In [1]:
```
import pandas as pd
import numpy as np
```

In [2]:
```
df = pd.read_csv(r'E:\MIMIC\MIMIC_BIG_DATA\MIMIC_BIG_DATA\case\stroke\250506重制版清洗后数据 saki.csv')
```

In [3]:
```
df.head()
```

Out[3]:

	Unnamed: 0	OASIS	SIRS	APS III	Weight	Heart_rate	SBP	DBP	MBP	Resp_rate	...	Scr_min	CKD	CRRT	Mechanical ventilation	Vasoactive_agent	LODS	DM	Liver_disease	Los_inf._AB	Los_icu_aki
0	0	41	3	52	55.3	79.0	107.0	63.0	71.0	23.0	...	53.04	NO	NO	NO	NO	5	NO	NO	12.83	13.38
1	1	35	3	51	65.0	76.0	127.0	73.0	87.0	26.0	...	44.20	NO	NO	YES	NO	7	NO	NO	15.40	38.67
2	2	26	3	52	48.0	94.0	118.0	51.0	68.0	18.0	...	185.64	YES	NO	NO	NO	6	NO	NO	13.50	17.17
3	3	24	4	41	156.1	92.0	121.0	79.0	92.0	30.0	...	53.04	NO	NO	NO	NO	2	YES	NO	5.07	31.67
4	4	32	3	54	64.1	114.0	160.0	78.0	94.0	26.0	...	97.24	NO	NO	YES	YES	4	YES	NO	1.62	20.70

5 rows × 70 columns

In [4]:
```
df = df.drop(['Unnamed: 0'], axis=1)
```

In [5]:
```
print(list(df.columns))
```

```
['OASIS', 'SIRS', 'APS III', 'Weight', 'Heart_rate', 'SBP', 'DBP', 'MBP', 'Resp_rate', 'Temperature', 'Spo2', 'Myocardial_infarct', 'Congestive_heart_failure', 'Peripheral_vascular_disease',
'Cerebrovascular_disease', 'Dementia', 'Chronic_pulmonary_disease', 'Rheumatic_disease', 'Peptic_ulcer_disease', 'Paraplegia', 'Renal_disease', 'Malignant_cancer', 'Metastatic_solid_tumor', 'A
IDS', 'Charlson_comorbidity_index', 'AKI stage', 'Gender', 'Age', 'SOFA score', 'SA-AKI', 'HTN', 'AKD', 'Po2', 'Pco2', 'PH', 'Base Excess', 'Totalco2', 'Aniongap', 'Bicarbonate', 'BUN', 'Calci
um', 'Chloride', 'Glucose', 'Sodium', 'Potassium', 'INR', 'PT', 'PTT', 'Hematocrit', 'Hemoglobin', 'MCH', 'MCHC', 'MCV', 'Platelet', 'RBC', 'RDW', 'GCS', 'Urineoutput_24hr', 'ACEI/ARB', 'Scr_m
in', 'CKD', 'CRRT', 'Mechanical ventilation', 'Vasoactive_agent', 'LODS', 'DM', 'Liver_disease', 'Los_inf._AB', 'Los_icu_aki']
```

In [6]:
```
columns_to_round = ['Los_inf._AB','Los_icu_aki','BMI','Age','PH','Calcium','Scr_min','Glucose','Pao2fio2ratio']
for col in columns_to_round:
    if col in df.columns:  # 确保列名存在于 DataFrame 中
        df[col] = df[col].round(2)
```

In [7]:
```
df.shape
```

Out[7]: (10234, 69)

In [ ]:
```
new_columns= [
    'OASIS', 'SIRS', 'APS III', 'Weight', 'Heart Rate', 'SBP', 'DBP', 'MBP', 'Resp Rate', 'Temperature', 'SpO2',
    'Myocardial Infarct', 'Congestive Heart Failure', 'Peripheral Vascular Disease', 'Cerebrovascular Disease',
    'Dementia', 'Chronic Pulmonary Disease', 'Rheumatic Disease', 'Peptic Ulcer Disease', 'Paraplegia', 'Renal Disease',
    'Malignant Cancer', 'Metastatic Solid Tumor', 'AIDS', 'Charlson Comorbidity Index', 'AKI Stage', 'Gender', 'Age',
    'SOFA Score', 'SA-AKI', 'HTN', 'AKD', 'PaO2', 'PaCO2', 'pH', 'Base Excess', 'Total CO2', 'Anion Gap', 'Bicarbonate',
    'BUN', 'Calcium', 'Chloride', 'Glucose', 'Sodium', 'Potassium', 'INR', 'PT', 'PTT', 'Hematocrit', 'Hemoglobin',
    'MCH', 'MCHC', 'MCV', 'Platelet', 'RBC', 'RDW', 'GCS', '24-hour Urine Output', 'ACEI/ARB', 'Scr Baseline', 'CKD', 'CRRT',
    'Mechanical Ventilation', 'Vasoactive Agent', 'LODS', 'DM', 'Liver Disease', 'Los_inf._AB', 'Los_icu_aki'
]

# 更改列名
df.columns = new_columns

# 打印更改后的列名
print(df.columns)
```

In [ ]:
```
df.head()
```

In [10]:
```
df.shape
```

Out[10]: (10234, 69)

In [ ]:
```
df.info()#快速检查数据集的基本属性和可能的缺失值问题。
```

In [12]:
```
df.shape
```

Out[12]: (10234, 69)

In [ ]:
```
df.select_dtypes(include='object').columns
```

In [ ]:
```
df.describe()
```

pd.set_option('display.max_rows', None) # 显示所有行 pd.set_option('display.max_columns', None) # 显示所有列 pd.set_option('display.width', None) # 不限制输出宽度
pd.set_option('display.max_colwidth', None) # 不限制列宽

现在调用describe()方法

df.describe(include='all')

In [15]:
```
print(list(df.columns))
```

```
['OASIS', 'SIRS', 'APS III', 'Weight', 'Heart Rate', 'SBP', 'DBP', 'MBP', 'Resp Rate', 'Temperature', 'SpO2', 'Myocardial Infarct', 'Congestive Heart Failure', 'Peripheral Vascular Disease',
'Cerebrovascular Disease', 'Dementia', 'Chronic Pulmonary Disease', 'Rheumatic Disease', 'Peptic Ulcer Disease', 'Paraplegia', 'Renal Disease', 'Malignant Cancer', 'Metastatic Solid Tumor', 'A
IDS', 'Charlson Comorbidity Index', 'AKI Stage', 'Gender', 'Age', 'SOFA Score', 'SA-AKI', 'HTN', 'AKD', 'PaO2', 'PaCO2', 'pH', 'Base Excess', 'Total CO2', 'Anion Gap', 'Bicarbonate', 'BUN', 'C
alcium', 'Chloride', 'Glucose', 'Sodium', 'Potassium', 'INR', 'PT', 'PTT', 'Hematocrit', 'Hemoglobin', 'MCH', 'MCHC', 'MCV', 'Platelet', 'RBC', 'RDW', 'GCS', '24-hour Urine Output', 'ACEI/AR
B', 'Scr Baseline', 'CKD', 'CRRT', 'Mechanical Ventilation', 'Vasoactive Agent', 'LODS', 'DM', 'Liver Disease', 'Los_inf._AB', 'Los_icu_aki']
```

缺失值检查

In [ ]:
```
import matplotlib.pyplot as plt
from missingno import missingno as msno
msno.bar(df)
```

In [17]:
```
missing_percentages = df.isnull().mean()

# 找出缺失数据超过50%的列
columns_to_drop = missing_percentages[missing_percentages > 0.10].index
```

In [18]:
```
columns_to_drop
```

Out[18]: Index(['PaO2', 'PaCO2', 'pH', 'Base Excess', 'Total CO2'], dtype='object')

In [19]:
```
df= df.drop(columns=columns_to_drop)
df.shape
```

Out[19]: (10234, 64)

In [20]:
```
df1=df
```

In [21]:
```
missing_ratio_per_row = df1.isna().mean(axis=1)

# 保留缺失比例≤10%的行
```

```
df1= df1[missing_ratio_per_row <= 0.10]
df1.shape
```

Out[21]: (9778, 64)

In [ ]:
```
msno.bar(df1)
```

缺失值填补

In [ ]:
```
# 使用众数填充缺失值
for column in df1.columns:
    mode_value = df1[column].mode()[0]  # 取众数的第一个值（如果有多个众数，这里只取第一个）
    df1[column] = df1[column].fillna(mode_value)

# 显示结果
print(df1)
```

In [ ]:
```
msno.bar(df1)
```

In [25]:
```
df1.shape
```

Out[25]: (9778, 64)

In [ ]:
```
df1.select_dtypes(include='object').columns
```

In [ ]:
```
df1.select_dtypes(include='float64').columns
```

In [28]:
```
df1.select_dtypes(include='int64').columns
```

Out[28]:
```
Index(['OASIS', 'SIRS', 'APS III', 'Charlson Comorbidity Index', 'AKI Stage',
       'SOFA Score', 'LODS'],
      dtype='object')
```

In [29]:
```
df1.select_dtypes(include='int32').columns
```

Out[29]: Index([], dtype='object')

In [ ]:
```
from missingno import missingno as msno
msno.bar(df1)
```

In [31]:
```
df1.isnull().sum()
```

Out[31]:
```
OASIS            0
SIRS             0
APS III          0
Weight           0
Heart Rate       0
                ..
LODS             0
DM               0
Liver Disease    0
Los_inf._AB      0
Los_icu_aki      0
Length: 64, dtype: int64
```

one-hot编码：

In [ ]:
```
df1.select_dtypes(include='object')#字符类型列
```

In [33]:
```
cols = df1.select_dtypes(include='object').columns
```

In [ ]:
```
df1.describe(include='all')
```

In [ ]:
```
cols
```

In [36]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
```

In [ ]:
```
for col in cols:#是一个循环，它遍历 cols 列表中的每个元素。
   df1[col] = le.fit_transform(df1[col])
```

In [ ]:
```
df1[cols]
```

In [ ]:
```
df1.head()
```

In [40]:
```
df1.CRRT.value_counts()
```

Out[40]:
```
CRRT
0    9262
1     516
Name: count, dtype: int64
```

In [41]:
```
df1['Vasoactive Agent'].value_counts()
```

Out[41]:
```
Vasoactive Agent
0    5632
1    4146
Name: count, dtype: int64
```

In [42]:
```
count = df1[(df1['Vasoactive Agent'] == 1) & (df1['CRRT'] == 1)].shape[0]
count
```

Out[42]: 393

In [ ]:
```
df1.head()
```

df1.to_csv('MIMIC_saki.csv')

分离目标变量 y（‘outcome’ 列）和特征变量 x（除了 ‘outcome’ 列之外的所有其他列）

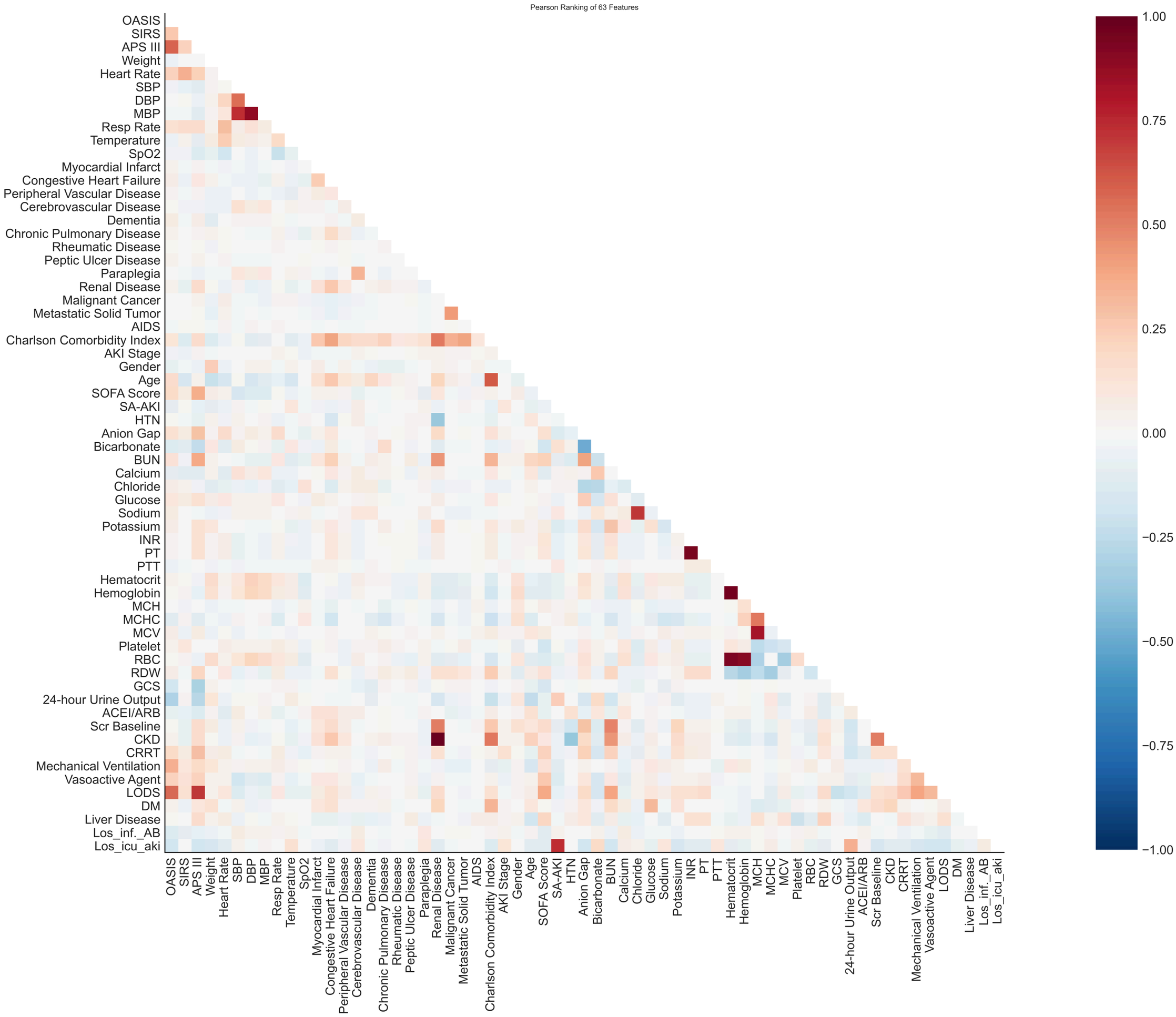
In [44]:
```
y = df1['AKD']
x = df1.drop('AKD',axis=1)
```

皮尔逊相关性

特征之间相关性筛选特征，去除特征（自变量）之间高相关性的特征

In [45]:
```
import matplotlib.pyplot as plt
from yellowbrick.features import Rank2D
import numpy as np

# 确保x只包含数值型数据
plt.rcParams.update({'font.size': 100})
plt.figure(figsize=(30, 30), dpi=300)
x = x.select_dtypes(include=[np.number])
```

```
# 实例化可视化器并禁用标题
visualizer = Rank2D(
    features=x.columns,
    algorithm='pearson',
    title=None  # 禁用标题
)

# 拟合和转换数据
visualizer.fit(x, y)
visualizer.transform(x)

# 获取轴对象
ax = visualizer.ax

ax.spines['left'].set_color('black')
ax.spines['left'].set_linewidth(1.5)
ax.spines['bottom'].set_color('black')
ax.spines['bottom'].set_linewidth(1.5)
ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)


# 移除网格线和刻度线
ax.grid(False)
ax.tick_params(axis='both', which='both', length=0)

# 设置主坐标轴标签字体
ax.tick_params(axis='both', labelsize=20)

# 关键修改：在finalize前准备所有设置
visualizer.finalize()  # 准备图形元素（包括颜色条）

# 获取颜色条并调整尺寸
cax = visualizer.ax.figure.axes[-1]  # 获取颜色条

# 获取当前颜色条的位置 [left, bottom, width, height]
pos = cax.get_position()

# 缩小颜色条尺寸：宽度缩小50%，高度缩小30%
new_width = pos.width * 0.5  # 宽度变为原来的一半
new_height = pos.height * 0.8  # 高度变为原来的70%

# 计算新位置（保持居中）
new_left = pos.x0 + (pos.width - new_width) / 2
new_bottom = pos.y0 + (pos.height - new_height) / 2

# 设置新的颜色条位置和大小
cax.set_position([new_left, new_bottom, new_width, new_height])

# 设置颜色条字体
cax.tick_params(labelsize=20)  # 设置颜色条字体


# 使用show()显示结果
visualizer.show()
```



Out[45]: <Axes: title={'center': 'Pearson Ranking of 63 Features'}>

import matplotlib.pyplot as plt from yellowbrick.features import Rank2D import numpy as np

确保x只包含数值型数据

x = x.select_dtypes(include=[np.number])

创建图形对象 - 直接在这里设置尺寸和DPI

fig, ax = plt.subplots(figsize=(30, 30), dpi=300)

实例化可视化器 - 禁用所有可能的标题选项

visualizer = Rank2D(features=x.columns, algorithm='pearson', ax=ax, # 使用我们创建的轴对象 title=None, # 禁用主标题 show_title=False # 额外确保不显示标题)

拟合和转换数据

visualizer.fit(x, y) visualizer.transform(x)

移除所有可能的标题元素

visualizer.finalize() # 准备图形元素 ax.set_title('') # 清除轴标题 fig.suptitle('') # 清除图形标题

设置边框

for spine in ['top', 'right']: ax.spines[spine].set_visible(False) for spine in ['left', 'bottom']: ax.spines[spine].set_visible(True) ax.spines[spine].set_linewidth(2) # 加粗边框

移除网格线和刻度线

ax.grid(False) ax.tick_params(axis='both', which='both', length=0)

设置主坐标轴标签字体

ax.tick_params(axis='both', labelsize=20)

设置颜色条字体

if hasattr(visualizer, 'ax') and hasattr(visualizer.ax.figure, 'axes'): for cax in visualizer.ax.figure.axes: if cax != ax: # 跳过主轴（只处理颜色条） cax.tick_params(labelsize=20)

plt.show() # 关闭图形，避免显示

plt.savefig('correlation_heatmap.png', bbox_inches='tight', pad_inches=0.1) plt.close() # 关闭图形，避免显示

In [46]:
```
def filter_highly_correlated_features(df1, correlation_threshold=0.9):
    """
    筛选出高度相关的特征。

    参数:
    df: pandas DataFrame，包含要筛选的特征。
    correlation_threshold: 相关性阈值，默认为0.9。

    返回:
    highly_correlated_features: 包含高度相关特征的列表。
    """
    # 计算特征之间的相关性矩阵
    correlation_matrix = df1.corr()

    # 初始化一个空列表来保存高度相关的特征对
    highly_correlated_features = []

    # 遍历相关性矩阵的上三角部分
    for i in range(len(correlation_matrix.columns)):
        for j in range(i):
            # 检查相关性是否超过阈值
            if abs(correlation_matrix.iloc[i, j]) > correlation_threshold:
                # 添加高度相关的特征对
                highly_correlated_features.append((correlation_matrix.columns[i], correlation_matrix.columns[j]))

    return highly_correlated_features
```

In [47]:
```
df1.corr()
```

Out[47]:

	OASIS	SIRS	APS III	Weight	Heart Rate	SBP	DBP	MBP	Resp Rate	Temperature	...	Scr Baseline	CKD	CRRT	Mechanical Ventilation	Vasoactive Agent	LODS	
OASIS	1.000000	0.264100	0.578275	-0.052295	0.219366	-0.032894	-0.003988	-0.025368	0.154013	-0.042473	...	0.013193	0.037182	0.186794	0.358611	0.219081	0.569052	0
SIRS	0.264100	1.000000	0.219039	0.006709	0.349755	-0.068094	0.009616	-0.020857	0.158177	0.042421	...	-0.077100	-0.074779	0.072357	0.112460	0.127954	0.144003	-0
APS III	0.578275	0.219039	1.000000	-0.009916	0.230983	-0.095446	-0.053854	-0.097426	0.178137	-0.087270	...	0.169296	0.167589	0.294814	0.180794	0.221376	0.703741	0
Weight	-0.052295	0.006709	-0.009916	1.000000	0.030737	0.016074	0.044396	0.025850	0.039843	0.070872	...	0.083728	0.012564	0.059683	0.120658	0.040151	0.035594	0
Heart Rate	0.219366	0.349755	0.230983	0.030737	1.000000	0.014388	0.200095	0.108736	0.303868	0.247936	...	-0.041599	-0.051049	0.075401	0.039975	0.032537	0.107069	-0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
LODS	0.569052	0.144003	0.703741	0.035594	0.107069	-0.131234	-0.047227	-0.077802	0.088859	-0.081208	...	0.141159	0.158962	0.254566	0.383996	0.315148	1.000000	0
DM	0.033569	-0.039457	0.098401	0.156248	-0.016485	0.052217	-0.059419	-0.031396	0.007742	0.006134	...	0.168884	0.211133	0.048907	0.003213	0.012505	0.061024	1
Liver Disease	0.013812	-0.006296	0.163851	0.052818	0.085686	-0.052602	0.012514	-0.022203	0.050293	0.008458	...	0.006502	-0.023431	0.130094	0.055737	0.035506	0.133441	-0
Los_inf._AB	-0.123887	-0.088343	-0.065911	-0.042560	-0.024800	0.077491	0.041935	0.053711	0.017553	0.032012	...	-0.024424	0.003270	-0.041297	-0.021981	-0.071063	-0.097656	-0
Los_icu_aki	-0.181802	-0.012281	-0.173234	-0.149960	-0.095444	0.049965	-0.019283	0.025126	-0.009950	0.139817	...	-0.148222	-0.109489	-0.110615	-0.168381	-0.193764	-0.175314	-0

64 rows × 64 columns

In [48]:
```
result = filter_highly_correlated_features(x)
```

In [49]:
```
result
```

Out[49]:
```
[('PT', 'INR'),
 ('Hemoglobin', 'Hematocrit'),
 ('RBC', 'Hematocrit'),
 ('RBC', 'Hemoglobin'),
 ('CKD', 'Renal Disease')]
```

根据以上结果，手动删除掉一些相关性较高的变量！

In [50]:
```
x = x.drop(['INR','Hematocrit','RBC','Renal Disease'],axis=1)
```

In [51]:
```
x.shape
```

Out[51]:
```
(9778, 59)
```

1.基于特征重要性方法：

In [ ]:
```
from sklearn.ensemble import GradientBoostingClassifier#导入梯度提升分类库
from yellowbrick.model_selection import FeatureImportances
visualizer = FeatureImportances(GradientBoostingClassifier(random_state=0)) # 选择模型
ax = plt.subplots(figsize=(20,20), dpi=300)
visualizer.fit(x,y)#训练模型
visualizer.poof(ax=ax)#绘制特征重要性图
```

In [ ]:
```
features = visualizer.features_[visualizer.feature_importances_>0]# 选择重要性大于0的特征
```

In [ ]:
```
features
```

In [ ]:
```
X_fm = x[features]# 选择重要性大于0的特征
```

In [ ]:
```
X_fm.shape# 查看数据的形状
```

Out[ ]:
```
(9778, 46)
```

In [ ]:
```
X_fm.head()# 查看数据的前5行
```

2.Brotua降维（非常准）：

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
```

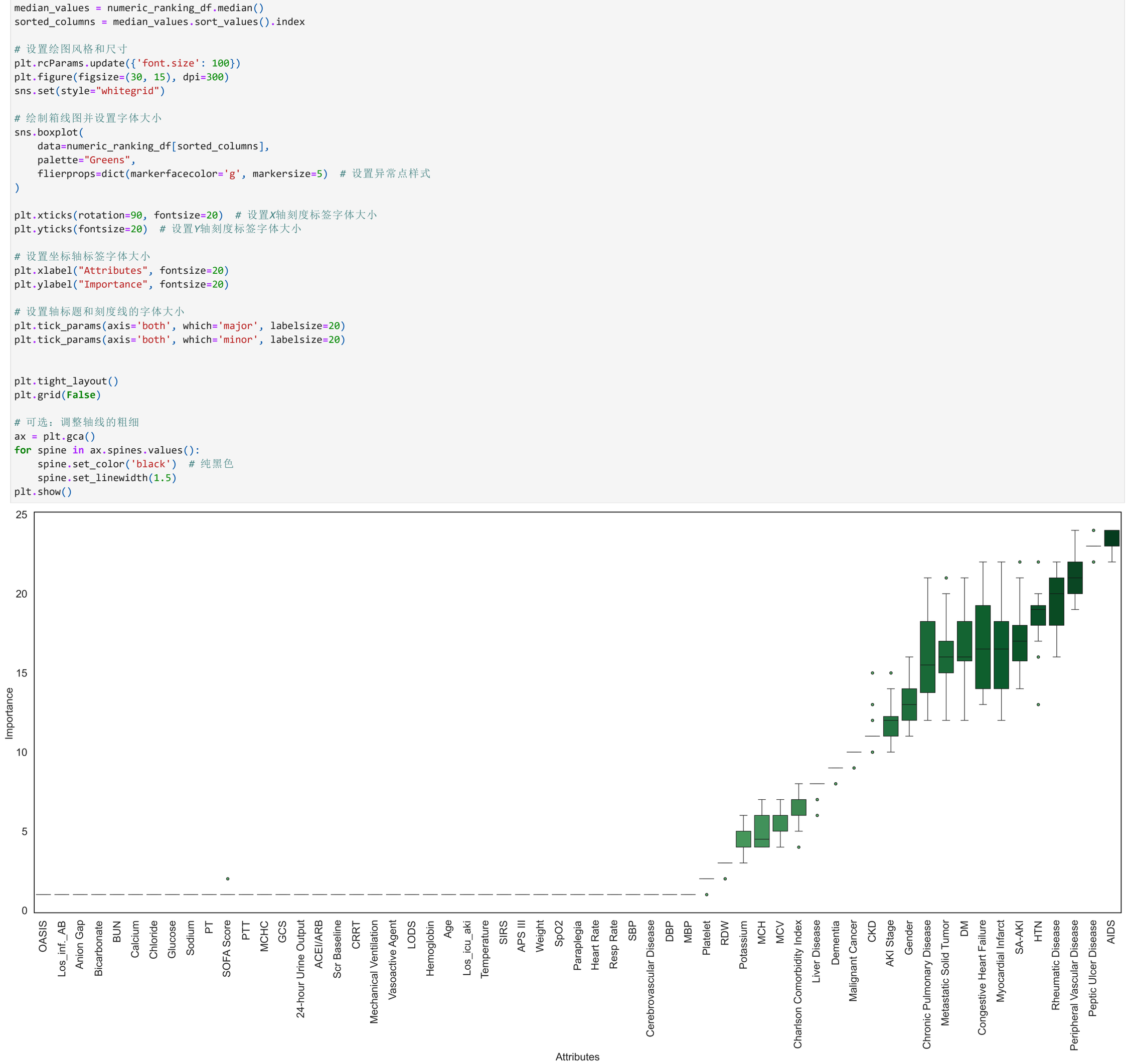
In [ ]:
```
from sklearn.ensemble import RandomForestClassifier
from boruta import BorutaPy
# 初始化随机森林模型
rf = RandomForestClassifier(n_jobs=-1, class_weight='balanced', max_depth=5)
# 初始化Boruta特征选择器
```

```
boruta_selector = BorutaPy(rf, n_estimators='auto', verbose=2, random_state=42)
# 对训练数据进行特征选择
boruta_selector.fit(x.values, y.values)
# 检查选中的特征
selected_features = x.columns[boruta_selector.support_].to_list()
# 打印被选择的特征
print("Selected Features: ", selected_features)
# 打印被剔除的特征
rejected_features = x.columns[~boruta_selector.support_].to_list()
print("Rejected Features: ", rejected_features)
# 打印有待定性的特征
tentative_features = x.columns[boruta_selector.support_weak_].to_list()
print("Tentative Features: ", tentative_features)
```

In [ ]:
```
feature_ranks = boruta_selector.ranking_
# 将特征名称和排名结合成一个DataFrame
feature_importance_df = pd.DataFrame({
    'Feature': x.columns,
    'Rank': feature_ranks
})
feature_importance_df
```

In [ ]:
```
rf = RandomForestClassifier(n_jobs=-1, class_weight='balanced', max_depth=5)
# 初始化存储特征排名的 DataFrame
ranking_df = pd.DataFrame(index=range(1, 21), columns=x.columns)
# 运行 Boruta 20 次
for i in range(20):
    print(f"Iteration {i+1}")
     # 初始化Boruta特征选择器
    boruta_selector = BorutaPy(rf, n_estimators='auto', verbose=2, random_state=i, max_iter=50)
    # 对训练数据进行特征选择
    boruta_selector.fit(x.values, y.values)
    # 获取特征排名
    feature_ranks = boruta_selector.ranking_
    # 将特征排名保存到 DataFrame 中
    ranking_df.loc[i+1] = feature_ranks
ranking_df
```

In [ ]:
```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

# 确保数据是数值型
numeric_ranking_df = ranking_df.apply(pd.to_numeric, errors='coerce')

# 计算每个特征的中位数并排序
median_values = numeric_ranking_df.median()
sorted_columns = median_values.sort_values().index

# 设置绘图风格和尺寸
plt.rcParams.update({'font.size': 100})
plt.figure(figsize=(30, 15), dpi=300)
sns.set(style="whitegrid")

# 绘制箱线图并设置字体大小
sns.boxplot(
    data=numeric_ranking_df[sorted_columns],
    palette="Greens",
    flierprops=dict(markerfacecolor='g', markersize=5)  # 设置异常点样式
)

plt.xticks(rotation=90, fontsize=20)  # 设置X轴刻度标签字体大小
plt.yticks(fontsize=20)  # 设置Y轴刻度标签字体大小

# 设置坐标轴标签字体大小
plt.xlabel("Attributes", fontsize=20)
plt.ylabel("Importance", fontsize=20)

# 设置轴标题和刻度线的字体大小
plt.tick_params(axis='both', which='major', labelsize=20)
plt.tick_params(axis='both', which='minor', labelsize=20)


plt.tight_layout()
plt.grid(False)

# 可选：调整轴线的粗细
ax = plt.gca()
for spine in ax.spines.values():
    spine.set_color('black')  # 纯黑色
    spine.set_linewidth(1.5)
plt.show()
```



In [ ]:
```
sorted_columns
```

Out[ ]:
```
Index(['OASIS', 'Los_inf._AB', 'Anion Gap', 'Bicarbonate', 'BUN', 'Calcium',
       'Chloride', 'Glucose', 'Sodium', 'PT', 'SOFA Score', 'PTT', 'MCHC',
       'GCS', '24-hour Urine Output', 'ACEI/ARB', 'Scr Baseline', 'CRRT',
       'Mechanical Ventilation', 'Vasoactive Agent', 'LODS', 'Hemoglobin',
       'Age', 'Los_icu_aki', 'Temperature', 'SIRS', 'APS III', 'Weight',
       'SpO2', 'Paraplegia', 'Heart Rate', 'Resp Rate', 'SBP',
       'Cerebrovascular Disease', 'DBP', 'MBP', 'Platelet', 'RDW', 'Potassium',
       'MCH', 'MCV', 'Charlson Comorbidity Index', 'Liver Disease', 'Dementia',
       'Malignant Cancer', 'CKD', 'AKI Stage', 'Gender',
       'Chronic Pulmonary Disease', 'Metastatic Solid Tumor', 'DM',
       'Congestive Heart Failure', 'Myocardial Infarct', 'SA-AKI', 'HTN',
       'Rheumatic Disease', 'Peripheral Vascular Disease',
       'Peptic Ulcer Disease', 'AIDS'],
      dtype='object')
```

In [59]:
```
X_boruta=x[['OASIS', 'Los_inf._AB', 'Anion Gap', 'Bicarbonate', 'BUN', 'Calcium',
       'Chloride', 'Glucose', 'Sodium', 'PT', 'SOFA Score', 'PTT', 'MCHC',
       'GCS', '24-hour Urine Output', 'ACEI/ARB', 'Scr Baseline', 'CRRT',
       'Mechanical Ventilation', 'Vasoactive Agent', 'LODS', 'Hemoglobin',
       'Age', 'Los_icu_aki', 'Temperature', 'SIRS', 'APS III', 'Weight',
       'SpO2', 'Paraplegia', 'Heart Rate', 'Resp Rate', 'SBP',
       'Cerebrovascular Disease', 'DBP', 'MBP']]
```

In [60]:
```
X_boruta.shape
```

Out[60]:
```
(9778, 36)
```

3.递归特征消除方法（RFE)

4.LassoCV可视化

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

# 划分训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=42, stratify=df1['AKD'])
```

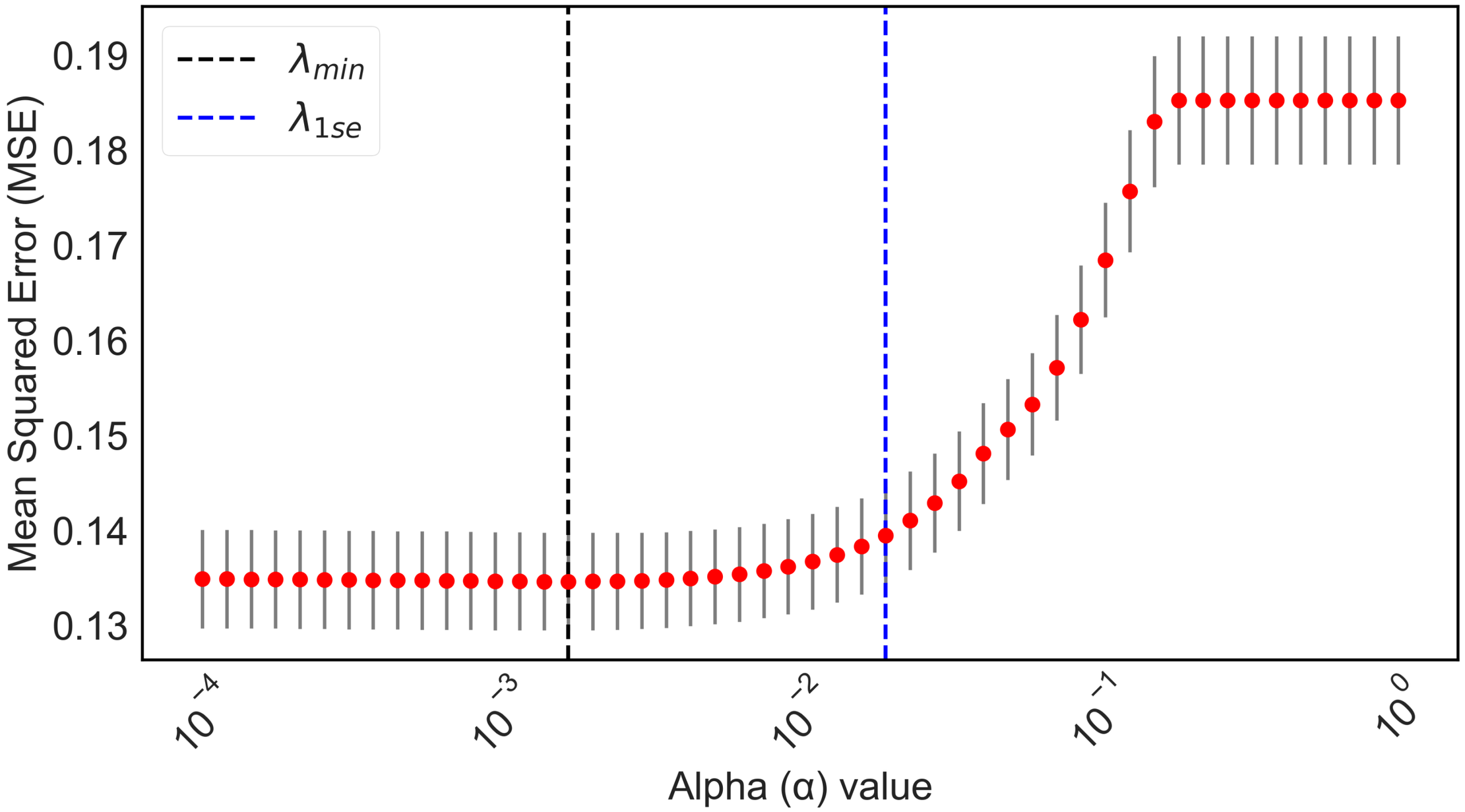
In [ ]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_error



# 假设 X_train, X_test, y_train, y_test 已经被正确地划分和准备

# 标准化数据
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 训练Lasso模型
lasso = Lasso(alpha=0.1)  # alpha是正则化强度的参数
lasso.fit(X_train_scaled, y_train)

# 打印Lasso模型的系数
coefficients = pd.Series(lasso.coef_, index=x.columns)
selected_features = coefficients[coefficients != 0].index
print("Selected features:")
print(selected_features)

# 计算测试集的均方误差
y_pred = lasso.predict(X_test_scaled)
mse = mean_squared_error(y_test, y_pred)

print(f"\nMean Squared Error on Test Set: {mse}")
```

```
Selected features:
Index(['CRRT', 'Mechanical Ventilation'], dtype='object')

Mean Squared Error on Test Set: 0.16622124130597035
```

In [ ]:
```
from sklearn.linear_model import LassoCV
from sklearn.model_selection import RepeatedKFold
import numpy as np
import matplotlib.pyplot as plt

# Assuming feature names are stored in feature_names list
feature_names = x.columns

# Define alpha range
alphas = np.logspace(-4, 0, 50)

# LassoCV with cross-validation
lasso_cv = LassoCV(alphas=alphas, cv=RepeatedKFold(n_splits=10, n_repeats=3, random_state=42), random_state=42)
lasso_cv.fit(X_train_scaled, y_train)

# Calculate MSE path and std
mse_path = lasso_cv.mse_path_.mean(axis=1)
mse_std = lasso_cv.mse_path_.std(axis=1)

# Find best alpha and 1-SE alpha
best_alpha_index = np.argmin(mse_path)
best_alpha = lasso_cv.alphas_[best_alpha_index]
one_se_index = np.where(mse_path <= mse_path[best_alpha_index] + mse_std[best_alpha_index])[0][0]
one_se_alpha = lasso_cv.alphas_[one_se_index]

print(f"Best alpha (λ_min): {best_alpha}")
print(f"1-SE rule alpha (λ_1se): {one_se_alpha}")

# Feature selection for both alphas
lasso_best_alpha = LassoCV(alphas=[best_alpha], cv=RepeatedKFold(n_splits=10, n_repeats=3, random_state=42), random_state=42)
lasso_best_alpha.fit(X_train_scaled, y_train)
selected_features_best = [feature_names[i] for i in np.where(lasso_best_alpha.coef_ != 0)[0]]
print(f"Selected features with λ_min: {selected_features_best}")

lasso_one_se_alpha = LassoCV(alphas=[one_se_alpha], cv=RepeatedKFold(n_splits=10, n_repeats=3, random_state=42), random_state=42)
lasso_one_se_alpha.fit(X_train_scaled, y_train)
selected_features_one_se = [feature_names[i] for i in np.where(lasso_one_se_alpha.coef_ != 0)[0]]
print(f"Selected features with λ_1se: {selected_features_one_se}")

# Plot with improved styling
plt.figure(figsize=(12, 6), dpi=300)

# Main plot elements
plt.errorbar(lasso_cv.alphas_, mse_path, yerr=mse_std, fmt='o', color='red',
             ecolor='gray', capsize=3, markersize=8)
plt.axvline(lasso_cv.alphas_[best_alpha_index], linestyle='--',
            color='black', linewidth=2, label=r'$\lambda_{min}$')
```

```
plt.axvline(lasso_cv.alphas_[one_se_index], linestyle='--',
            color='blue', linewidth=2, label=r'$\lambda_{1se}$')

# Axis settings
plt.xscale('log')
plt.xlabel('Alpha (α) value', fontsize=20)  # Larger and bolder
plt.ylabel('Mean Squared Error (MSE)', fontsize=20)

# Tick and legend settings
plt.xticks(rotation=45, fontsize=20)
plt.yticks(fontsize=20)
plt.legend(fontsize=20, frameon=True)

# Border and grid styling
ax = plt.gca()
for spine in ax.spines.values():
    spine.set_color('black')
    spine.set_linewidth(1.5)  # Thicker borders

plt.grid(False)
plt.show()
```

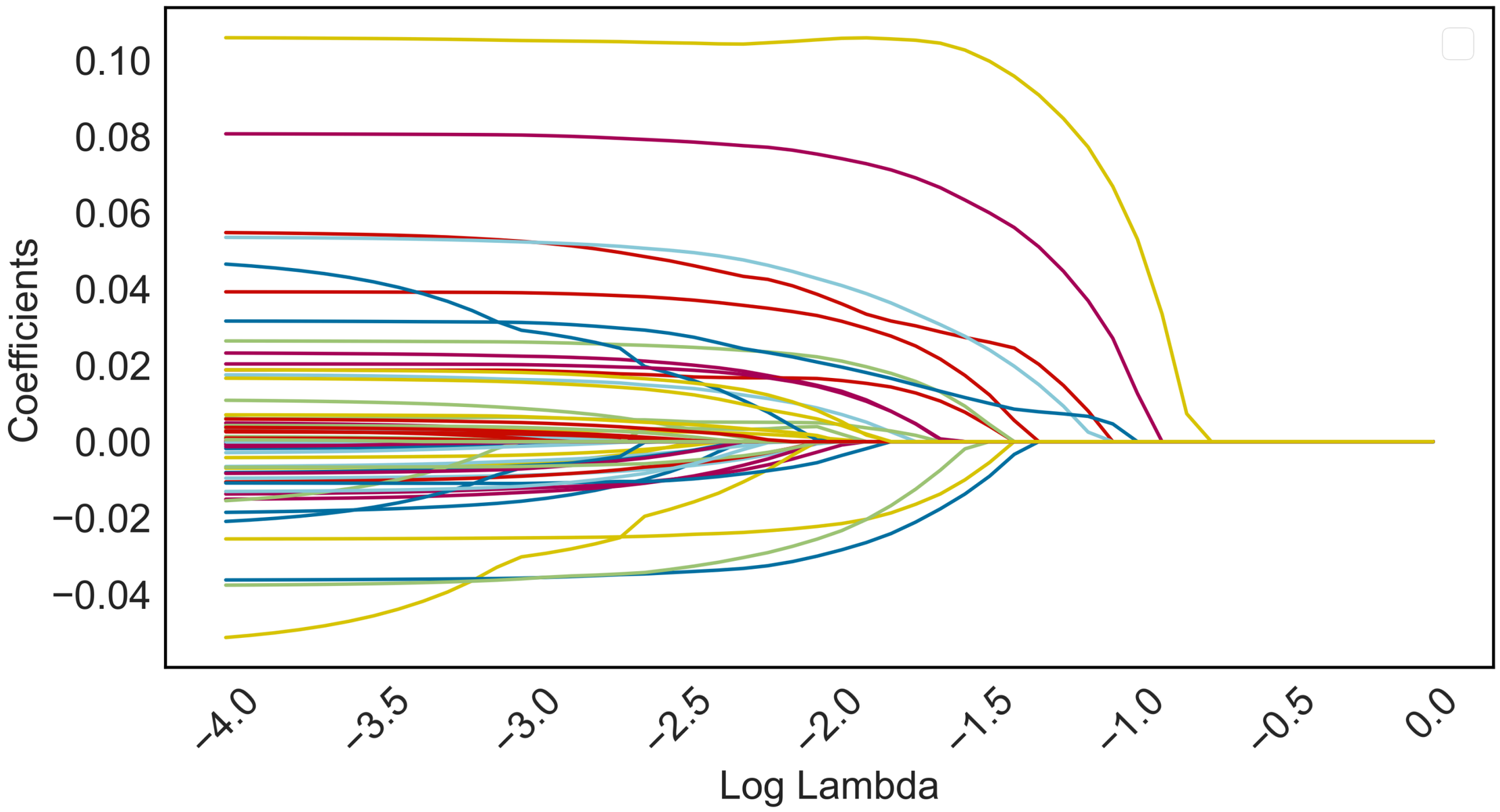
```
Best alpha (λ_min): 0.0016768329368110067
1-SE rule alpha (λ_1se): 0.019306977288832496
Selected features with λ_min: ['OASIS', 'SIRS', 'APS III', 'Weight', 'Heart Rate', 'SBP', 'MBP', 'Resp Rate', 'SpO2', 'Myocardial Infarct', 'Congestive Heart Failure', 'Peripheral Vascular Dis
ease', 'Cerebrovascular Disease', 'Dementia', 'Rheumatic Disease', 'Paraplegia', 'Malignant Cancer', 'Metastatic Solid Tumor', 'AIDS', 'AKI Stage', 'SOFA Score', 'SA-AKI', 'HTN', 'Anion Gap',
'Calcium', 'Chloride', 'Glucose', 'Sodium', 'Potassium', 'PT', 'Hemoglobin', 'MCHC', 'MCV', 'Platelet', 'GCS', '24-hour Urine Output', 'ACEI/ARB', 'Scr Baseline', 'CRRT', 'Mechanical Ventilati
on', 'Vasoactive Agent', 'LODS', 'Liver Disease', 'Los_inf._AB', 'Los_icu_aki']
Selected features with λ_1se: ['APS III', 'Weight', 'MBP', 'Resp Rate', 'SpO2', 'Cerebrovascular Disease', 'Paraplegia', 'ACEI/ARB', 'Scr Baseline', 'CRRT', 'Mechanical Ventilation', 'Vasoacti
ve Agent', 'LODS', 'Los_inf._AB']
```



In [ ]:

```
coefs = []

for a in alphas:
    lasso = Lasso(alpha=a, max_iter=10000)
    lasso.fit(X_train_scaled, y_train)
    coefs.append(lasso.coef_)

# 可视化系数路径
plt.figure(figsize=(12, 6), dpi=300)
ax = plt.gca()

# 使用对数坐标显示正则化参数
ax.plot(np.log10(alphas), coefs)
plt.xlabel('Log Lambda', fontsize=20)
plt.ylabel('Coefficients', fontsize=20)
plt.xticks(rotation=45, fontsize=20)
plt.yticks(fontsize=20)
plt.legend(fontsize=20, frameon=True)
ax = plt.gca()
for spine in ax.spines.values():
    spine.set_color('black')
    spine.set_linewidth(1.5)
plt.grid(False)

plt.show()
```

```
No artists with labels found to put in legend.  Note that artists whose label start with an underscore are ignored when legend() is called with no argument.
```



In [ ]:
```
print(list(X_boruta.columns))
```

```
['OASIS', 'Los_inf._AB', 'Anion Gap', 'Bicarbonate', 'BUN', 'Calcium', 'Chloride', 'Glucose', 'Sodium', 'PT', 'SOFA Score', 'PTT', 'MCHC', 'GCS', '24-hour Urine Output', 'ACEI/ARB', 'Scr Baseline', 'CRRT', 'Mechanical Ventilation', 'Vasoactive Agent', 'LODS', 'Hemoglobin', 'Age', 'Los_icu_aki', 'Temperature', 'SIRS', 'APS III', 'Weight', 'SpO2', 'Paraplegia', 'Heart Rate', 'Resp Rate', 'SBP', 'Cerebrovascular Disease', 'DBP', 'MBP']
```

In [ ]:
```
selected_features_one_se
```

Out[ ]:
```
['APS III',
 'Weight',
 'MBP',
 'Resp Rate',
 'SpO2',
 'Cerebrovascular Disease',
 'Paraplegia',
 'ACEI/ARB',
 'Scr Baseline',
 'CRRT',
 'Mechanical Ventilation',
 'Vasoactive Agent',
 'LODS',
 'Los_inf._AB']
```

In [ ]:
```
import networkx as nx
import matplotlib.pyplot as plt

# 定义Boruta和Lasso选择的特征
boruta_features =['OASIS', 'Los_inf._AB', 'Anion Gap', 'Bicarbonate', 'BUN', 'Calcium', 'Chloride','Glucose', 'Sodium', 'PT',
                  'SOFA Score', 'PTT', 'MCHC', 'GCS', '24-hour Urine Output', 'ACEI/ARB', 'Scr Baseline', 'CRRT', 'Mechanical Ventilation',
                    'Vasoactive Agent', 'LODS', 'Hemoglobin', 'Age', 'Los_icu_aki', 'Temperature', 'SIRS', 'APS III', 'Weight', 'SpO2',
                   'Paraplegia', 'Heart Rate', 'Resp Rate', 'SBP', 'Cerebrovascular Disease', 'DBP', 'MBP']

lasso_features = selected_features_one_se

# 创建集合用于求交集
boruta_set = set(boruta_features)  # Boruta特征集合
lasso_set = set(lasso_features)    # Lasso特征集合
intersection = boruta_set.intersection(lasso_set)  # 两个集合的交集
boruta_only = boruta_set - intersection  # 仅Boruta选择的特征
lasso_only = lasso_set - intersection    # 仅Lasso选择的特征

# 创建图
G = nx.Graph()

# 添加节点和边
for feature in boruta_only:
    G.add_edge('Boruta', feature, color='lightcoral')  # 淡红色表示仅被Boruta选择的特征

for feature in lasso_only:
    G.add_edge('Lasso', feature, color='lightblue')  # 淡蓝色表示仅被Lasso选择的特征

for feature in intersection:
    G.add_edge('Boruta', feature, color='lightcoral')  # 淡红色边连接交集特征到Boruta
    G.add_edge('Lasso', feature, color='lightblue')  # 淡蓝色边连接交集特征到Lasso

# 获取边的颜色
edge_colors = [data['color'] for _, _, data in G.edges(data=True)]

# 设置节点的颜色
node_colors = []
for node in G.nodes():
    if node == 'Boruta':
        node_colors.append('lightcoral')  # Boruta节点淡红色
    elif node == 'Lasso':
        node_colors.append('lightblue')   # Lasso节点淡蓝色
    elif node in boruta_only:
        node_colors.append('lightcoral')  # 仅被Boruta选择的特征淡红色
    elif node in lasso_only:
        node_colors.append('lightblue')   # 仅被Lasso选择的特征淡蓝色
    elif node in intersection:
        node_colors.append('plum')        # 交集特征节点用淡紫色表示

# 绘制图形
plt.figure(figsize=(13, 13), dpi=300)
pos = nx.spring_layout(G, seed=42)  # 使用spring布局
nx.draw_networkx(
    G,
    pos,
    edge_color=edge_colors,
    node_color=node_colors,
    with_labels=True,
```

```
    node_size=1000,
    font_size=10,
    edgecolors='none'  # 移除节点边框
)
plt.title('Feature Selection by Boruta and Lasso')
plt.grid(False)
plt.show()
```

In [ ]:
```
from matplotlib_venn import venn2
import matplotlib.pyplot as plt

# 创建图形并设置大小
plt.figure(figsize=(10, 10), dpi=300)

# 定义颜色
color1 = "#EEA3FF"  # 浅蓝色
color2 = "#F99D9D"  # 浅红色

# 绘制韦恩图
v = venn2([set(lasso_features), set(boruta_features)],
          set_labels=('Lasso Features', 'Boruta Features'),
          set_colors=(color1, color2),
          alpha=0.5)

# 设置字体大小
font_size = 20  # 设置统一的字体大小
for text in v.set_labels:  # 集合标签
    if text is not None:
        text.set_fontsize(font_size)
for text in v.subset_labels:  # 子集数字标签
    if text is not None:
        text.set_fontsize(font_size)

# 可选：添加自定义标题（如果需要）
# plt.title("Feature Selection Comparison", fontsize=18)

# 显示图形
plt.show()
```

In [ ]:
```
print(np.intersect1d(boruta_features,lasso_features))
```

```
['ACEI/ARB' 'APS III' 'CRRT' 'Cerebrovascular Disease' 'LODS'
 'Los_inf._AB' 'MBP' 'Mechanical Ventilation' 'Paraplegia' 'Resp Rate'
 'Scr Baseline' 'SpO2' 'Vasoactive Agent' 'Weight']
```

In [61]:
```
X_1 = X_boruta[['ACEI/ARB', 'APS III' ,'CRRT', 'Cerebrovascular Disease', 'LODS',
                'Los_inf._AB', 'MBP', 'Mechanical Ventilation', 'Paraplegia', 'Resp Rate',
                'Scr Baseline' ,'SpO2', 'Vasoactive Agent', 'Weight']]
```

In [62]:
```
X_1
```

Out[62]:

	ACEI/ARB	APS III	CRRT	Cerebrovascular Disease	LODS	Los_inf._AB	MBP	Mechanical Ventilation	Paraplegia	Resp Rate	Scr Baseline	SpO2	Vasoactive Agent	Weight
0	1	52	0	0	5	12.83	71.0	0	0	23.0	53.04	100.0	0	55.3
1	0	51	0	0	7	15.40	87.0	1	0	26.0	44.20	99.0	0	65.0
2	0	52	0	0	6	13.50	68.0	0	0	18.0	185.64	94.0	0	48.0
3	0	41	0	0	2	5.07	92.0	0	0	30.0	53.04	93.0	0	156.1
4	1	54	0	0	4	1.62	94.0	1	0	26.0	97.24	95.0	1	64.1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
10229	0	50	0	0	8	2.80	122.0	1	1	20.0	70.72	97.0	1	134.5
10230	0	32	0	0	8	14.78	86.0	1	0	16.0	79.56	100.0	1	59.0
10231	0	39	0	0	5	17.00	91.0	0	0	15.0	44.20	100.0	1	85.0
10232	1	110	0	1	12	8.83	133.0	1	0	20.0	61.88	100.0	0	110.0
10233	1	64	0	0	8	26.83	87.0	1	0	20.0	167.96	98.0	0	77.6

9778 rows × 14 columns

In [ ]:
```
X_1.info()
```

In [ ]:
```

```

5.卡方检验

常用的特征方法是：几种特征筛选出的结果，取交集。

划分数据集：

先数据标准化，再用smote处理不平衡问题

In [ ]:
```
print(X_1.dtypes)
```

In [ ]:
```
y.value_counts()
```

Out[ ]:
```
AKD
0    7378
1    2400
Name: count, dtype: int64
```

In [ ]:
```
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from collections import Counter

# 1. 初始数据划分（分层抽样）
X_train_raw, X_test, y_train, y_test = train_test_split(
    X_1,
    y,
    test_size=0.3,
    stratify=y,  # 保持原始分布
    random_state=42
)
# 2. 标准化处理（仅在训练集上拟合）
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_raw)  # 仅用原始训练集计算参数
X_test = scaler.transform(X_test)                   # 使用相同参数转换测试集
```

In [ ]:
```
import joblib
```

joblib.dump(scaler, '0511重置版saki_scaler.pkl')

In [ ]:
```
# 3. 在标准化后的训练集上应用SMOTE
smote = SMOTE(random_state=42)
X_train, y_train = smote.fit_resample(X_train_scaled, y_train)  # 重命名为标准名称

# 4. 验证数据状态
print("\n=== 数据处理验证 ===")
print(f"[原始训练集] 样本数: {len(X_train_raw)}, 特征数: {X_train_raw.shape[1]}")
print(f"[平衡后训练集] 样本数: {len(X_train)}, 类别分布: {dict(Counter(y_train))}")
print(f"[测试集] 样本数: {len(X_test)}, 类别分布: {dict(Counter(y_test))}")

# 5. 最终可用数据集:
# X_train, y_train: 已标准化且平衡的训练数据
# X_test, y_test: 已标准化且保持原始分布的测试数据
```

```
=== 数据处理验证 ===
[原始训练集] 样本数: 6844, 特征数: 14
[平衡后训练集] 样本数: 10328, 类别分布: {1: 5164, 0: 5164}
[测试集] 样本数: 2934, 类别分布: {0: 2214, 1: 720}
```

构建预测模型:

In [ ]:
```
import shap #模型可解释性分析
import lightgbm as lgb # 集成学习算法
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier, AdaBoostClassifier # 集成学习算法
from sklearn.tree import DecisionTreeClassifier # 决策树

from sklearn.model_selection import train_test_split, RandomizedSearchCV, cross_val_score, StratifiedKFold #划分数据集、交叉验证
from sklearn.metrics import f1_score, precision_score, recall_score, roc_auc_score #评价指标

from sklearn.metrics import confusion_matrix, roc_curve, classification_report #RandomizedSearchCV是Python中的一个类，位于sklearn.model_selection模块中，用于随机搜索最佳超参数组合。该类会接收一
# 相比GridSearchCV网格搜索方法，随机搜索方法可以在更短的时间内找到一个相对较好的超参数组合，适
```

In [ ]:
```
from sklearn.linear_model import LogisticRegression # 逻辑回归
from sklearn.svm import SVC # 支持向量机
from sklearn.neighbors import KNeighborsClassifier # K近邻
from xgboost import XGBClassifier # XGBoost
from sklearn.ensemble import GradientBoostingClassifier # 梯度提升树
from sklearn.ensemble import RandomForestClassifier # 随机森林
from sklearn.ensemble import ExtraTreesClassifier # 极端随机树
from sklearn.ensemble import AdaBoostClassifier # AdaBoost
from lightgbm import LGBMClassifier # LightGBM
```

In [ ]:
```
# 初始化各个模型
models = {
    "Logistic Regression": {'model':LogisticRegression(random_state=0),
                            'param_grid':{
                                'penalty': ['l1', 'l2', 'elasticnet', 'none'],
                                'C': np.linspace(0.01, 10, 10)
                            }},

    "SVM": {'model':SVC(probability=True,random_state=0),
            'param_grid':{
                'C': np.linspace(0.01, 10, 10),
                'kernel': ['linear', 'rbf', 'poly'],
                'gamma': ['scale', 'auto']
            }},

    "KNN": {'model':KNeighborsClassifier(),
            'param_grid':{
                'n_neighbors': range(5, 100, 5),
                'weights': ['uniform', 'distance'],
                'p': [1, 2]  # 1表示曼哈顿距离，2表示欧几里得距离
           }},



"Decision Tree": {'model':DecisionTreeClassifier(random_state=0),
                  'param_grid':{
                    'criterion': ['gini', 'entropy'],
                    'max_depth': range(3, 9),
                    'min_samples_split': range(2, 6),
                    'min_samples_leaf': range(2, 6)
                   }},

"Random Forest": {'model':RandomForestClassifier(random_state=0),
                  'param_grid':{
                     'n_estimators': range(10, 50, 10),
                     'max_depth': range(3, 9),
                     'min_samples_split': range(2, 6),
                     'min_samples_leaf': range(2, 6)
                  }},

"Adaboost": {'model':AdaBoostClassifier(random_state=0),
             'param_grid':{
                'n_estimators': range(10, 50, 10),
                'learning_rate': np.linspace(0.01, 1, 10),
             }},

"Gradient Boosting": {'model':GradientBoostingClassifier(random_state=0),
                      'param_grid':{
                         'n_estimators': range(10, 50, 10),
                         'learning_rate': np.linspace(0.01, 1, 10),
                         'max_depth': range(3, 9),
                         'subsample': np.linspace(0.1, 1, 5)
                      }},

    "XGBoost": {'model':XGBClassifier(random_state=0),
                'param_grid':{
                    'n_estimators': range(10, 50, 10),
                    'learning_rate': np.linspace(0.01, 1, 10),
                    'max_depth': range(3, 8),
                    'subsample': np.linspace(0.1, 1, 5)
                }},

    "LightGBM": {'model':LGBMClassifier(force_col_wise=True, verbosity=-1,random_state=0),
                 'param_grid':{
                     'n_estimators': range(10, 50, 10),
                     'learning_rate': np.linspace(0.01, 1, 10),
                     'max_depth': range(3, 8),
                     'subsample': np.linspace(0.1, 1, 5)
                 }},

}

# 训练并评估每个模型
from sklearn.model_selection import RandomizedSearchCV
best_estimators = []
names = []
for model_name, model_info in models.items():
    print(f"\n============================ {model_name} ============================")

    # 使用GridSearchCV进行自动调参
    grid_search = RandomizedSearchCV(model_info['model'], model_info['param_grid'], cv=5, scoring='roc_auc',n_jobs=-1) # accuracy roc_auc
```

```
    grid_search.fit(X_train, y_train)

    # 获取最优的超参数配置和模型
    best_params = grid_search.best_params_
    best_model = grid_search.best_estimator_
    best_estimators.append(best_model)
    names.append(model_name)
    print(best_params)
    print(best_model)

from sklearn.ensemble import BaggingClassifier

bag = BaggingClassifier( ## If None, then the base estimator is a decision tree.
                                bootstrap_features=False,random_state=0)
param_grid = {'n_estimators':[10,30,50,70,80,150,160, 170,175,180,185]}
random_search  = RandomizedSearchCV(estimator=bag,param_distributions=param_grid,cv=5,n_jobs=-1,random_state=48)
random_search.fit(X_train,y_train)
names.append('Bagging')
best_estimators.append(random_search.best_estimator_)
from sklearn.naive_bayes import  GaussianNB
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
clf1 = LogisticRegression(random_state=0)
clf2 = RandomForestClassifier(random_state=0)
clf3 = GaussianNB()
clf = VotingClassifier(estimators=[('lr', clf1), ('rf', clf2), ('gnb', clf3)],
voting='soft')

params = {'lr__C': [1.0, 100.0], 'rf__n_estimators': [20, 200]}
grid = RandomizedSearchCV(estimator=clf, param_distributions=params, cv=5,n_jobs=-1,random_state=0)
grid.fit(X_train,y_train)
names.append('Voting')

best_estimators.append(grid.best_estimator_)
```

In [ ]:

```
def score_summary_roc(names, classifiers,X_train_,y_train_,x_test_,y_test_):
    from sklearn.metrics import accuracy_score,roc_curve,confusion_matrix
    from sklearn.metrics import RocCurveDisplay,auc
    plt.figure(figsize=(10, 10), dpi=300)
    ax = plt.gca()
    cols=["Classifier", "Accuracy", "ROC_AUC", "Recall", "Precision", "F1"]

    df = []
    for name, clf in zip(names, classifiers):
        clf.fit(X_train_, y_train_)

        pred = clf.predict(x_test_)
        accuracy = accuracy_score(y_test_, pred)

        pred_proba = clf.predict_proba(x_test_)[:, 1]

        fpr, tpr, thresholds = roc_curve(y_test_, pred_proba)
        roc_auc = auc(fpr, tpr)
        display = RocCurveDisplay(fpr=fpr, tpr=tpr, roc_auc=roc_auc,estimator_name=name)
        display.plot(ax=ax)
        plt.grid(visible=False)

        cm = confusion_matrix(y_test_, pred)

        # recall: TP/(TP+FN)
        recall = cm[1,1]/(cm[1,1] +cm[1,0])

        # precision: TP/(TP+FP)
        precision = cm[1,1]/(cm[1,1] +cm[0,1])

        # F1 score: TP/(TP+FP)
        f1 = 2*recall*precision/(recall + precision)
        df.append([name, accuracy*100, roc_auc, recall, precision, f1])


    data_table = pd.DataFrame(data=df,columns=cols)


    return(np.round(data_table.reset_index(drop=True), 2))
```

In [ ]:

```
result = score_summary_roc(names,best_estimators,X_train,y_train,X_test,y_test)
```

In [ ]:

```
def score_summary_roc(names, classifiers, X_train_, y_train_, x_test_, y_test_):
    from sklearn.metrics import accuracy_score, roc_curve, confusion_matrix
    from sklearn.metrics import RocCurveDisplay, auc
    import matplotlib.pyplot as plt
    import pandas as pd
    import numpy as np

    # 创建图形并设置大小
    plt.figure(figsize=(10, 10), dpi=300)
    ax = plt.gca()

    # 设置更粗的边框
    for spine in ax.spines.values():
        spine.set_linewidth(1)  # 增加边框宽度
        spine.set_color('black')  # 设置边框颜色为纯黑

    cols = ["Classifier", "Accuracy", "ROC_AUC", "Recall", "Precision", "F1"]
    df = []

    for name, clf in zip(names, classifiers):
        clf.fit(X_train_, y_train_)

        pred = clf.predict(x_test_)
        accuracy = accuracy_score(y_test_, pred)

        pred_proba = clf.predict_proba(x_test_)[:, 1]

        fpr, tpr, thresholds = roc_curve(y_test_, pred_proba)
        roc_auc = auc(fpr, tpr)
        display = RocCurveDisplay(fpr=fpr, tpr=tpr, roc_auc=roc_auc, estimator_name=name)
        display.plot(ax=ax)
        plt.grid(visible=False)

        cm = confusion_matrix(y_test_, pred)

        # recall: TP/(TP+FN)
        recall = cm[1,1]/(cm[1,1] + cm[1,0])

        # precision: TP/(TP+FP)
        precision = cm[1,1]/(cm[1,1] + cm[0,1])

        # F1 score: TP/(TP+FP)
        f1 = 2 * recall * precision / (recall + precision)
        df.append([name, accuracy*100, roc_auc, recall, precision, f1])

    # 设置更大的字体
    font_size = 18
    plt.xticks(fontsize=font_size)  # X轴刻度字体
```

```
    plt.yticks(fontsize=font_size)  # Y轴刻度字体
    plt.xlabel("False Positive Rate", fontsize=font_size)  # X轴标签字体
    plt.ylabel("True Positive Rate", fontsize=font_size)  # Y轴标签字体

    # 设置图例字体大小
    legend = ax.legend(loc="lower right", fontsize=font_size)
    # 设置图例边框更粗更黑
    legend.get_frame().set_linewidth(1)
    legend.get_frame().set_edgecolor("black")

    data_table = pd.DataFrame(data=df, columns=cols)

    return(np.round(data_table.reset_index(drop=True), 2))
```

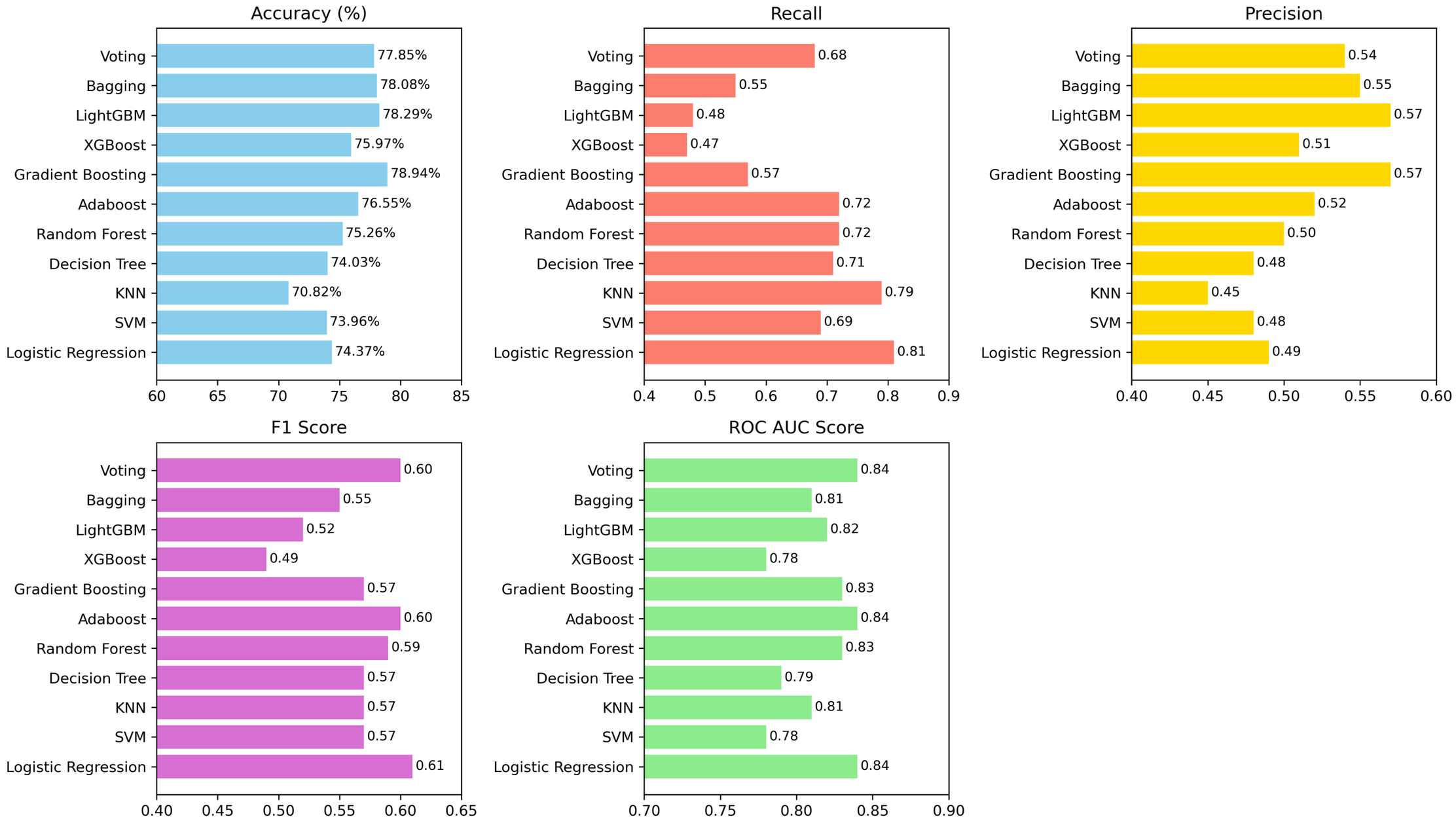
In [ ]:
```
result = score_summary_roc(names,best_estimators,X_train,y_train,X_test,y_test)
```

In [ ]:
```
result
```

Out[ ]:

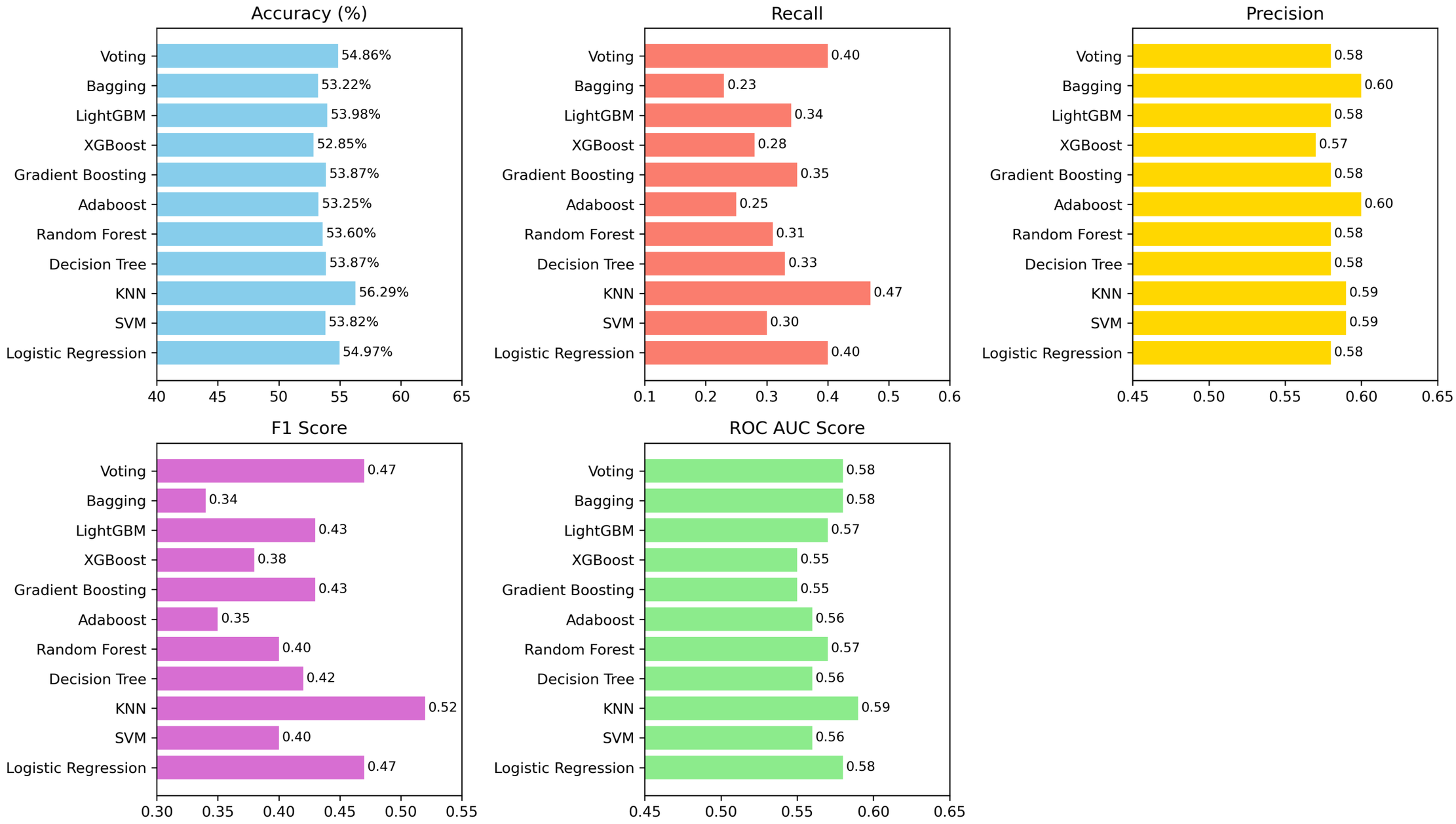
	Classifier	Accuracy	ROC_AUC	Recall	Precision	F1
0	Logistic Regression	74.37	0.84	0.81	0.49	0.61
1	SVM	73.89	0.78	0.68	0.48	0.56
2	KNN	70.82	0.81	0.79	0.45	0.57
3	Decision Tree	73.45	0.78	0.64	0.47	0.54
4	Random Forest	75.66	0.83	0.70	0.50	0.59
5	Adaboost	76.24	0.83	0.70	0.51	0.59
6	Gradient Boosting	77.74	0.83	0.65	0.54	0.59
7	XGBoost	79.21	0.84	0.56	0.58	0.57
8	LightGBM	79.52	0.83	0.57	0.58	0.58
9	Bagging	78.08	0.81	0.55	0.55	0.55
10	Voting	77.85	0.84	0.68	0.54	0.60

In [ ]:
```
import matplotlib.pyplot as plt
import pandas as pd

# Your original data
data = {
    "Classifier": ["Logistic Regression", "SVM", "KNN", "Decision Tree", "Random Forest",
                   "Adaboost", "Gradient Boosting", "XGBoost", "LightGBM", "Bagging", "Voting"],
    "Accuracy": [74.37, 73.96, 70.82, 74.03, 75.26, 76.55, 78.94, 75.97, 78.29, 78.08, 77.85],
    "ROC_AUC": [0.84, 0.78, 0.81, 0.79, 0.83, 0.84, 0.83, 0.78, 0.82, 0.81, 0.84],
    "Recall": [0.81, 0.69, 0.79, 0.71, 0.72, 0.72, 0.57, 0.47, 0.48, 0.55, 0.68],
    "Precision": [0.49, 0.48, 0.45, 0.48, 0.50, 0.52, 0.57, 0.51, 0.57, 0.55, 0.54],
    "F1": [0.61, 0.57, 0.57, 0.57, 0.59, 0.60, 0.57, 0.49, 0.52, 0.55, 0.60]
}

df = pd.DataFrame(data)

# Set white background and remove grid
plt.style.use('default')
plt.rcParams['figure.facecolor'] = 'white'
plt.rcParams['axes.facecolor'] = 'white'

# Plotting the metrics
plt.figure(figsize=(14, 8),dpi=1200)

# Helper function to add value labels
def add_value_labels(bars, xlim_range, is_percent=False):
    for bar in bars:
        width = bar.get_width()
        label = f"{width:.2f}%" if is_percent else f"{width:.2f}"
        plt.text(width + 0.01 * xlim_range,
                 bar.get_y() + bar.get_height()/2,
                 label,
                 va='center',
                 ha='left',
                 fontsize=9)

# Accuracy plot
plt.subplot(2, 3, 1)
bars = plt.barh(df['Classifier'], df['Accuracy'], color='skyblue')
plt.title('Accuracy (%)')
plt.xlim(60, 85)
add_value_labels(bars, 25, is_percent=True)

# ROC_AUC plot
plt.subplot(2, 3, 5)
bars = plt.barh(df['Classifier'], df['ROC_AUC'], color='lightgreen')
plt.title('ROC AUC Score')
plt.xlim(0.7, 0.9)
add_value_labels(bars, 0.2)

# Recall plot
plt.subplot(2, 3, 2)
bars = plt.barh(df['Classifier'], df['Recall'], color='salmon')
plt.title('Recall')
plt.xlim(0.4, 0.9)
add_value_labels(bars, 0.5)

# Precision plot
plt.subplot(2, 3, 3)
bars = plt.barh(df['Classifier'], df['Precision'], color='gold')
plt.title('Precision')
plt.xlim(0.4, 0.6)
add_value_labels(bars, 0.2)

# F1 plot
plt.subplot(2, 3, 4)
bars = plt.barh(df['Classifier'], df['F1'], color='orchid')
plt.title('F1 Score')
plt.xlim(0.4, 0.65)
add_value_labels(bars, 0.25)

plt.tight_layout()
plt.suptitle('Comparison of Internal Data Classifier Performance Metrics', y=1.02, fontsize=14)
plt.show()
```



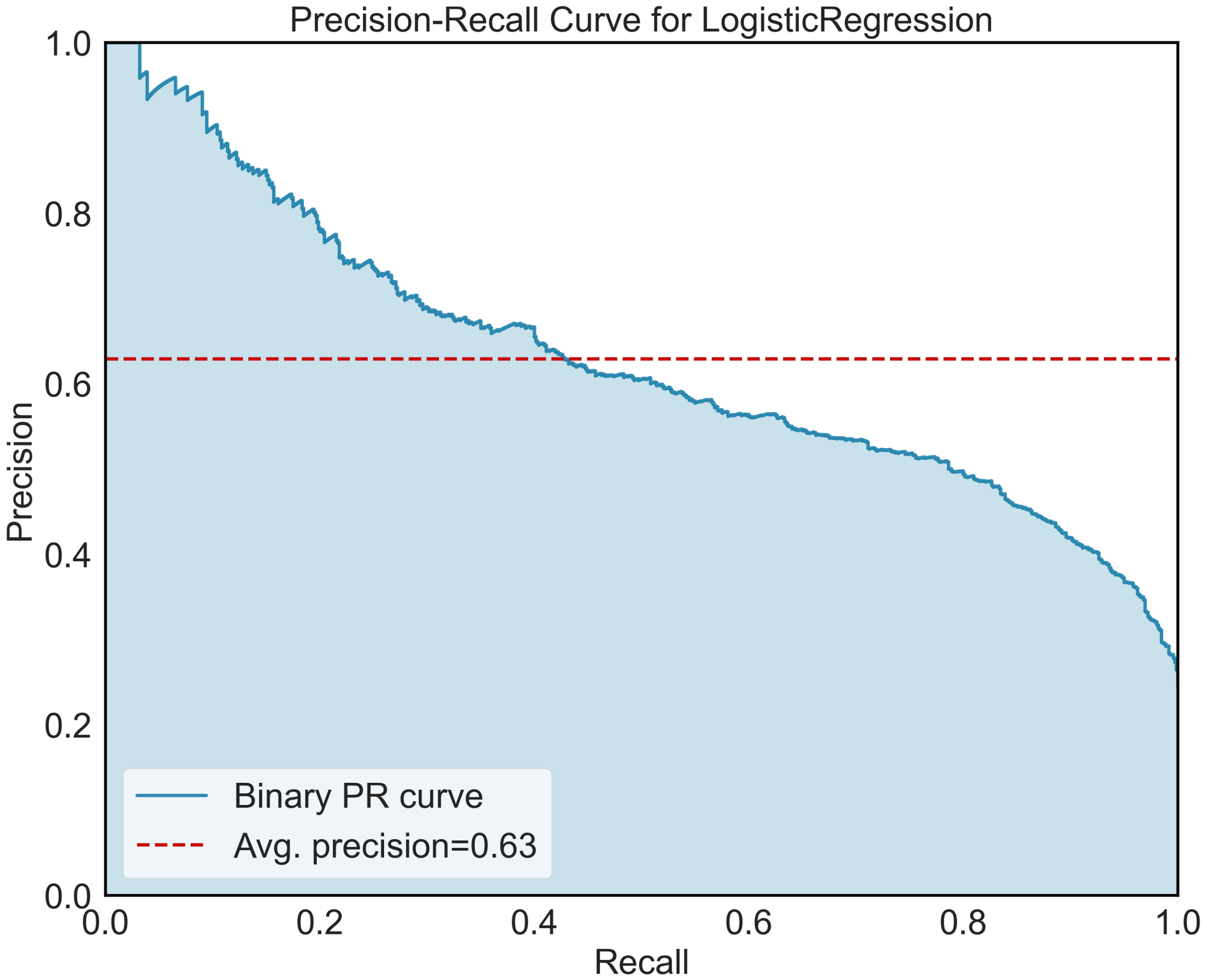
In [ ]:

```python
import matplotlib.pyplot as plt
import pandas as pd

# Your original data
data = {
    "Classifier": ["Logistic Regression", "SVM", "KNN", "Decision Tree", "Random Forest",
                   "Adaboost", "Gradient Boosting", "XGBoost", "LightGBM", "Bagging", "Voting"],
    "Accuracy": [54.97, 53.82, 56.29, 53.87, 53.60, 53.25, 53.87, 52.85, 53.98, 53.22, 54.86],
    "ROC_AUC": [0.58, 0.56, 0.59, 0.56, 0.57, 0.56, 0.55, 0.55, 0.57, 0.58, 0.58],
    "Precision": [0.58, 0.59, 0.59, 0.58, 0.58, 0.60, 0.58, 0.57, 0.58, 0.60, 0.58],
    "Recall": [0.40, 0.30, 0.47, 0.33, 0.31, 0.25, 0.35, 0.28, 0.34, 0.23, 0.40],
    "F1": [0.47, 0.40, 0.52, 0.42, 0.40, 0.35, 0.43, 0.38, 0.43, 0.34, 0.47]
}

df = pd.DataFrame(data)

# Set white background and remove grid
plt.style.use('default')
plt.rcParams['figure.facecolor'] = 'white'
plt.rcParams['axes.facecolor'] = 'white'

# Plotting the metrics
plt.figure(figsize=(14, 8),dpi=300)

# Helper function to add value labels
def add_value_labels(bars, xlim_range, is_percent=False):
    for bar in bars:
        width = bar.get_width()
        label = f"{width:.2f}%" if is_percent else f"{width:.2f}"
        plt.text(width + 0.01 * xlim_range,
                 bar.get_y() + bar.get_height()/2,
                 label,
                 va='center',
                 ha='left',
                 fontsize=9)

# Accuracy plot
plt.subplot(2, 3, 1)
bars = plt.barh(df['Classifier'], df['Accuracy'], color='skyblue')
plt.title('Accuracy (%)')
plt.xlim(40, 65)
add_value_labels(bars, 25, is_percent=True)

# ROC_AUC plot
plt.subplot(2, 3, 5)
bars = plt.barh(df['Classifier'], df['ROC_AUC'], color='lightgreen')
plt.title('ROC AUC Score')
plt.xlim(0.45, 0.65)
add_value_labels(bars, 0.2)

# Recall plot
plt.subplot(2, 3, 2)
bars = plt.barh(df['Classifier'], df['Recall'], color='salmon')
plt.title('Recall')
plt.xlim(0.1, 0.6)
add_value_labels(bars, 0.5)

# Precision plot
plt.subplot(2, 3, 3)
bars = plt.barh(df['Classifier'], df['Precision'], color='gold')
plt.title('Precision')
plt.xlim(0.45, 0.65)
add_value_labels(bars, 0.2)

# F1 plot
plt.subplot(2, 3, 4)
bars = plt.barh(df['Classifier'], df['F1'], color='orchid')
plt.title('F1 Score')
plt.xlim(0.3, 0.55)
add_value_labels(bars, 0.25)

plt.tight_layout()
plt.suptitle('Comparison of External Data Classifier Performance Metrics', y=1.02, fontsize=14)
plt.show()
```



In [ ]:

```python
import matplotlib.pyplot as plt
from yellowbrick.classifier import PrecisionRecallCurve

# 假设best_estimators, X_train, y_train, X_test, y_test已经定义
model = best_estimators[0]
viz = PrecisionRecallCurve(model)

# 在fit和score之前设置figure的大小和分辨率
plt.figure(figsize=(10, 8), dpi=300)
plt.rcParams.update({
    'axes.labelsize': 18,          # 坐标轴标签字体大小
    'xtick.labelsize': 18,         # X轴刻度字体大小
    'ytick.labelsize': 18,         # Y轴刻度字体大小
    'legend.fontsize': 18,         # 图例字体大小
})

ax = plt.gca()
for spine in ax.spines.values():
    spine.set_color('black')     # 设置边框为深黑色
    spine.set_linewidth(1.5)


viz.fit(X_train, y_train)
viz.score(X_test, y_test)
viz.show()
```



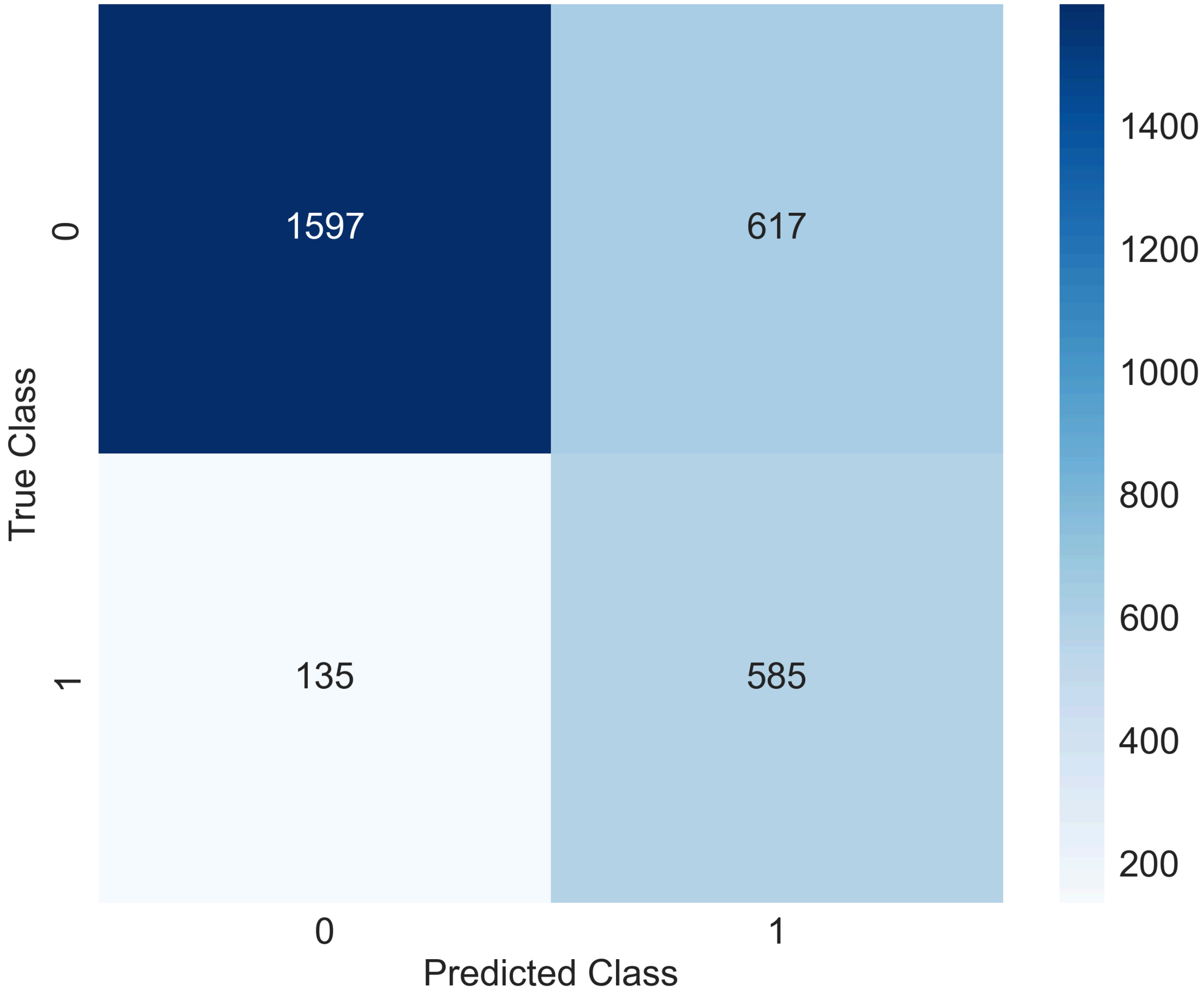
Out[ ]: <Axes: title={'center': 'Precision-Recall Curve for LogisticRegression'}, xlabel='Recall', ylabel='Precision'>

In [ ]:
```
names
```

Out[ ]:
```
['Logistic Regression',
 'SVM',
 'KNN',
 'Decision Tree',
 'Random Forest',
 'Adaboost',
 'Gradient Boosting',
 'XGBoost',
 'LightGBM',
 'Bagging',
 'Voting']
```

混淆矩阵

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import seaborn as sns  # 可选：使用seaborn来美化图表
# 假设bestestimators[0]是一个已经训练好的分类器
# Xtrain, ytrain是训练数据，Xtest, ytest是测试数据
# 以下代码将替换Yellowbrick的混淆矩阵可视化
# 计算混淆矩阵
cm = confusion_matrix(y_test, best_estimators[0].predict(X_test))
# 绘制混淆矩阵
plt.figure(figsize=(10, 8), dpi=300)  # 设置图像大小和分辨率
plt.rcParams.update({
    'font.size': 18,           # 全局字体大小
    'axes.titlesize': 18,       # 标题大小
    'axes.labelsize': 18,       # 坐标轴标签大小
    'xtick.labelsize': 18,      # X轴刻度大小
    'ytick.labelsize': 18,      # Y轴刻度大小
    'figure.dpi': 300,          # 分辨率
})


sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')  # 使用seaborn的heatmap函数
plt.xlabel('Predicted Class')
plt.ylabel('True Class')
plt.show()  # 显示图像
```



校准曲线

In [ ]:
```python
import matplotlib.pyplot as plt
from sklearn.calibration import CalibrationDisplay, calibration_curve
from sklearn.metrics import brier_score_loss

# 假设 best_estimators 是一个包含已经训练好的分类器的列表
# X_test 和 y_test 是测试数据集的特征和标签

fig, ax = plt.subplots(figsize=(12, 12), dpi=300)  # 创建图形和坐标轴

# 遍历每个分类器并绘制校准曲线
for clf in best_estimators:
    # 计算Brier score
    y_prob = clf.predict_proba(X_test)[:, 1]
    brier_score = brier_score_loss(y_test, y_prob)

    # 创建CalibrationDisplay对象并绘制校准曲线
    display = CalibrationDisplay.from_estimator(
        clf, X_test, y_test, ax=ax, name=f'{clf.__class__.__name__} (Brier: {brier_score:.3f})'
    )

plt.grid(False)
plt.show()
```

SHAP---模型可解释性分析

In [ ]:
```python
best_estimators
```

In [ ]:
```python
import shap
X_train = pd.DataFrame(X_train,columns=X_1.columns)
model = best_estimators[0] # 选择一个模型

# compute SHAP values
explainer = shap.Explainer(model, X_train)
shap_values = explainer(X_train)
```

In [ ]:
```python
shap.plots.violin(shap_values,  plot_type="layered_violin")
```

In [ ]:
```python
plt.figure(figsize=(10,10), dpi=300)
plt.grid(False)
shap.plots.bar(shap_values,max_display=20)
```

In [ ]:
```python
import shap
import matplotlib.pyplot as plt
# 构建 shap解释器
explainer = shap.Explainer(model, X_train)
# 计算测试集的shap值
shap_values = explainer(X_train)
# 特征标签
labels = X_train.columns
# 设置字体和字体大小
```

```
plt.rcParams['font.family'] = 'serif'
plt.rcParams['font.serif'] = 'Times new Roman'
plt.rcParams['font.size'] = 13
# 禁用网格线
plt.rcParams['axes.grid'] = False
# 设置图表大小和分辨率
plt.figure(figsize=(10, 10), dpi=300)
# 生成SHAP摘要图
shap.summary_plot(shap_values, X_train, feature_names=labels, plot_type="dot")
# 显示图表
plt.show()
```

In [ ]:
```
import joblib
joblib.dump(model, '0511重置版saki_lr_model1.pkl')  # 假设 model 是训练好的模型
```

Out[ ]:
```
['0511重置版saki_lr_model1.pkl']
```

In [ ]:
```
from sklearn.linear_model import LogisticRegression
import joblib

# 假设你已经有了一个训练好的模型
model = LogisticRegression()
model.fit(X_train, y_train)

# 使用joblib保存模型
joblib.dump(model, 'model1.joblib')
```

Out[ ]:
```
['model1.joblib']
```

In [ ]:
```
shap_values_top100 = shap_values[:1000]

# 绘制前100个样本的SHAP值热力图
shap.plots.heatmap(shap_values_top100)
```

In [ ]:
```
shap.plots.heatmap(shap_values)
```

In [ ]:
```
explanation = explainer(X_train.loc[:1000,:])
```

In [ ]:
```
X_train
```

In [ ]:
```
shap.plots.scatter(explanation[:, "Scr_baseline"])
```

In [ ]:
```
shap.plots.scatter(explanation[:, "Scr_baseline"], color=explanation[:, "Age"])
```

In [ ]:

In [ ]:
```
shap.plots.waterfall(shap_values[0])
```

In [ ]:
```
shap.plots.waterfall(shap_values[1])
```

In [ ]:
```
shap.plots.waterfall(shap_values[3])
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:
```
from lime.lime_tabular import LimeTabularExplainer
```

In [ ]:
```
explainer = LimeTabularExplainer(X_train.values, feature_names=X_1.columns, class_names=np.unique(y_train), mode='classification')
```

In [ ]:
```
instance = X_train.loc[11]

 # Generate an explanation for the instance
explanation = explainer.explain_instance(instance,model.predict_proba, num_features=5)

 # Display the explanation
explanation.show_in_notebook()
```

In [ ]:

In [ ]:
```
import joblib
joblib.dump(model, 'saki_lr_model.pkl')  # 假设 model 是训练好的模型
```

Out[ ]:
```
['saki_lr_model.pkl']
```

In [ ]:
```
coefficients = model.coef_[0]
intercept = model.intercept_[0]
```

In [ ]:
```
coefficients
```

Out[ ]:
```
array([-0.38681916, -0.01213823, -0.16050141,  0.36487372,  0.29474987,
        0.4989831 ,  0.31881465,  0.18528972,  0.35857963,  0.17374482,
        0.20034995,  0.04636802, -0.2696443 ,  0.1108195 , -0.48366887,
        0.17549857,  0.1038173 ,  0.41248211,  0.16337912,  0.16645699])
```

In [ ]:

In [64]:
```
final= pd.read_csv(r'E:\MIMIC\MIMIC_BIG_DATA\MIMIC_BIG_DATA\case\sepsis\250510 EICU数据 saki.csv')
final.head()
```

Out[64]:

	Unnamed: 0	patientunitstayid	Scr_baseline	akimintime	akimaxtime	los_aki	AKD	cxx_aki	diagnosisoffset	diagnosisstring	...	dementia1	copd1	ctd1	pud1	liver1	age_score_charlson	fin
0	0	141304	1.24	-127.0	7293	7420	0	1	23.0	cardiovascular|shock / hypotension|sepsis	...	0	0	0	0	0	3	
1	1	141751	0.90	-101.0	6042	6143	0	1	45.0	cardiovascular|shock / hypotension|signs and s...	...	0	0	0	0	0	2	
2	2	141920	1.03	-113.0	2469	2582	0	0	15.0	cardiovascular|shock / hypotension|sepsis	...	0	0	0	0	0	4	
3	3	141945	2.17	-205.0	6539	6744	0	1	25.0	cardiovascular|shock / hypotension|sepsis	...	0	0	0	0	0	3	
4	4	142388	1.11	-94.0	7122	7216	0	1	73.0	cardiovascular|shock / hypotension|sepsis	...	0	0	0	0	0	2	

5 rows × 137 columns

In [65]:
```
print(list(final.columns))
```

```
['Unnamed: 0', 'patientunitstayid', 'Scr_baseline', 'akimintime', 'akimaxtime', 'los_aki', 'AKD', 'cxx_aki', 'diagnosisoffset', 'diagnosisstring', 'patienthealthsystemstayid', 'unitvisitnumbe
r', 'hospitalid', 'hospitaladmitoffset', 'hospitaldischargeoffset', 'unitadmitoffset', 'unitdischargeoffset', 'apache_iv', 'hospitaldischargeyear', 'age', 'hosp_mort', 'gender', 'admissionheig
ht', 'admissionweight', 'icu_los_hours', 'aniongap_min', 'aniongap_max', 'albumin_min', 'albumin_max', 'bicarbonate_min', 'bicarbonate_max', 'bilirubin_min', 'bilirubin_max', 'creatinine_min',
'creatinine_max', 'chloride_min', 'chloride_max', 'glucose_min', 'glucose_max', 'hematocrit_min', 'hematocrit_max', 'hemoglobin_min', 'hemoglobin_max', 'lactate_min', 'lactate_max', 'platelet_
min', 'platelet_max', 'potassium_min', 'potassium_max', 'inr_min', 'inr_max', 'pt_min', 'pt_max', 'sodium_min', 'sodium_max', 'bun_min', 'bun_max', 'wbc_min', 'wbc_max', 'window_start7', 'CRR
T', 'apachepatientresultsid', 'physicianspeciality', 'physicianinterventioncategory', 'acutephysiologyscore', 'apachescore', 'apacheversion', 'predictedicumortality', 'actualicumortality', 'pr
edictediculos', 'actualiculos', 'predictedhospitalmortality', 'actualhospitalmortality', 'predictedhospitallos', 'actualhospitallos', 'preopmi', 'preopcardiaccath', 'ptcawithin24h', 'unabridge
dunitlos', 'unabridgedhosplos', 'apacheapsvarid', 'intubated', 'vent', 'dialysis', 'eyes', 'motor', 'verbal', 'meds', 'wbc', 'sodium', 'meanbp', 'hematocrit', 'creatinine', 'bun', 'glucose',
'oasis', 'vasopressor', 'ACEI/ARB', 'GCS', 'chartoffset', 'heartrate', 'respiratoryrate', 'spo2', 'nibp_systolic', 'nibp_diastolic', 'temperature', 'PT', 'paO2', 'FiO2', 'PaO2/FiO2', 'LODS',
'BaseExcess', 'drugstartoffset', 'Los_inf._AB', 'mets6', 'aids6', 'liver3', 'stroke2', 'renal2', 'dm', 'cancer2', 'leukemia2', 'lymphoma2', 'mi1', 'chf1', 'pvd1', 'tia1', 'dementia1', 'copd1',
'ctd1', 'pud1', 'liver1', 'age_score_charlson', 'final_charlson_score', 'Cerebrovascular_disease', 'paCO2', 'paraplegia']
```

print(final['Scr_baseline'].value_counts().to_string())

In [66]:
```
final['MBP'] = final['nibp_diastolic'] + (final['nibp_systolic'] - final['nibp_diastolic']) / 3
```

In [67]:
```
final=final[['patientunitstayid','Scr_baseline','AKD','cxx_aki','apache_iv', 'age','CRRT', 'LODS', 'BaseExcess','Los_inf._AB','vent',
             'temperature','respiratoryrate', 'ACEI/ARB','vasopressor','bun', 'glucose', 'oasis','wbc', 'sodium','acutephysiologyscore',
             'Cerebrovascular_disease','admissionweight','paO2','paCO2','spo2','MBP','paraplegia','age','gender']]
```

In [68]:
```
print(list(final.columns))
```

```
['patientunitstayid', 'Scr_baseline', 'AKD', 'cxx_aki', 'apache_iv', 'age', 'CRRT', 'LODS', 'BaseExcess', 'Los_inf._AB', 'vent', 'temperature', 'respiratoryrate', 'ACEI/ARB', 'vasopressor', 'b
un', 'glucose', 'oasis', 'wbc', 'sodium', 'acutephysiologyscore', 'Cerebrovascular_disease', 'admissionweight', 'paO2', 'paCO2', 'spo2', 'MBP', 'paraplegia', 'age', 'gender']
```

In [69]:
```
X_1.columns
```

Out[69]:
```
Index(['ACEI/ARB', 'APS III', 'CRRT', 'Cerebrovascular Disease', 'LODS',
       'Los_inf._AB', 'MBP', 'Mechanical Ventilation', 'Paraplegia',
       'Resp Rate', 'Scr Baseline', 'SpO2', 'Vasoactive Agent', 'Weight'],
      dtype='object')
```

In [70]:
```
final1=final[['Scr_baseline', 'AKD',  'CRRT', 'LODS', 'vent',  'respiratoryrate', 'ACEI/ARB', 'vasopressor',
              'acutephysiologyscore', 'Cerebrovascular_disease', 'admissionweight', 'spo2','MBP','Los_inf._AB','paraplegia','age','gender']]
```

In [71]:
```
rename_mapping = {
    'Scr_baseline': 'Scr Baseline',
    'CRRT': 'CRRT',
    'LODS': 'LODS',
    'respiratoryrate': 'Resp Rate',
    'ACEI/ARB': 'ACEI/ARB',
    'vasopressor': 'Vasoactive Agent',
    'acutephysiologyscore': 'APS III',
    'Cerebrovascular_disease': 'Cerebrovascular Disease',
    'admissionweight': 'Weight',
    'spo2':'SpO2',
    'vent':'Mechanical Ventilation',
    'paraplegia':'Paraplegia',
    'age':'Age',
    'gender':'Gender'
}

# 使用rename方法替换变量名
final2= final1.rename(columns=rename_mapping)
```

In [72]:
```
final2.head()
```

Out[72]:

	Scr Baseline	AKD	CRRT	LODS	Mechanical Ventilation	Resp Rate	ACEI/ARB	Vasoactive Agent	APS III	Cerebrovascular Disease	Weight	SpO2	MBP	Los_inf._AB	Paraplegia	Age	Age	Gender
0	1.24	0	0	4	1.0	28.0	0	0	41.0	0	NaN	95.0	72.000000	0.00	0	70.0	70.0	1.0
1	0.90	0	0	8	1.0	25.0	0	0	NaN	0	NaN	94.0	57.666667	0.19	0	60.0	60.0	0.0
2	1.03	0	0	4	1.0	28.0	0	0	41.0	0	53.4	97.0	67.000000	0.00	0	81.0	81.0	0.0
3	2.17	0	0	1	0.0	14.0	0	0	40.0	0	74.8	69.0	67.666667	0.00	0	72.0	72.0	0.0
4	1.11	0	0	7	1.0	22.0	0	0	101.0	0	96.6	97.0	144.333333	0.86	0	67.0	67.0	1.0

In [73]:
```
threshold = 0.9 * len(final2.columns)
final2_cleaned = final2.dropna(thresh=threshold)

final2_cleaned.shape
```

Out[73]:
```
(4842, 18)
```

In [74]:
```
final2_cleaned['AKD'].value_counts()
```

Out[74]:
```
AKD
1    2463
0    2379
Name: count, dtype: int64
```

In [75]:
```
threshold = 1 * len(final2.columns)
final3 = final2.dropna(thresh=threshold)

final3.shape
```

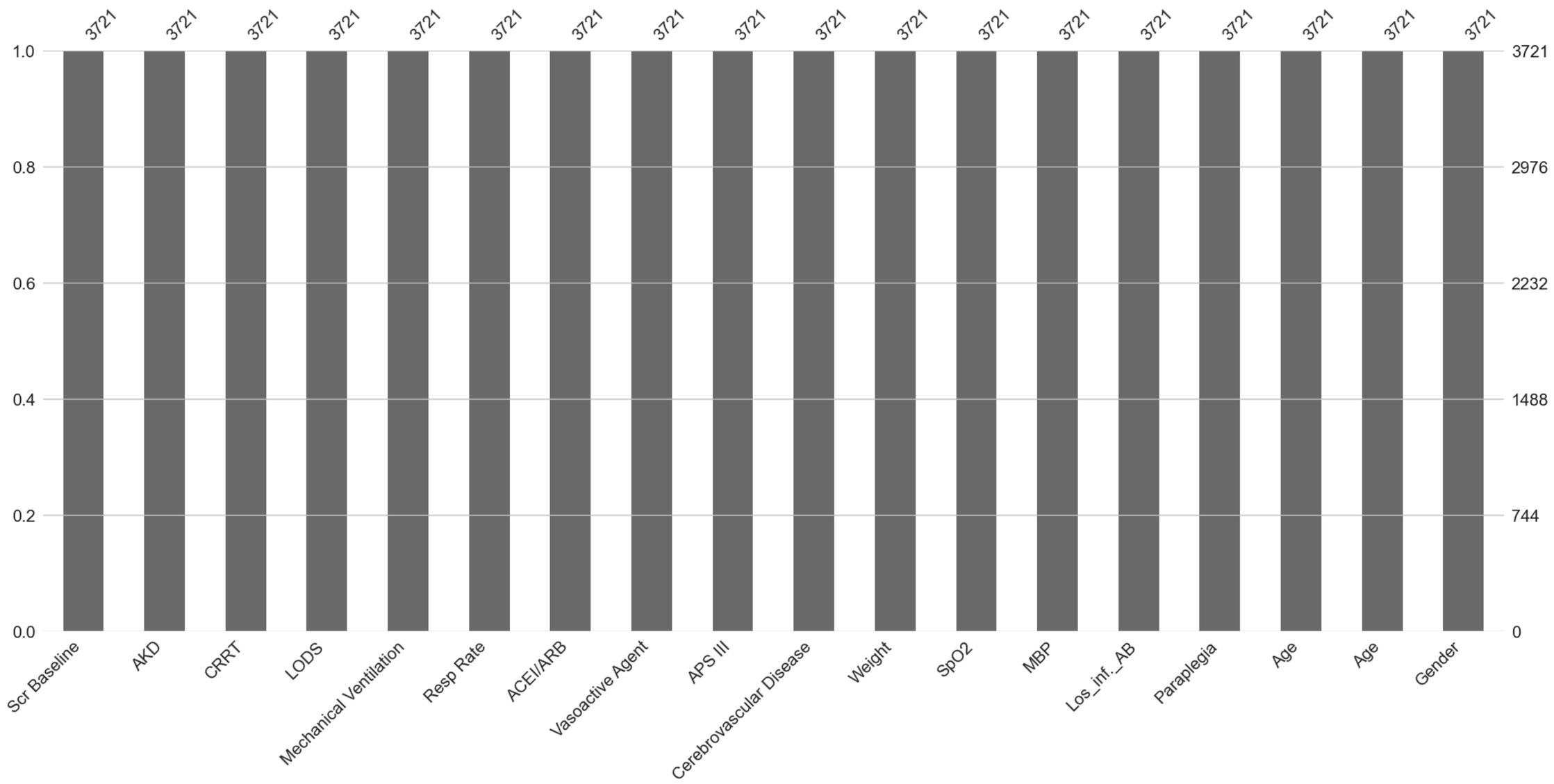
Out[75]:
```
(3721, 18)
```

In [76]:
```
final3['AKD'].value_counts()
```

Out[76]:
```
AKD
1    1894
0    1827
Name: count, dtype: int64
```

In [77]:
```
msno.bar(final3)
```

Out[77]:
```
<Axes: >
```



In [ ]:
```
final3['Scr Baseline'] = final3['Scr Baseline'] * 88.4
```

In [79]:
```
final3.head()
```

Out[79]:

	Scr Baseline	AKD	CRRT	LODS	Mechanical Ventilation	Resp Rate	ACEI/ARB	Vasoactive Agent	APS III	Cerebrovascular Disease	Weight	SpO2	MBP	Los_inf._AB	Paraplegia	Age	Age	Gender
2	91.052	0	0	4	1.0	28.0	0	0	41.0	0	53.4	97.0	67.000000	0.00	0	81.0	81.0	0.0
3	191.828	0	0	1	0.0	14.0	0	0	40.0	0	74.8	69.0	67.666667	0.00	0	72.0	72.0	0.0
4	98.124	0	0	7	1.0	22.0	0	0	101.0	0	96.6	97.0	144.333333	0.86	0	67.0	67.0	1.0
5	109.616	1	0	4	0.0	20.0	0	0	42.0	0	71.4	98.0	71.666667	0.00	0	72.0	72.0	1.0
6	162.656	1	0	7	1.0	20.0	0	0	63.0	0	76.7	99.0	85.666667	0.28	0	60.0	60.0	0.0

In [ ]:

In [80]:
```
X_1.columns
```

Out[80]:
```
Index(['ACEI/ARB', 'APS III', 'CRRT', 'Cerebrovascular Disease', 'LODS',
       'Los_inf._AB', 'MBP', 'Mechanical Ventilation', 'Paraplegia',
       'Resp Rate', 'Scr Baseline', 'SpO2', 'Vasoactive Agent', 'Weight'],
      dtype='object')
```

In [81]:
```
column_order = ['ACEI/ARB', 'APS III' ,'CRRT', 'Cerebrovascular Disease', 'LODS',
                'Los_inf._AB', 'MBP' ,'Mechanical Ventilation', 'Paraplegia', 'Resp Rate',
                'Scr Baseline' ,'SpO2' ,'Vasoactive Agent', 'Weight','AKD'
]

# 按指定顺序排列列
final3 = final3[column_order]
```

In [82]:
```
final3.head()
```

Out[82]:

	ACEI/ARB	APS III	CRRT	Cerebrovascular Disease	LODS	Los_inf._AB	MBP	Mechanical Ventilation	Paraplegia	Resp Rate	Scr Baseline	SpO2	Vasoactive Agent	Weight	AKD
2	0	41.0	0	0	4	0.00	67.000000	1.0	0	28.0	91.052	97.0	0	53.4	0
3	0	40.0	0	0	1	0.00	67.666667	0.0	0	14.0	191.828	69.0	0	74.8	0
4	0	101.0	0	0	7	0.86	144.333333	1.0	0	22.0	98.124	97.0	0	96.6	0
5	0	42.0	0	0	4	0.00	71.666667	0.0	0	20.0	109.616	98.0	0	71.4	1
6	0	63.0	0	0	7	0.28	85.666667	1.0	0	20.0	162.656	99.0	0	76.7	1

final3.to_csv('eICU_saki.csv')

In [ ]:
```
y_val= final3['AKD']
X_val= final3.drop('AKD',axis=1)
```

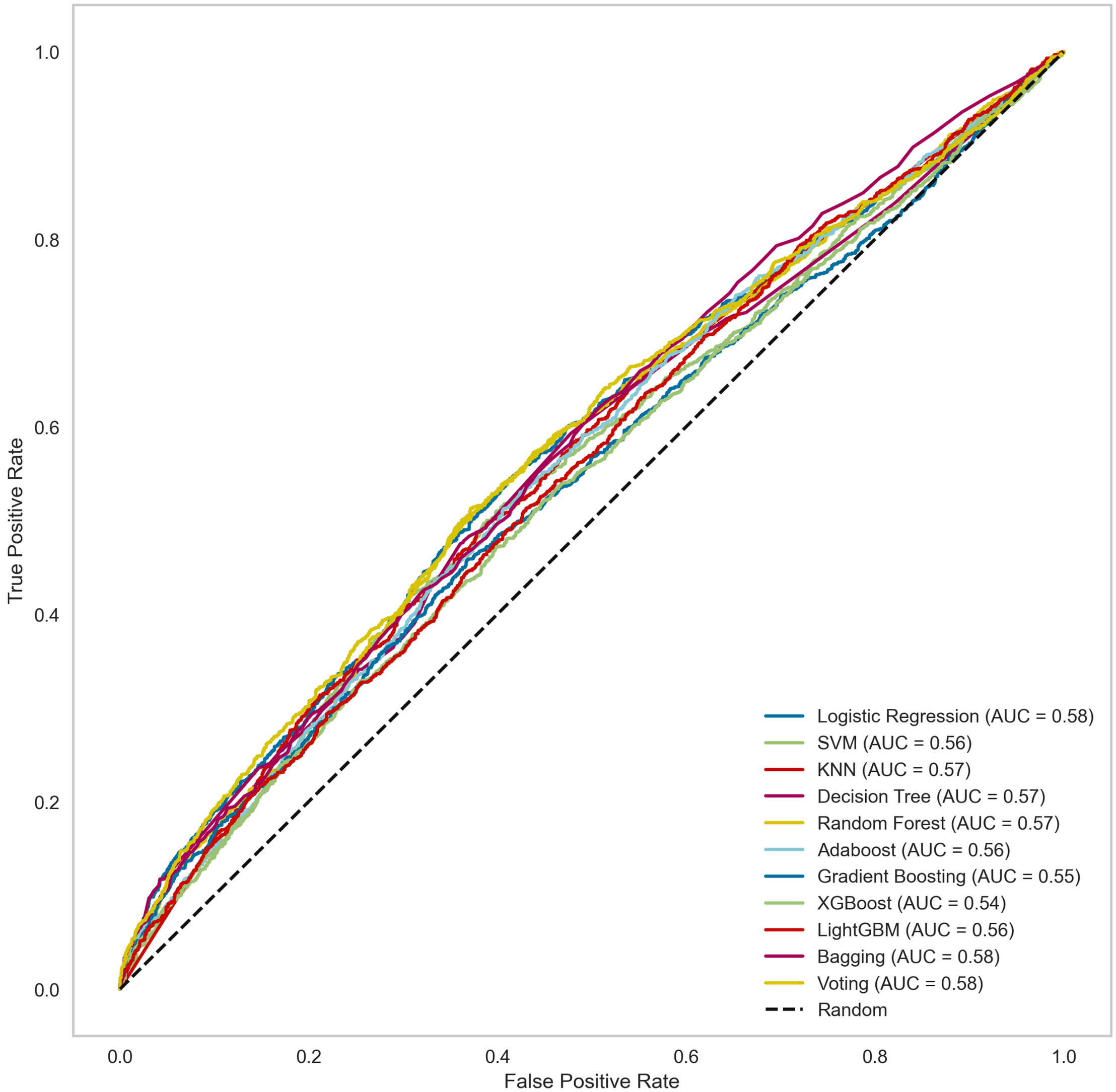
In [ ]:
```
import streamlit as st
import joblib
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [ ]:
```
X_val= scaler.transform(X_val)
```

In [ ]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, roc_curve
import matplotlib.pyplot as plt

# 假设 best_estimators 是一个包含已经训练好的分类器的列表
model_names = ['Logistic Regression','SVM','KNN','Decision Tree','Random Forest',
               'Adaboost','Gradient Boosting','XGBoost','LightGBM','Bagging','Voting']  # 模型名称列表

# 初始化性能指标字典
performance_metrics = {}

# 对每个分类器进行评估
for name, model in zip(model_names, best_estimators):
    y_pred = model.predict(X_val)
    y_pred_proba = model.predict_proba(X_val)[:, 1]

    # 计算性能指标
    accuracy = accuracy_score(y_val, y_pred)
    precision = precision_score(y_val, y_pred)
    recall = recall_score(y_val, y_pred)
    f1 = f1_score(y_val, y_pred)
    auc_score = roc_auc_score(y_val, y_pred_proba)
```

```
    # 存储性能指标
    performance_metrics[name] = {
        'Accuracy': accuracy,
        'Precision': precision,
        'Recall': recall,
        'F1 Score': f1,
        'AUC': auc_score
    }

# 打印性能指标
for name, metrics in performance_metrics.items():
    print(f"{name} Performance:")
    for metric, value in metrics.items():
        print(f"{metric}: {value:.4f}")
    print()

# 绘制ROC曲线
plt.figure(figsize=(10, 10), dpi=300)
for name, model in zip(model_names, best_estimators):
    y_pred_proba = model.predict_proba(X_val)[:, 1]
    fpr, tpr, _ = roc_curve(y_val, y_pred_proba)
    auc_score = performance_metrics[name]['AUC']
    plt.plot(fpr, tpr, label=f'{name} (AUC = {auc_score:.2f})')

plt.plot([0, 1], [0, 1], 'k--', label='Random')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc="lower right")
plt.grid(False)
plt.show()
```

```
Logistic Regression Performance:
Accuracy: 0.5496
Precision: 0.5846
Recall: 0.3976
F1 Score: 0.4733
AUC: 0.5781

SVM Performance:
Accuracy: 0.5394
Precision: 0.5939
Recall: 0.3004
F1 Score: 0.3990
AUC: 0.5573

KNN Performance:
Accuracy: 0.5474
Precision: 0.5772
Recall: 0.4145
F1 Score: 0.4825
AUC: 0.5674

Decision Tree Performance:
Accuracy: 0.5370
Precision: 0.5861
Recall: 0.3073
F1 Score: 0.4032
AUC: 0.5657

Random Forest Performance:
Accuracy: 0.5361
Precision: 0.5817
Recall: 0.3157
F1 Score: 0.4093
AUC: 0.5739

Adaboost Performance:
Accuracy: 0.5284
Precision: 0.5918
Recall: 0.2365
F1 Score: 0.3380
AUC: 0.5647

Gradient Boosting Performance:
Accuracy: 0.5289
Precision: 0.5809
Recall: 0.2672
F1 Score: 0.3660
AUC: 0.5468

XGBoost Performance:
Accuracy: 0.5257
Precision: 0.5722
Recall: 0.2698
F1 Score: 0.3667
AUC: 0.5432

LightGBM Performance:
Accuracy: 0.5292
Precision: 0.5632
Recall: 0.3342
F1 Score: 0.4195
AUC: 0.5559

Bagging Performance:
Accuracy: 0.5321
Precision: 0.6041
Recall: 0.2344
F1 Score: 0.3378
AUC: 0.5778

Voting Performance:
Accuracy: 0.5485
Precision: 0.5821
Recall: 0.4007
F1 Score: 0.4747
AUC: 0.5816
```



In [ ]:

In [ ]:

In [ ]:
```python
from PIL import Image, ImageDraw, ImageFont

# 打开两张图片，使用原始字符串或双反斜杠
image1 = Image.open(r'E:\MIMIC\image\Feature Selection by Boruta and Lassov2.png')
image2 = Image.open(r'E:\MIMIC\image\Feature Selection by Boruta and Lassov4.png')
# 确保两张图片的尺寸一致
width, height = image1.size
image1 = image1.resize((width, height))
image2 = image2.resize((width, height))

# 创建一个新图片，宽度为两张图片宽度之和，高度与单张图片相同
total_width = image1.width + image2.width
max_height = max(image1.height, image2.height)
new_image = Image.new('RGB', (total_width, max_height))

# 拼接图片
new_image.paste(image1, (0, 0))
new_image.paste(image2, (image1.width, 0))

# 创建绘图对象
draw = ImageDraw.Draw(new_image)

# 定义字体和大小，这里使用PIL内置的字体
font = ImageFont.load_default()

# 在每张图片的左上角标注"A"和"B"
text_position1 = (10, 10)  # "A"的位置
text_position2 = (image1.width + 10, 10)  # "B"的位置
draw.text(text_position1, "A", font=font, fill=(255, 0, 0))
draw.text(text_position2, "B", font=font, fill=(255, 0, 0))

# 显示拼接后的图片
new_image.show()
```

In [ ]:
```python
new_image.save('E:\MIMIC\image\Boruta_Lasso v1.png')
```

```
<>:1: SyntaxWarning: invalid escape sequence '\M'
<>:1: SyntaxWarning: invalid escape sequence '\M'
C:\Users\Lenovo\AppData\Local\Temp\ipykernel_23624\867099026.py:1: SyntaxWarning: invalid escape sequence '\M'
  new_image.save('E:\MIMIC\image\Boruta_Lasso v1.png')
```

In [ ]:
```
from PIL import Image, ImageDraw, ImageFont

# 打开两张图片，使用原始字符串或双反斜杠
image1 = Image.open(r'E:\MIMIC\image\Lasso Regressionv2.png')
image2 = Image.open(r'E:\MIMIC\image\Lasso Pathsv2.png')

# 确保两张图片的宽度一致
width = min(image1.width, image2.width)
image1 = image1.resize((width, image1.height))
image2 = image2.resize((width, image2.height))

# 创建一个新图片，宽度与单张图片相同，高度为两张图片高度之和
total_height = image1.height + image2.height
max_width = max(image1.width, image2.width)
new_image = Image.new('RGB', (max_width, total_height))

# 拼接图片
new_image.paste(image1, (0, 0))
new_image.paste(image2, (0, image1.height))

# 创建绘图对象
draw = ImageDraw.Draw(new_image)

# 定义字体和大小，这里使用PIL内置的字体
font = ImageFont.load_default()

# 在每张图片的左上角标注"A"和"B"
text_position1 = (10, 10)  # "A"的位置
text_position2 = (10, image1.height + 10)  # "B"的位置
draw.text(text_position1, "A", font=font, fill=(255, 0, 0))
draw.text(text_position2, "B", font=font, fill=(255, 0, 0))

# 显示拼接后的图片
new_image.show()
```

In [ ]:
```

```

In [ ]:
```
import joblib

# 加载保存的模型
model = joblib.load(r'E:\MIMIC\MIMIC_BIG_DATA\MIMIC_BIG_DATA\case\stroke\0511重置版saki_lr_model1.pkl')

# 现在你可以使用这个模型进行预测
# 例如:
# predictions = model.predict(X_new)
```

In [ ]:
```
model = best_estimators[0]
```

决策分析曲线

In [ ]:
```
from sklearn.metrics import confusion_matrix
# 矢量化计算模型净收益的函数
def compute_net_benefit_model_vectorized(thresholds, y_pred_scores, y_labels):
    # 先将 y_labels 转换为 numpy 数组以避免多维索引问题
    y_labels = np.array(y_labels)
    y_pred_scores = np.array(y_pred_scores)
        # 计算总样本数
    n = len(y_labels)
    # 预分配数组
    net_benefit_model = np.zeros_like(thresholds)
    # 将预测得分和阈值进行广播计算
    y_pred_matrix = (y_pred_scores[:, None] > thresholds).astype(int)
    # 矢量化计算混淆矩阵的元素：TP 和 FP
    tp = (y_pred_matrix & y_labels[:, None]).sum(axis=0)
    fp = ((y_pred_matrix == 1) & (y_labels[:, None] == 0)).sum(axis=0)
    # 计算净收益
    net_benefit_model = (tp / n) - (fp / n) * (thresholds / (1 - thresholds))
    return net_benefit_model
# 矢量化计算 Treat all 策略净收益的函数
def compute_net_benefit_all_vectorized(thresholds, y_labels):
    # 将 y_labels 转换为 numpy 数组以避免多维索引问题
    y_labels = np.array(y_labels)
    # 计算混淆矩阵的元素（基于 Treat all 策略，所有样本被视为正类）
    tn, fp, fn, tp = confusion_matrix(y_labels, y_labels).ravel()
    total = tp + tn
    # 预分配数组
    net_benefit_all = np.zeros_like(thresholds)
    # 矢量化计算净收益
    net_benefit_all = (tp / total) - (tn / total) * (thresholds / (1 - thresholds))
    return net_benefit_all
# 绘制 DCA 的函数
def plot_dca_custom(thresholds, net_benefit_model, net_benefit_all):
    fig, ax = plt.subplots(figsize=(8, 6),dpi=1200)
    # 绘制净收益曲线
    ax.plot(thresholds, net_benefit_model, color='deepskyblue', label='Model')
    ax.plot(thresholds, net_benefit_all, color='black', label='Treat all')
    ax.plot((0, 1), (0, 0), color='#808080', label='Treat none')
    # 填充模型比 Treat all 和 Treat none 优势部分
    y2 = np.maximum(net_benefit_all, 0)
    y1 = np.maximum(net_benefit_model, y2)
    ax.fill_between(thresholds, y1, y2, color='deepskyblue', alpha=0.3)  # 保持原来的填充颜色
    # 美化图表
    ax.set_xlim(0, 1)
    ax.set_ylim(net_benefit_model.min() - 0.15, net_benefit_model.max() + 0.15)
    ax.set_xlabel('Threshold Probability', fontdict={'family': 'Times New Roman', 'fontsize': 15})
    ax.set_ylabel('Net Benefit', fontdict={'family': 'Times New Roman', 'fontsize': 15})
    ax.grid(True)
    ax.legend(loc='upper right')
    plt.savefig("DCA.pdf", bbox_inches='tight')
    plt.show()
# 运行 DCA 分析函数
def run_dca_analysis(model, X_test, y_test):
    # 使用模型预测概率
    y_pred_scores = model.predict_proba(X_test)[:, 1]  # 获得正类的预测概率
    y_labels = y_test  # 使用测试集的真实标签
    # 定义阈值范围
    thresholds = np.arange(0, 1, 0.01)
    # 计算不同阈值下的净收益
    net_benefit_model = compute_net_benefit_model_vectorized(thresholds, y_pred_scores, y_labels)
    net_benefit_all = compute_net_benefit_all_vectorized(thresholds, y_labels)
    # 调用绘图函数
    plot_dca_custom(thresholds, net_benefit_model, net_benefit_all)
```

In [ ]:
```
run_dca_analysis(model, X_test, y_test)
```

In [ ]:
```

```