MIMIC-IV Mortality Prediction Tutorial

A step-by-step guide using clinical data and machine learning

Introduction

- Predictive Analytics in Healthcare
  - Leveraging large-scale EHR data to predict patient outcomes
  - MIMIC-IV: Publicly available ICU dataset with structured and unstructured data

Dataset Overview

- Key MIMIC-IV Tables:
  - ADMISSIONS
  - PATIENTS
  - ICUSTAYS
  - DIAGNOSES_ICD

Goal

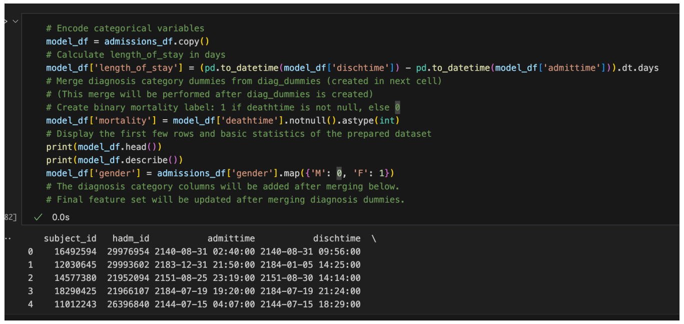
- Predict in-hospital mortality at the time of ICU admission using EHR data

Import Libraries

- pandas
- numpy
- scikit-learn
- tensorflow / PyTorch
- transformers (ClinicalBERT)

Data Loading

- Load data from MIMIC IV BigQuery into DataFrames

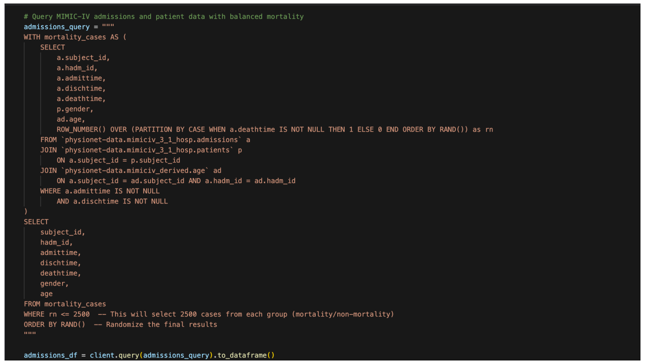


```python
# Encode categorical variables
model_df = admissions_df.copy()
# Calculate length_of_stay in days
model_df['length_of_stay'] = (pd.to_datetime(model_df['dischtime']) - pd.to_datetime(model_df['admittime'])).dt.days
# Merge diagnosis category dummies from diag_dummies (created in next cell)
# (This merge will be performed after diag_dummies is created)
# Create binary mortality label: 1 if deathtime is not null, else 0
model_df['mortality'] = model_df['deathtime'].notnull().astype(int)
# Display the first few rows and basic statistics of the prepared dataset
print(model_df.head())
print(model_df.describe())
model_df['gender'] = admissions_df['gender'].map({'M': 0, 'F': 1})
# The diagnosis category columns will be added after merging below.
# Final feature set will be updated after merging diagnosis dummies.
```

```
✓ 0.0s
```

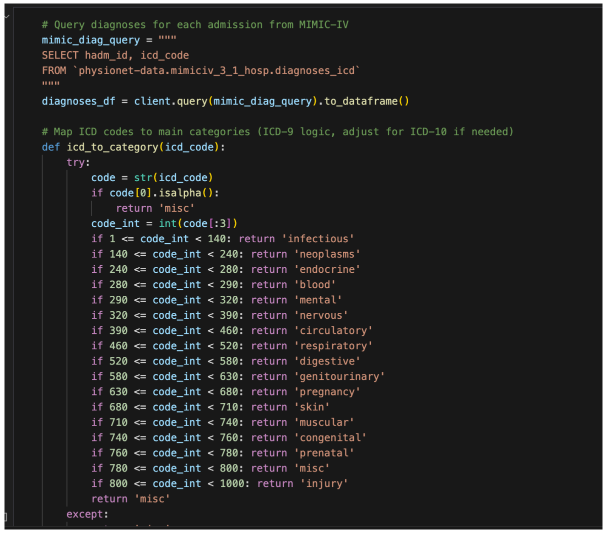
```
   subject_id   hadm_id           admittime           dischtime  \
0    16492594  29976954 2140-08-31 02:40:00 2140-08-31 09:56:00
1    12030645  29993602 2183-12-31 21:50:00 2184-01-05 14:25:00
2    14577380  21952094 2151-08-25 23:19:00 2151-08-30 14:14:00
3    18290425  21966107 2184-07-19 19:20:00 2184-07-19 21:24:00
4    11012243  26396840 2144-07-15 04:07:00 2144-07-15 18:29:00
```

Length of Stay, Mortality and Gender



```python
# Query MIMIC-IV admissions and patient data with balanced mortality
admissions_query = """
WITH mortality_cases AS (
    SELECT
        a.subject_id,
        a.hadm_id,
        a.admittime,
        a.dischtime,
        a.deathtime,
        p.gender,
        ad.age,
        ROW_NUMBER() OVER (PARTITION BY CASE WHEN a.deathtime IS NOT NULL THEN 1 ELSE 0 END ORDER BY RAND()) as rn
    FROM `physionet-data.mimiciv_3_1_hosp.admissions` a
    JOIN `physionet-data.mimiciv_3_1_hosp.patients` p
        ON a.subject_id = p.subject_id
    JOIN `physionet-data.mimiciv_derived.age` ad
        ON a.subject_id = ad.subject_id AND a.hadm_id = ad.hadm_id
    WHERE a.admittime IS NOT NULL
        AND a.dischtime IS NOT NULL
)
SELECT
    subject_id,
    hadm_id,
    admittime,
    dischtime,
    deathtime,
    gender,
    age
FROM mortality_cases
WHERE rn <= 2500  -- This will select 2500 cases from each group (mortality/non-mortality)
ORDER BY RAND()  -- Randomize the final results
"""

admissions_df = client.query(admissions_query).to_dataframe()
```

Admissions Data



```python
# Query diagnoses for each admission from MIMIC-IV
mimic_diag_query = """
SELECT hadm_id, icd_code
FROM `physionet-data.mimiciv_3_1_hosp.diagnoses_icd`
"""
diagnoses_df = client.query(mimic_diag_query).to_dataframe()

# Map ICD codes to main categories (ICD-9 logic, adjust for ICD-10 if needed)
def icd_to_category(icd_code):
    try:
        code = str(icd_code)
        if code[0].isalpha():
            return 'misc'
        code_int = int(code[:3])
        if 1 <= code_int < 140: return 'infectious'
        if 140 <= code_int < 240: return 'neoplasms'
        if 240 <= code_int < 280: return 'endocrine'
        if 280 <= code_int < 290: return 'blood'
        if 290 <= code_int < 320: return 'mental'
        if 320 <= code_int < 390: return 'nervous'
        if 390 <= code_int < 460: return 'circulatory'
        if 460 <= code_int < 520: return 'respiratory'
        if 520 <= code_int < 580: return 'digestive'
        if 580 <= code_int < 630: return 'genitourinary'
        if 630 <= code_int < 680: return 'pregnancy'
        if 680 <= code_int < 710: return 'skin'
        if 710 <= code_int < 740: return 'muscular'
        if 740 <= code_int < 760: return 'congenital'
        if 760 <= code_int < 780: return 'prenatal'
        if 780 <= code_int < 800: return 'misc'
        if 800 <= code_int < 1000: return 'injury'
        return 'misc'
    except:
```

Diagnoses code categories

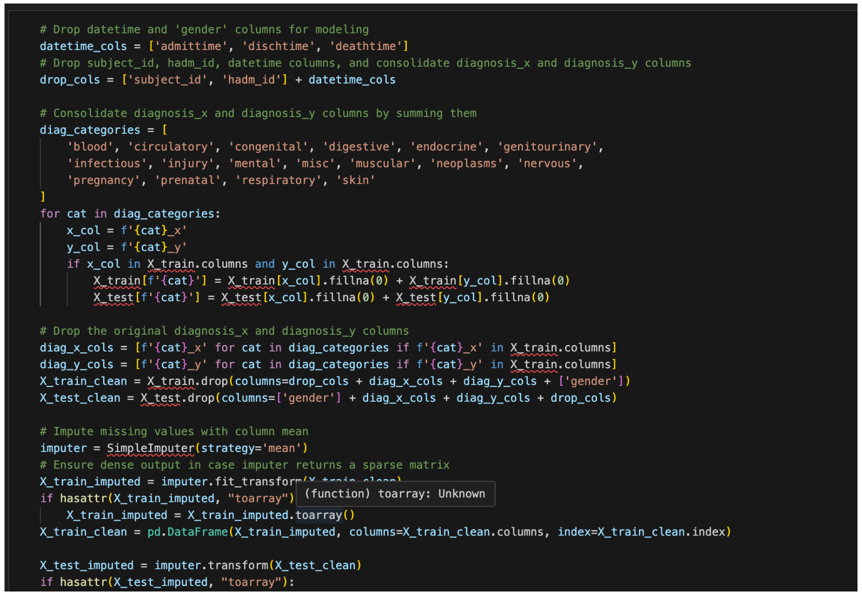
Feature Engineering

– Demographics: age, gender, ethnicity

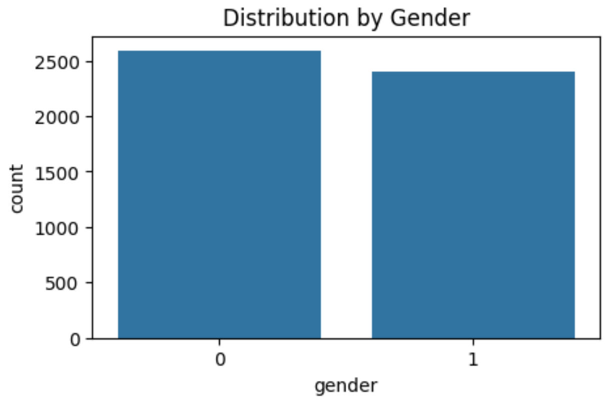
– Admission info: type, source

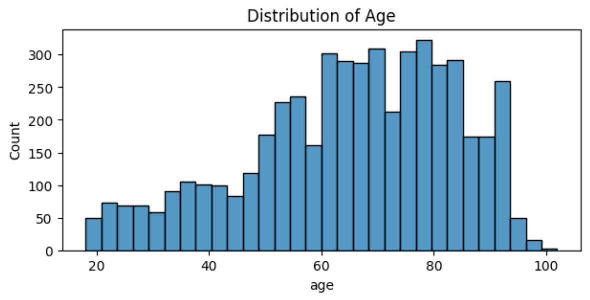
– Clinical measurements: lab results, vitals

– Diagnosis codes grouped by ICD-9 categories



```
# Drop datetime and 'gender' columns for modeling
datetime_cols = ['admittime', 'dischtime', 'deathtime']
# Drop subject_id, hadm_id, datetime columns, and consolidate diagnosis_x and diagnosis_y columns
drop_cols = ['subject_id', 'hadm_id'] + datetime_cols

# Consolidate diagnosis_x and diagnosis_y columns by summing them
diag_categories = [
    'blood', 'circulatory', 'congenital', 'digestive', 'endocrine', 'genitourinary',
    'infectious', 'injury', 'mental', 'misc', 'muscular', 'neoplasms', 'nervous',
    'pregnancy', 'prenatal', 'respiratory', 'skin'
]
for cat in diag_categories:
    x_col = f'{cat}_x'
    y_col = f'{cat}_y'
    if x_col in X_train.columns and y_col in X_train.columns:
        X_train[f'{cat}'] = X_train[x_col].fillna(0) + X_train[y_col].fillna(0)
        X_test[f'{cat}'] = X_test[x_col].fillna(0) + X_test[y_col].fillna(0)

# Drop the original diagnosis_x and diagnosis_y columns
diag_x_cols = [f'{cat}_x' for cat in diag_categories if f'{cat}_x' in X_train.columns]
diag_y_cols = [f'{cat}_y' for cat in diag_categories if f'{cat}_y' in X_train.columns]
X_train_clean = X_train.drop(columns=drop_cols + diag_x_cols + diag_y_cols + ['gender'])
X_test_clean = X_test.drop(columns=['gender'] + diag_x_cols + diag_y_cols + drop_cols)

# Impute missing values with column mean
imputer = SimpleImputer(strategy='mean')
# Ensure dense output in case imputer returns a sparse matrix
X_train_imputed = imputer.fit_transform(X_train_clean)
if hasattr(X_train_imputed, "toarray") (function) toarray: Unknown
    X_train_imputed = X_train_imputed.toarray()
X_train_clean = pd.DataFrame(X_train_imputed, columns=X_train_clean.columns, index=X_train_clean.index)

X_test_imputed = imputer.transform(X_test_clean)
if hasattr(X_test_imputed, "toarray"):
```



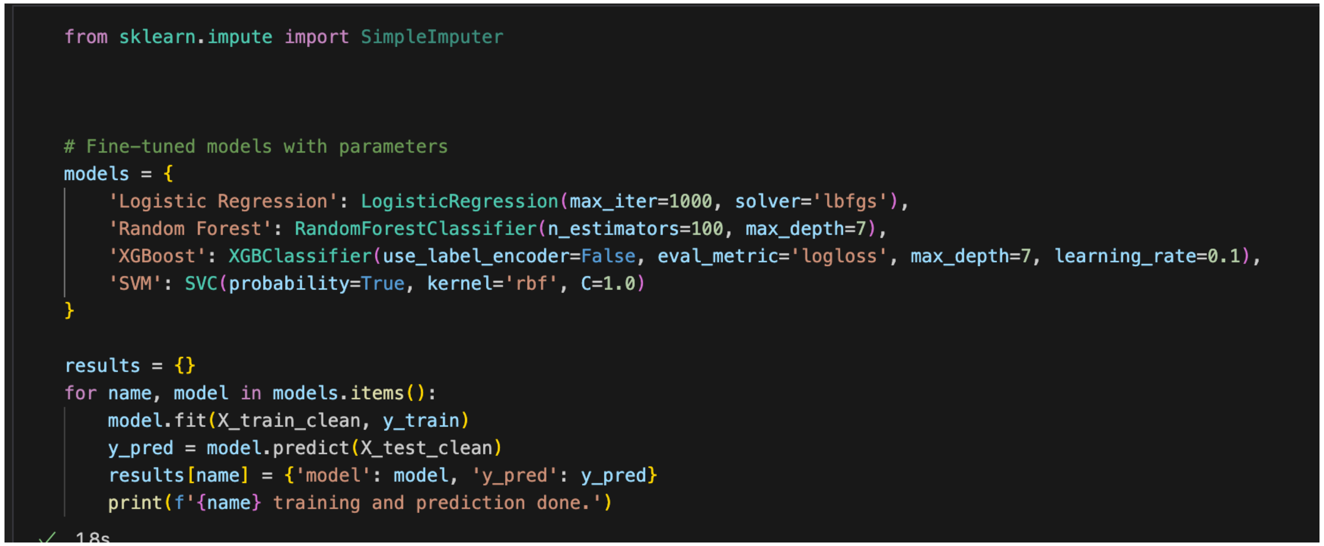


Modeling

– • Train classifiers: Logistic Regression, Random Forest, XGBoost and SVM

– • Fine-tune ClinicalBERT for text classification

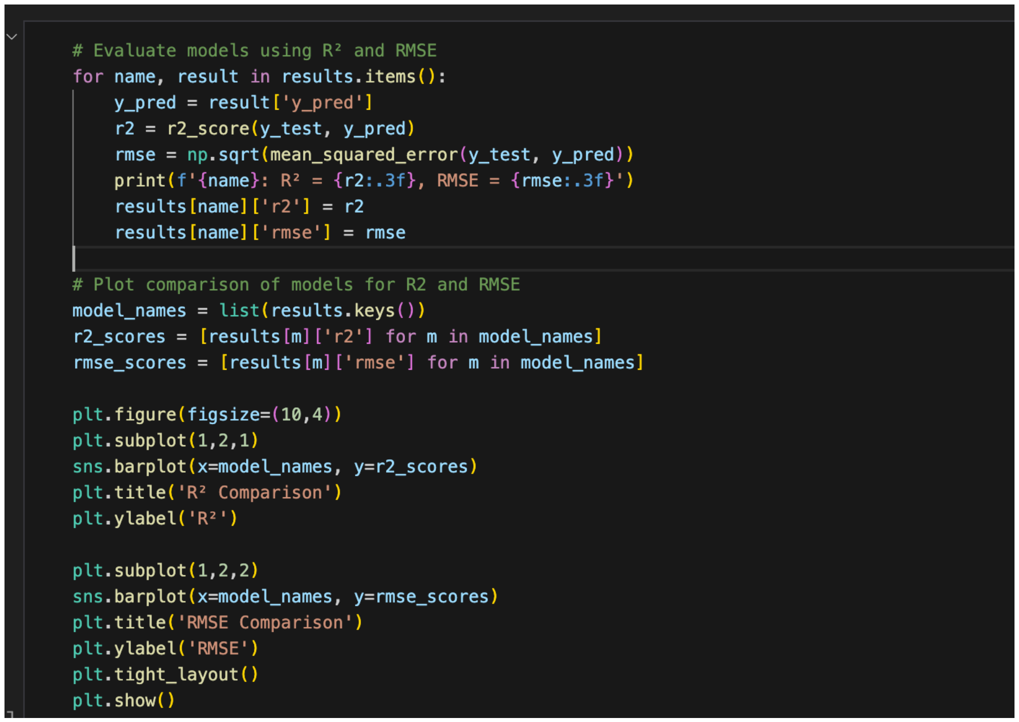
– • Split data: 80% train, 20% test



```
from sklearn.impute import SimpleImputer



# Fine-tuned models with parameters
models = {
    'Logistic Regression': LogisticRegression(max_iter=1000, solver='lbfgs'),
    'Random Forest': RandomForestClassifier(n_estimators=100, max_depth=7),
    'XGBoost': XGBClassifier(use_label_encoder=False, eval_metric='logloss', max_depth=7, learning_rate=0.1),
    'SVM': SVC(probability=True, kernel='rbf', C=1.0)
}

results = {}
for name, model in models.items():
    model.fit(X_train_clean, y_train)
    y_pred = model.predict(X_test_clean)
    results[name] = {'model': model, 'y_pred': y_pred}
    print(f'{name} training and prediction done.')
✓ 1.8s
```

Evaluation

- R^2 and RMSE Score Comparison for various models
- Confusion matrix analysis
- Feature Importance



```python
# Evaluate models using R² and RMSE
for name, result in results.items():
    y_pred = result['y_pred']
    r2 = r2_score(y_test, y_pred)
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))
    print(f'{name}: R² = {r2:.3f}, RMSE = {rmse:.3f}')
    results[name]['r2'] = r2
    results[name]['rmse'] = rmse

# Plot comparison of models for R2 and RMSE
model_names = list(results.keys())
r2_scores = [results[m]['r2'] for m in model_names]
rmse_scores = [results[m]['rmse'] for m in model_names]

plt.figure(figsize=(10,4))
plt.subplot(1,2,1)
sns.barplot(x=model_names, y=r2_scores)
plt.title('R² Comparison')
plt.ylabel('R²')

plt.subplot(1,2,2)
sns.barplot(x=model_names, y=rmse_scores)
plt.title('RMSE Comparison')
plt.ylabel('RMSE')
plt.tight_layout()
plt.show()
```

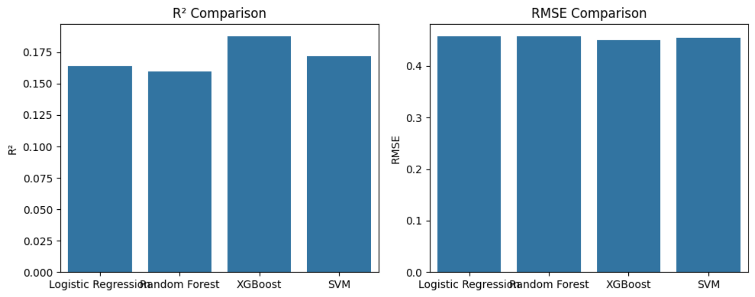
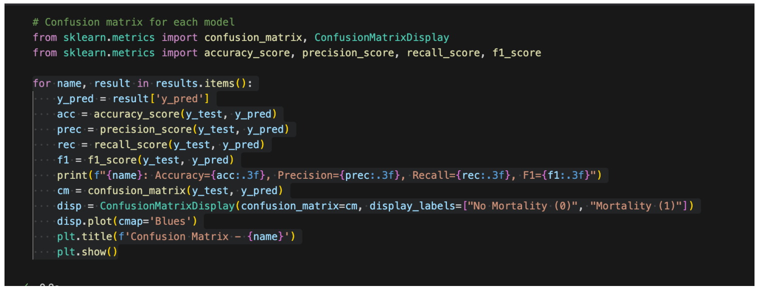


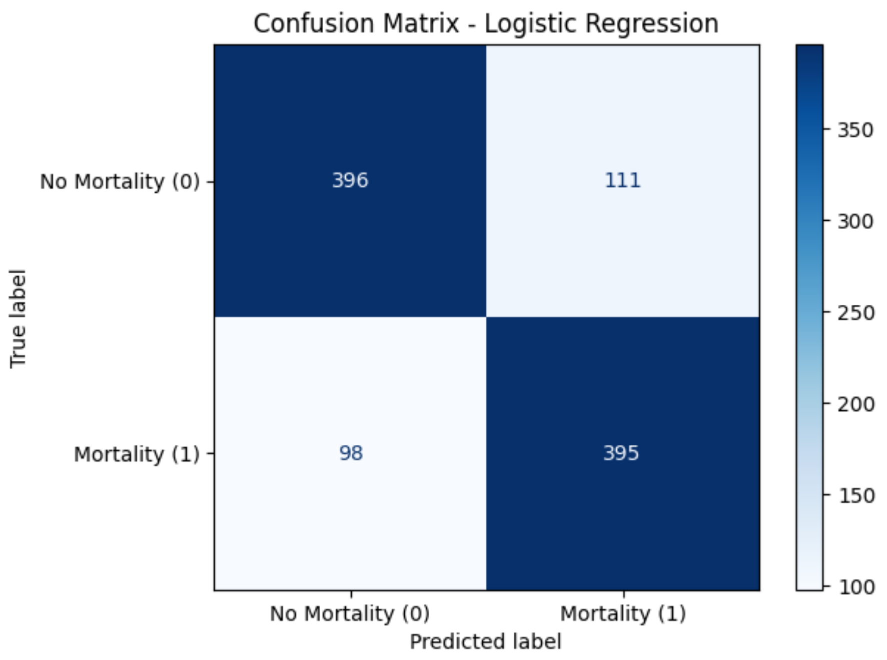
Table 1

Logistic Regression: R² = 0.164	**RMSE = 0.457**
Random Forest: R² = 0.160	RMSE = 0.458
XGBoost: R² = 0.188	RMSE = 0.451
SVM: R² = 0.172	RMSE = 0.455

Evaluation - Logistics Regression

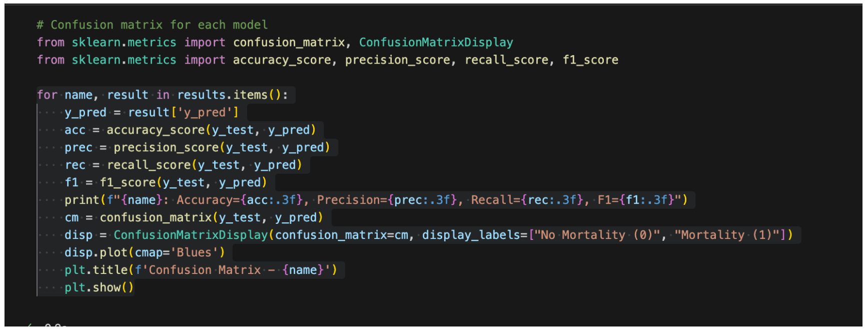


```
# Confusion matrix for each model
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

for name, result in results.items():
    y_pred = result['y_pred']
    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred)
    rec = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    print(f"{name}: Accuracy={acc:.3f}, Precision={prec:.3f}, Recall={rec:.3f}, F1={f1:.3f}")
    cm = confusion_matrix(y_test, y_pred)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["No Mortality (0)", "Mortality (1)"])
    disp.plot(cmap='Blues')
    plt.title(f'Confusion Matrix - {name}')
    plt.show()
```

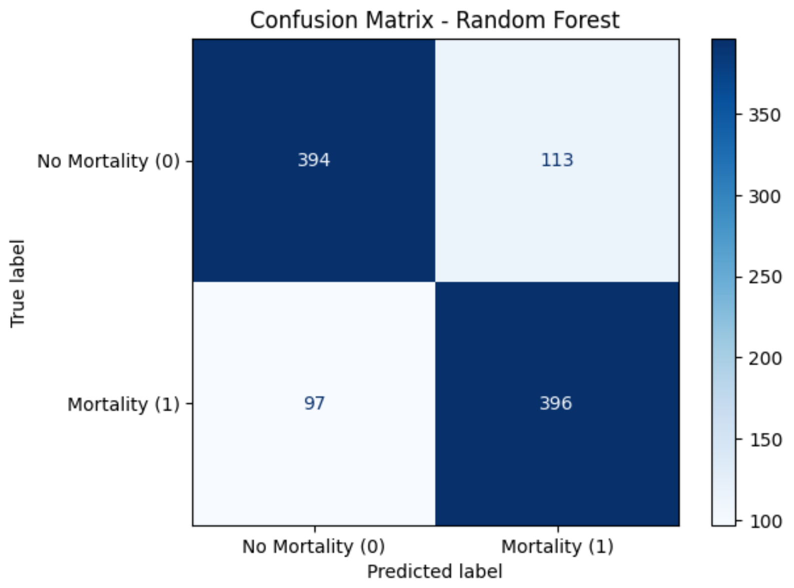


Logistic Regression: Accuracy=0.791, Precision=0.781, Recall=0.801, F1=0.791

Evaluation - Random Forest

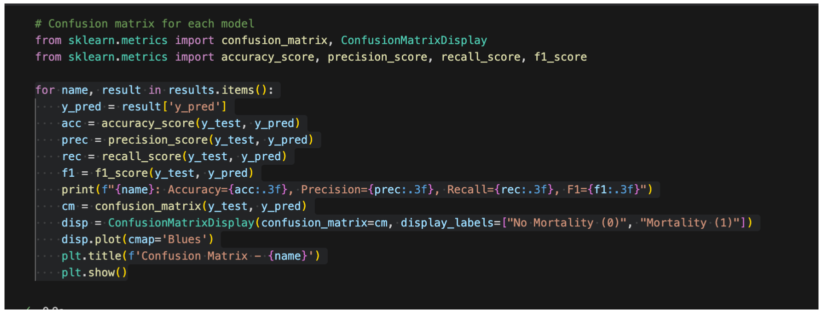


```python
# Confusion matrix for each model
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

for name, result in results.items():
    y_pred = result['y_pred']
    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred)
    rec = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    print(f"{name}: Accuracy={acc:.3f}, Precision={prec:.3f}, Recall={rec:.3f}, F1={f1:.3f}")
    cm = confusion_matrix(y_test, y_pred)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["No Mortality (0)", "Mortality (1)"])
    disp.plot(cmap='Blues')
    plt.title(f'Confusion Matrix - {name}')
    plt.show()
```

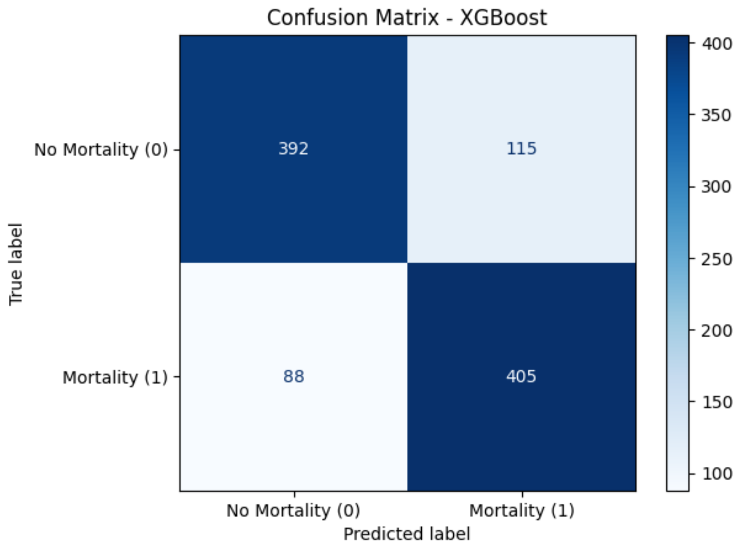


Random Forest: Accuracy=0.790, Precision=0.778, Recall=0.803, F1=0.790

Evaluation - XGBoost

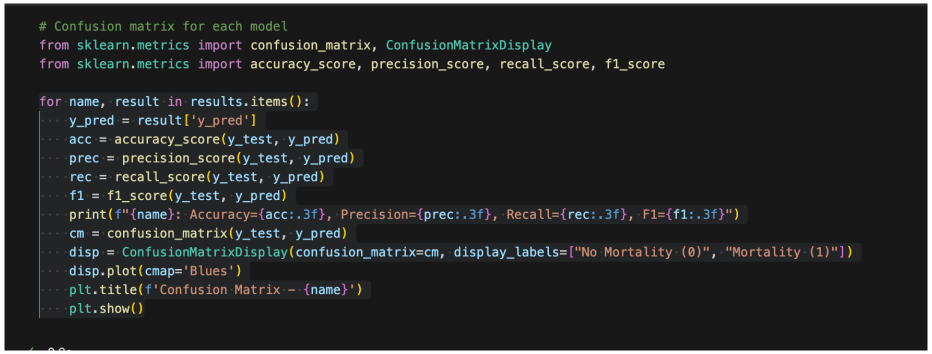


```python
# Confusion matrix for each model
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

for name, result in results.items():
    y_pred = result['y_pred']
    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred)
    rec = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    print(f"{name}: Accuracy={acc:.3f}, Precision={prec:.3f}, Recall={rec:.3f}, F1={f1:.3f}")
    cm = confusion_matrix(y_test, y_pred)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["No Mortality (0)", "Mortality (1)"])
    disp.plot(cmap='Blues')
    plt.title(f'Confusion Matrix - {name}')
    plt.show()
```

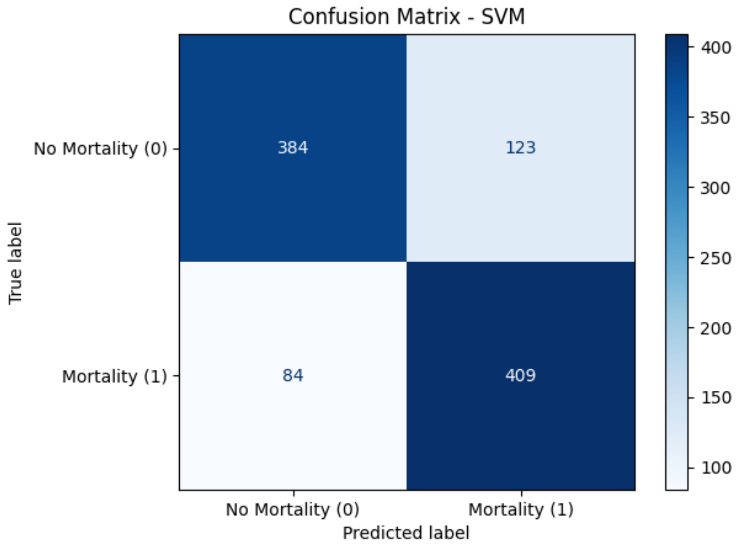


XGBoost: Accuracy=0.797, Precision=0.779, Recall=0.822, F1=0.800

Evaluation - SVM



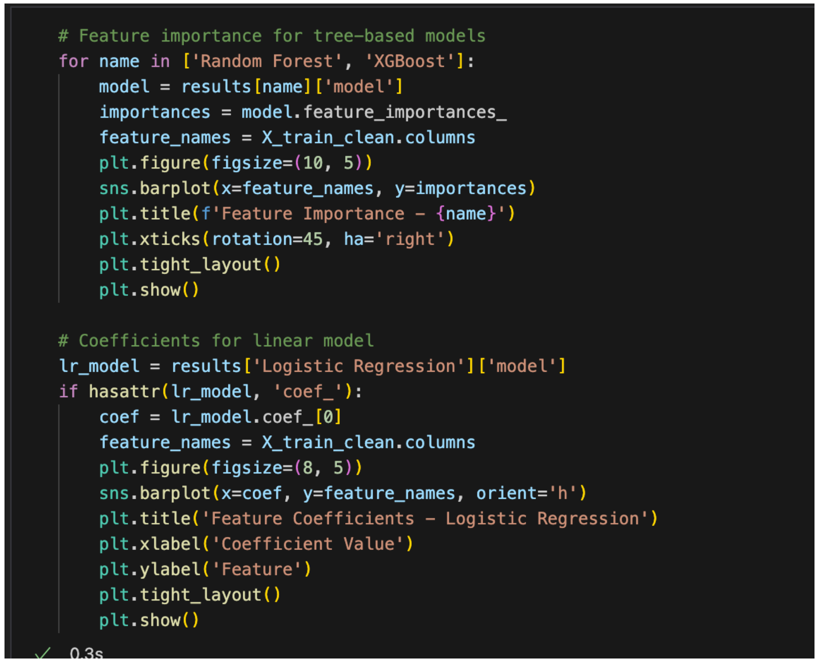
```
# Confusion matrix for each model
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

for name, result in results.items():
    y_pred = result['y_pred']
    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred)
    rec = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    print(f"{name}: Accuracy={acc:.3f}, Precision={prec:.3f}, Recall={rec:.3f}, F1={f1:.3f}")
    cm = confusion_matrix(y_test, y_pred)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["No Mortality (0)", "Mortality (1)"])
    disp.plot(cmap='Blues')
    plt.title(f'Confusion Matrix - {name}')
    plt.show()
```



SVM: Accuracy=0.793, Precision=0.769, Recall=0.830, F1=0.798

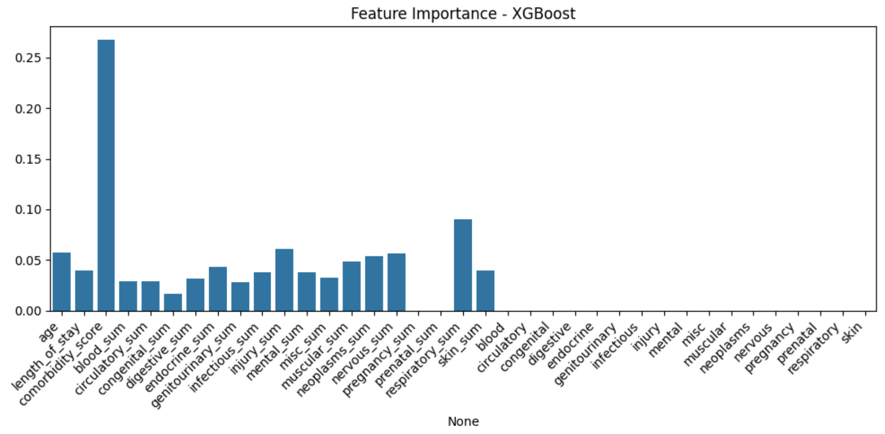
Feature Importance

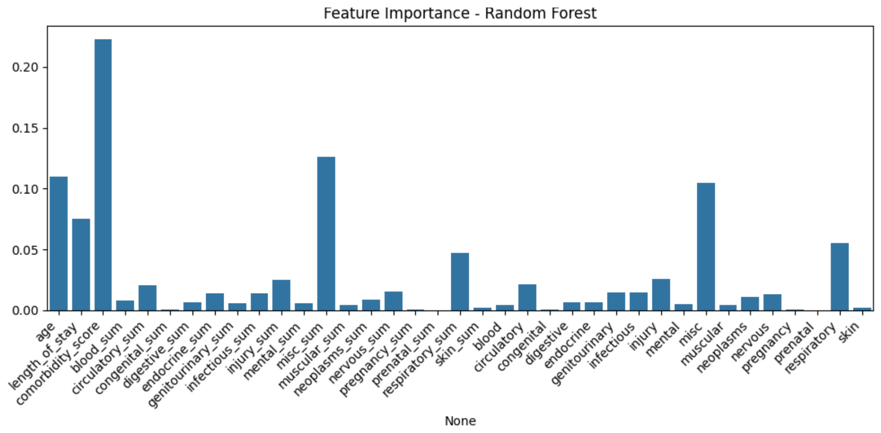
- Identify top predictive features influencing mortality risk



```
# Feature importance for tree-based models
for name in ['Random Forest', 'XGBoost']:
    model = results[name]['model']
    importances = model.feature_importances_
    feature_names = X_train_clean.columns
    plt.figure(figsize=(10, 5))
    sns.barplot(x=feature_names, y=importances)
    plt.title(f'Feature Importance - {name}')
    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()
    plt.show()

# Coefficients for linear model
lr_model = results['Logistic Regression']['model']
if hasattr(lr_model, 'coef_'):
    coef = lr_model.coef_[0]
    feature_names = X_train_clean.columns
    plt.figure(figsize=(8, 5))
    sns.barplot(x=coef, y=feature_names, orient='h')
    plt.title('Feature Coefficients - Logistic Regression')
    plt.xlabel('Coefficient Value')
    plt.ylabel('Feature')
    plt.tight_layout()
    plt.show()
```

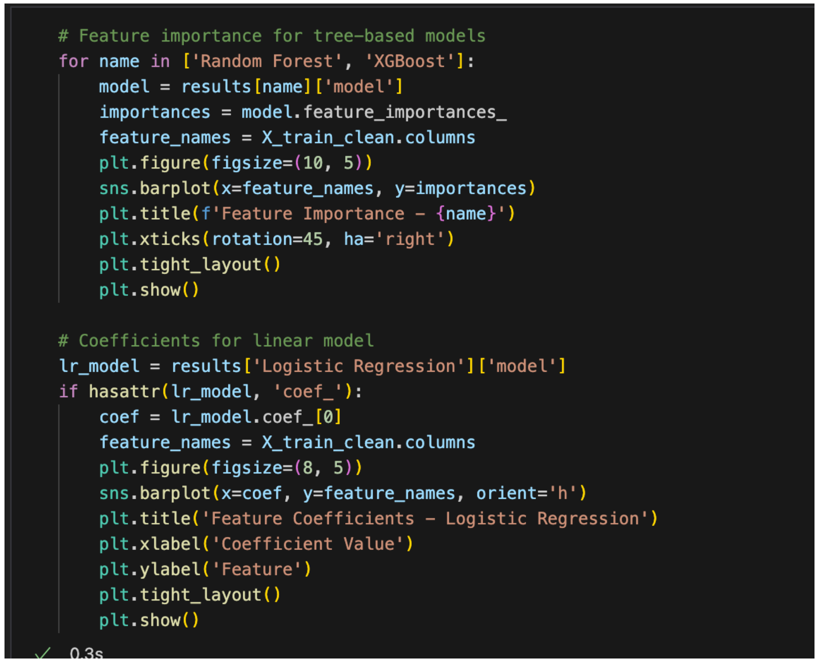
✓ 0.3s





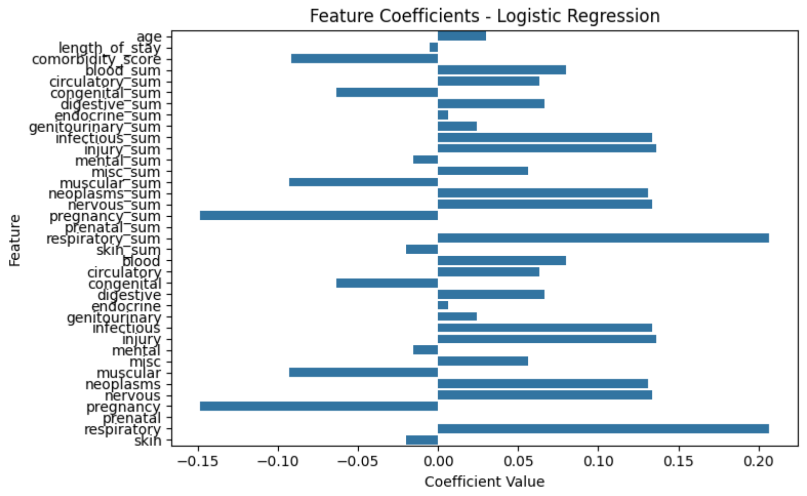
Feature Importance

- Identify top predictive features influencing mortality risk



```python
# Feature importance for tree-based models
for name in ['Random Forest', 'XGBoost']:
    model = results[name]['model']
    importances = model.feature_importances_
    feature_names = X_train_clean.columns
    plt.figure(figsize=(10, 5))
    sns.barplot(x=feature_names, y=importances)
    plt.title(f'Feature Importance - {name}')
    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()
    plt.show()

# Coefficients for linear model
lr_model = results['Logistic Regression']['model']
if hasattr(lr_model, 'coef_'):
    coef = lr_model.coef_[0]
    feature_names = X_train_clean.columns
    plt.figure(figsize=(8, 5))
    sns.barplot(x=coef, y=feature_names, orient='h')
    plt.title('Feature Coefficients - Logistic Regression')
    plt.xlabel('Coefficient Value')
    plt.ylabel('Feature')
    plt.tight_layout()
    plt.show()
```

✓ 0.3s



Conclusion & Next Steps

- <u>With R^2 score of 0.172 and RMSE of 0.455, SVM model provide the best recall (True Positive) for prediction</u>
- Integrate real-time EHR streaming data
- Deploy model in clinical decision support systems