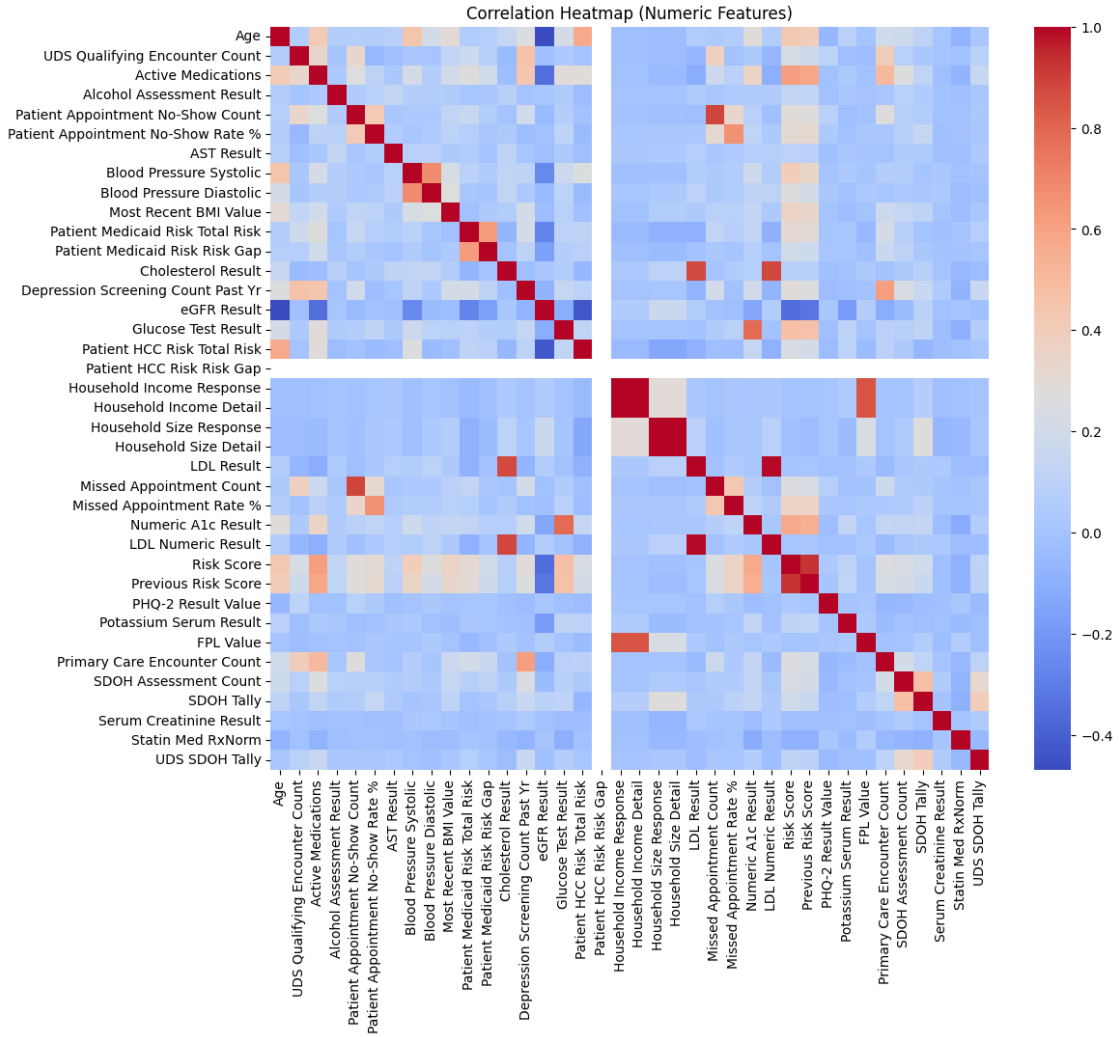
```
Number of columns to drop: 417
```

[19]:
```
# Impute missing values + encode categoricals

from sklearn.impute import SimpleImputer

target_col = 'ED Episode Admit Last-6-Mths' # Define target here too for consistency

# Separate numerical + categorical
numeric_cols = df.select_dtypes(include='number').columns.tolist()
# Exclude the target column from numeric imputation
if target_col in numeric_cols:
    numeric_cols.remove(target_col)

cat_cols = df.select_dtypes(include=['object', 'category']).columns.tolist()

# Impute numerics with median
num_imputer = SimpleImputer(strategy='median')
df[numeric_cols] = num_imputer.fit_transform(df[numeric_cols])

# Impute categoricals with most frequent
cat_imputer = SimpleImputer(strategy='most_frequent')
df[cat_cols] = cat_imputer.fit_transform(df[cat_cols])

# One-hot encode categoricals
df = pd.get_dummies(df, columns=cat_cols, drop_first=True)

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2672 entries, 0 to 2671
Columns: 7049 entries, Age to Vision Screening Code_Z01.01
dtypes: bool(6970), datetime64[ns](40), float64(39)
memory usage: 19.4 MB
```

[20]:
```python
# Visualizations!

import matplotlib.pyplot as plt
import seaborn as sns

# 1. Target distribution
df[target].value_counts().plot(kind='bar')
plt.title("Target Distribution: 6-Month Admission")
plt.show()

# 2. Histogram of top numeric features
# df[numeric_cols].hist(figsize=(12,10))
# plt.tight_layout()
# plt.show()

# 3. Correlation heatmap
plt.figure(figsize=(12,10))
sns.heatmap(df[numeric_cols].corr(), cmap='coolwarm')
plt.title("Correlation Heatmap (Numeric Features)")
plt.show()
```


Target Distribution: 6-Month Admission
2500
2000
1500
1000
500
0
0.0
1.0
2.0
3.0
4.0
5.0
6.0
34.0
7.0
ED Episode Admit Last-6-Mths



[21]:
```
# Convert target to binary (admitted = 1 if >0)
y = (df[target] > 0).astype(int)
X = df.drop(columns=[target])
```

[22]:
```
# Clean and prepare data

# Make a copy of the dataframe to avoid modifying the original `df` from the␣
 ↪previous cell.
df_processed = df.copy()

# Target column name
target_col_name = "ED Episode Admit Last-6-Mths"

# Isolate the target variable BEFORE any potentially destructive␣
 ↪transformations.
```

```
# Apply fillna(0) and >0 to ensure binary classification, and then convert to␣
 ↪int.
y = (df_processed[target_col_name].fillna(0) > 0).astype(int)

# Drop the target column from the feature set X
X = df_processed.drop(columns=[target_col_name])

# Identify columns that are actual datetime objects in X.
# These are the columns from `df.info()` with dtypes `datetime64[ns]` from the␣
 ↪previous step.
datetime_cols_in_X = X.select_dtypes(include=['datetime64[ns]']).columns.
 ↪tolist()

# Convert identified datetime columns in X to numeric days since epoch.
# This avoids incorrectly converting other numeric columns.
for col in datetime_cols_in_X:
    X[col] = (X[col] - pd.Timestamp("1970-01-01")).dt.days

# Replace remaining missing numerical values in X with column medians.
# This imputation step is for the features (X) after date conversion.
X = X.fillna(X.median(numeric_only=True))

# Train/test split
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# Standardize numerical data
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

[23]:
```
# Logistic Regression + evaluation function
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score

def evaluate(feature_list):
    idxs = [X.columns.get_loc(f) for f in feature_list]

    model = LogisticRegression(
        max_iter=500,
        class_weight="balanced"
    )
```

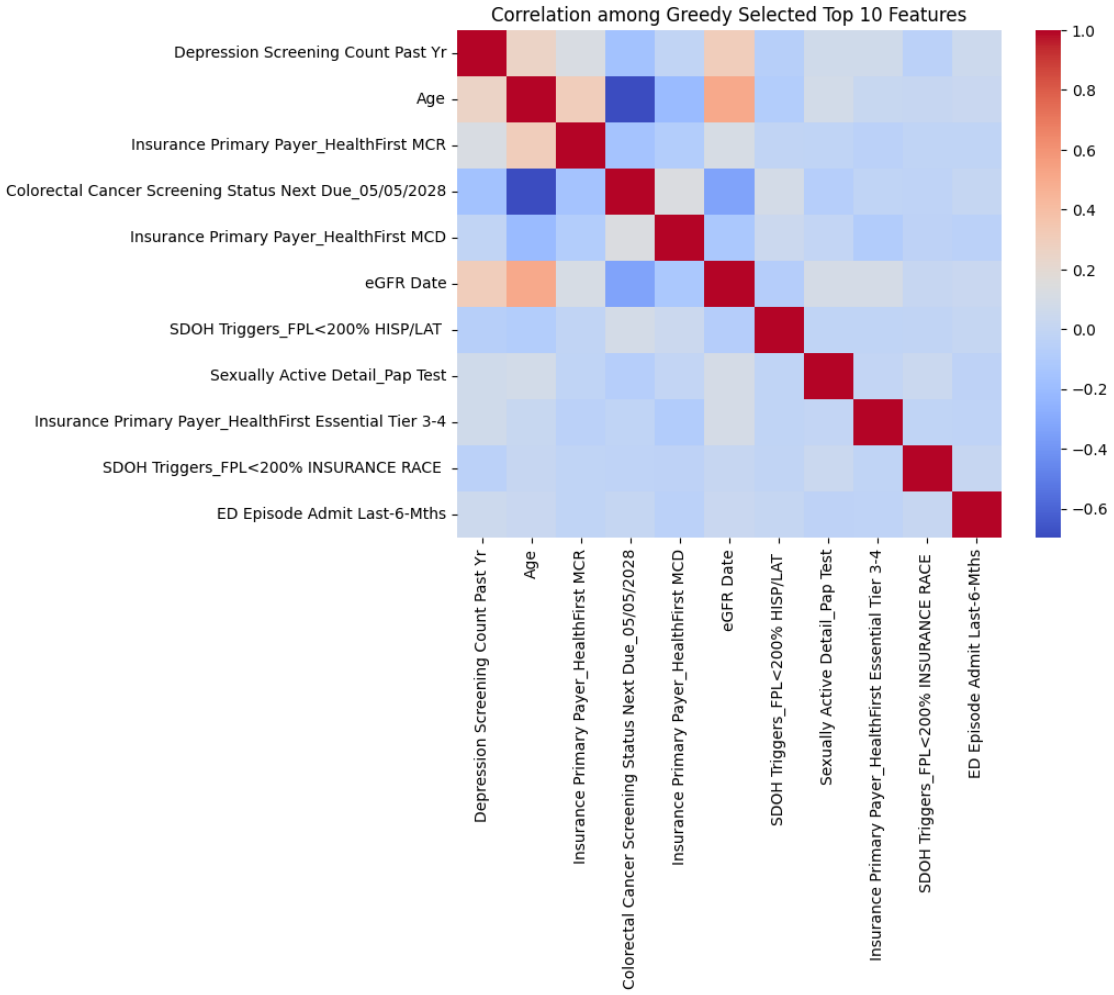
```
    model.fit(X_train_scaled[:, idxs], y_train)

    preds = model.predict_proba(X_test_scaled[:, idxs])[:, 1]

    return roc_auc_score(y_test, preds)
```

[24]:
```
# Greedy forward feature selection

# Initialize lists: 'remaining' holds features not yet selected, 'selected' holds chosen features,
# and 'scores' stores the AUC for the selected feature set at each step.
remaining = list(X.columns)
selected = []
scores = []

# Perform 10 steps of forward feature selection.
# In each step, we find the single best feature to add to our 'selected' set.
for step in range(10):
    best_feature = None
    best_auc = -1 # Initialize with a low AUC score to ensure the first valid AUC is higher

    # Iterate through all features not yet selected to find the one that maximizes AUC when added.
    for feat in remaining:
        # Create a temporary list of features that includes currently selected features plus one 'candidate' feature.
        try_features = selected + [feat]
        # Evaluate the performance (AUC) of the model using this candidate set of features.
        auc = evaluate(try_features)

        # If this candidate set yields a better AUC than the current best, update best_auc and best_feature.
        if auc > best_auc:
            best_auc = auc
            best_feature = feat

    # Add the best performing feature from this step to the 'selected' list.
    selected.append(best_feature)
    # Record the AUC achieved with this new set of selected features.
    scores.append(best_auc)
    # Remove the selected feature from the 'remaining' list so it's not considered again.
    remaining.remove(best_feature)
```

```
    # Print the result for the current step.
    # print(f"Step {step+1}: Selected {best_feature} - AUC {best_auc:.4f}")

# After all steps are complete, print the final list of top 10 selected␣
 ↪features and their corresponding AUC scores.
print("\nTop 10 selected features:")
for i, (feat, auc) in enumerate(zip(selected, scores), 1):
    print(f"{i}. {feat} - AUC {auc:.4f}")
```

```
Top 10 selected features:
1. Depression Screening Count Past Yr - AUC 0.6041
2. Age - AUC 0.6427
3. Insurance Primary Payer_HealthFirst MCR - AUC 0.6774
4. Colorectal Cancer Screening Status Next Due_05/05/2028 - AUC 0.7006
5. Insurance Primary Payer_HealthFirst MCD - AUC 0.7161
6. eGFR Date - AUC 0.7434
7. SDOH Triggers_FPL<200% HISP/LAT  - AUC 0.7624
8. Sexually Active Detail_Pap Test - AUC 0.7767
9. Insurance Primary Payer_HealthFirst Essential Tier 3-4 - AUC 0.7881
10. SDOH Triggers_FPL<200% INSURANCE RACE  - AUC 0.7979
```

[25]:
```
# Are the top 10 selected features correlated to each other?
selected_df = df[selected + [target]]
corr = selected_df.corr()
plt.figure(figsize=(8,6))
sns.heatmap(corr, annot=False, cmap="coolwarm")
plt.title("Correlation among Greedy Selected Top 10 Features")
plt.show()
```



[26]:
```
# 5 fold Cross-Validation

from sklearn.model_selection import StratifiedKFold, cross_val_score
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
import numpy as np

# Use only the greedy-selected top 10 features
X_selected = X[selected]

# Build pipeline to avoid data leakage
pipeline = Pipeline([
    ("scaler", StandardScaler()),
    ("logreg", LogisticRegression(
        max_iter=500,
```

```
        class_weight="balanced"
    ))
])

# 5-fold stratified CV
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

cv_scores = cross_val_score(
    pipeline,
    X_selected,
    y,
    cv=cv,
    scoring="roc_auc"
)

print("5-fold CV AUC scores:", cv_scores)
print("Mean CV AUC:", np.mean(cv_scores))
print("Std CV AUC:", np.std(cv_scores))
```

```
5-fold CV AUC scores: [0.68913933 0.64506091 0.62050071 0.66932466 0.68208749]
Mean CV AUC: 0.6612226192922849
Std CV AUC: 0.025297443168453084
```

[27]:
```
from sklearn.linear_model import LogisticRegressionCV
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import roc_auc_score, accuracy_score
from sklearn.model_selection import cross_val_score, StratifiedKFold


def lasso_classification_comparison(X, y, greedy_selected_features, greedy_auc):

    # Standardize features
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)
    X_scaled = pd.DataFrame(X_scaled, columns=X.columns)

    # Use LogisticRegressionCV with L1 penalty (Lasso for classification)
    # C is inverse of regularization strength (smaller C = more regularization)
    Cs = np.logspace(-4, 2, 20)  # Test range of regularization strengths

    # add verbose=1 to see progress
    lasso_logistic = LogisticRegressionCV(
        Cs=Cs,
        cv=5,
        penalty='l1',
        solver='liblinear',  # Required for L1 penalty
        max_iter=50000,
```

```
        class_weight='balanced',  # Handle class imbalance like your greedy␣
↪method
        scoring='roc_auc',  # Use AUC like your greedy method
        random_state=42,
        n_jobs=-1
    )

    print("\nFitting L1-regularized Logistic Regression...")
    lasso_logistic.fit(X_scaled, y)

    print(f"Optimal C (inverse regularization): {lasso_logistic.C_[0]:.6f}")
    print(f"  (Smaller C = more regularization, like larger alpha in Lasso)")

    # Get coefficients
    coefficients = pd.DataFrame({
        'feature': X.columns,
        'coefficient': lasso_logistic.coef_[0]
    })

    # Select non-zero features
    selected_features = coefficients[coefficients['coefficient'] != 0].copy()
    selected_features = selected_features.sort_values(
        'coefficient',
        key=abs,
        ascending=False
    )

    print(f"\nNumber of features selected: {len(selected_features)}/{len(X.
↪columns)}")
    print(f"Feature reduction: {(1 - len(selected_features)/len(X.columns))*100:
↪.1f}%")

    print("\nTop 20 selected features and coefficients:")
    print(selected_features.head(20).to_string(index=False))

    if len(selected_features) > 20:
        print(f"\n... and {len(selected_features) - 20} more features")

    # Calculate performance metrics using CLASSIFICATION metrics
    y_pred_proba = lasso_logistic.predict_proba(X_scaled)[:, 1]
    y_pred = lasso_logistic.predict(X_scaled)

    train_auc = roc_auc_score(y, y_pred_proba)
    train_acc = accuracy_score(y, y_pred)

    print(f"\nTRAIN Performance:")
    print(f"  AUC: {train_auc:.4f}")
```

```
    print(f"  Accuracy: {train_acc:.4f}")

    # Cross-validation with proper classification scoring
    cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
    cv_auc_scores = cross_val_score(
        lasso_logistic, X_scaled, y, cv=cv, scoring='roc_auc'
    )
    cv_acc_scores = cross_val_score(
        lasso_logistic, X_scaled, y, cv=cv, scoring='accuracy'
    )

    print(f"\nCROSS-VALIDATION Performance (5-fold):")
    print(f"  AUC: {cv_auc_scores.mean():.4f} ± {cv_auc_scores.std():.4f}")
    print(f"  Accuracy: {cv_acc_scores.mean():.4f} ± {cv_acc_scores.std():.4f}")

    # Comparison with Greedy Forward Selection
    print("\n" + "="*70)
    print("COMPARISON: L1 Logistic Regression vs Greedy Forward Selection")
    print("="*70)

    print(f"\nNumber of features:")
    print(f"  Greedy Forward Selection: {len(greedy_selected_features)}")
    print(f"  L1 Logistic (Lasso):      {len(selected_features)}")

    print(f"\nCross-Validation AUC:")
    print(f"  Greedy Forward Selection: {greedy_auc:.4f}")
    print(f"  L1 Logistic (Lasso):      {cv_auc_scores.mean():.4f} ±␣
↪{cv_auc_scores.std():.4f}")

    # Feature overlap analysis
    lasso_set = set(selected_features['feature'].tolist())
    greedy_set = set(greedy_selected_features)

    common = lasso_set & greedy_set
    only_lasso = lasso_set - greedy_set
    only_greedy = greedy_set - lasso_set

    print(f"\nFeature Overlap:")
    print(f"  Common features: {len(common)}/{len(greedy_set)}")
    if common:
        print(f"    {sorted(list(common)[:10])}")
        if len(common) > 10:
            print(f"    ... and {len(common) - 10} more")

    print(f"\n  Only in L1 Logistic: {len(only_lasso)}")
    if only_lasso and len(only_lasso) <= 10:
        print(f"    {sorted(list(only_lasso))}")
```

```
    elif only_lasso:
        print(f"    {sorted(list(only_lasso))[:10]}")
        print(f"    ... and {len(only_lasso) - 10} more")

    print(f"\n  Only in Greedy: {len(only_greedy)}")
    if only_greedy:
        print(f"    {sorted(list(only_greedy))}")

    return {
        'model': lasso_logistic,
        'scaler': scaler,
        'selected_features': selected_features,
        'feature_names': selected_features['feature'].tolist(),
        'train_auc': train_auc,
        'cv_auc_mean': cv_auc_scores.mean(),
        'cv_auc_std': cv_auc_scores.std(),
        'cv_auc_scores': cv_auc_scores,
        'optimal_C': lasso_logistic.C_[0]
    }


"""
# After the greedy forward selection completes you have:
# - selected: list of top 10 features from greedy
# - scores: list of AUC scores from greedy
# - X: your full feature matrix
# - y: your binary target
"""
greedy_best_auc = scores[-1]  # Last score is with all 10 features

lasso_results = lasso_classification_comparison(
    X=X,
    y=y,
    greedy_selected_features=selected,
    greedy_auc=greedy_best_auc
)
```

```
Fitting L1-regularized Logistic Regression…
Optimal C (inverse regularization): 0.297635
  (Smaller C = more regularization, like larger alpha in Lasso)

Number of features selected: 934/7048
Feature reduction: 86.7%

Top 20 selected features and coefficients:
```

```
                                                 feature  coefficient
Colorectal Cancer Screening Status Next Due_05/05/2028     0.590377
                           Blood Pressure Value_120/80     0.403904
                      FPL Date_5/12/2025 12:00:00 AM     0.358494
                      FPL Date_3/27/2025 12:00:00 AM     0.316371
                          Blood Pressure Vitals Date     0.294742
                         Blood Pressure Value_120/90     0.281138
                         Blood Pressure Value_107/71     0.269044
             Insurance Primary Payer_Bluecard Program     0.265444
                      FPL Date_7/25/2025 12:00:00 AM     0.254953
         Housing Situation-Date_9/16/2025 12:00:00 AM    -0.250033
   Insurance Primary Payer_Fidelis Essential Tier 3 -4     0.247067
Colorectal Cancer Screening Status Next Due_06/24/2007     0.239999
                Insurance Primary Payer_Metroplus CHP     0.238730
                         Blood Pressure Value_130/90     0.237610
                   Patient Appointment No-Show Rate %     0.235711
                      FPL Date_11/19/2024 12:00:00 AM     0.235564
Colorectal Cancer Screening Status Next Due_07/10/2027     0.231628
                      FPL Date_7/22/2025 12:00:00 AM     0.231442
                       FPL Date_4/1/2025 12:00:00 AM     0.228969
                                     EHR Sex_female     0.227478

… and 914 more features

TRAIN Performance:
  AUC: 1.0000
  Accuracy: 0.9993

CROSS-VALIDATION Performance (5-fold):
  AUC: 0.5253 ± 0.0521
  Accuracy: 0.8645 ± 0.0084

======================================================================
COMPARISON: L1 Logistic Regression vs Greedy Forward Selection
======================================================================

Number of features:
  Greedy Forward Selection: 10
  L1 Logistic (Lasso):      934

Cross-Validation AUC:
  Greedy Forward Selection: 0.7979
  L1 Logistic (Lasso):      0.5253 ± 0.0521

Feature Overlap:
  Common features: 6/10
    ['Colorectal Cancer Screening Status Next Due_05/05/2028', 'Depression
Screening Count Past Yr', 'Insurance Primary Payer_HealthFirst MCD', 'SDOH
```

```
Triggers_FPL<200% INSURANCE RACE ', 'Sexually Active Detail_Pap Test', 'eGFR
Date']

  Only in L1 Logistic: 928
    ['Anti-HTN Med Name_Irbesartan', 'Anti-HTN Med Name_Lisinopril-
hydroCHLOROthiazide', 'Anti-HTN Med Name_Metoprolol Succinate', 'Anti-HTN Med
Name_Metoprolol Tartrate', 'Anti-HTN Med Name_Prazosin HCl', 'Anti-HTN Med
Name_Toprol XL', 'Anti-HTN Med Name_amLODIPine Besylate-Valsartan', 'Anti-HTN
Med Name_cloNIDine HCl', 'Blood Pressure Value_100/52', 'Blood Pressure
Value_100/66']
    … and 918 more

  Only in Greedy: 4
    ['Age', 'Insurance Primary Payer_HealthFirst Essential Tier 3-4', 'Insurance
Primary Payer_HealthFirst MCR', 'SDOH Triggers_FPL<200% HISP/LAT ']
```

0.0.3 Discussion of Early Results and Moving Forward

- The greedy selection approach identified a ranked set of features contributing incremental predictive value.
- The AUC values supply an interpretable measure of classification.
- Further improvements planned include:
    1. Cross validate the model with testing data
    2. Plot the AUC for more than 10 features
    3. Check correlation of selected features to make sure we are not selecting highly correlated features
    4. Attempt to compare to Lasso if enough time

The project was a collaborative effort between Mike and Spencer with both of us working on the data cleaning step and Mike taking charge of the algorithm and Spencer doing the write up and github steps.

0.0.4 Later Results and Finalizations (final Jupyter notebook)

After the initial notebook, we accomplished most of our planned further improvements. Spencer moved forward with implementing the lasso algorithm to compare to the previous team working on this dataset per professor feedback, and Mike implemented the 5-fold cross validation and added a heatmap to check if the top 10 selected features from the greedy algorithm are correlated to each other. * Successful improvements we managed to do after the initial notebook: 1. Cross validate the model with 5-fold CV 2. Check correlation of selected features using heatmap 3. Compare to Lasso algorithm

The project can be shared with future students.

Github link: https://github.com/99x5zbrvgj-droid/CSC108HHCFinal