Chosing the best model for our cleaned dataset

Sigmoid kernel

sigmoid kernel is a good choice for classification problems where the decision boundary is not linear but not too complex. It is a versatile kernel that can handle both numerical and categorical data and can be used with logistic regression, making it a popular choice for many classification problems Bas recommended that i try with the Sigmoid kernel because when we looked at the scatter plots in iteration 0 the data points were not well separated in the feature space, and the NearestNeighbor is have difficulty distinguishing between the different classes. In this case, it would be good to try to use a more complex classification algorithm or kernel function that is capable of capturing non-linear relationships between the features and the target variable

In [ ]:
```
from sklearn.model_selection import train_test_split
import sklearn
import pandas
import seaborn
import numpy as np

from sklearn.preprocessing import StandardScaler
from sklearn import preprocessing
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Load data
columns = ["ID", "PRG", "PL", "PR", "SK", "TS", "M11", "BD2", "Age", "Insurance", "
df = pandas.read_csv("Paitients_Files_Train.csv", names=columns)

# Preprocessing
df.drop(columns=["ID"], inplace=True)
encoder = preprocessing.LabelEncoder()
df["Sepsis"] = encoder.fit_transform(df["Sepsis"])
X = df.iloc[:,:-1]  # input features
y = df.iloc[:, -1]  # target variable (sepsis)
scaler = StandardScaler()
scaler.fit(X)
X_scaled = scaler.transform(X)
df_scaled = pandas.concat([pandas.DataFrame(X_scaled, columns=X.columns), y], axis=

# Remove outliers
candidates = columns[1:-2]
z_scores = np.abs((df[candidates] - df[candidates].mean()) / df[candidates].std())
threshold = 4
df_cleaned = df_scaled[(z_scores < threshold).all(axis=1)]

# Select features and target
features = ["PL", "Age", "M11", "PR"]
target = "Sepsis"
X = df_cleaned[features]
```

```
y = df_cleaned[target]

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Train and evaluate a model with Sigmoid kernel
clf = SVC(kernel='sigmoid')
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
acc = accuracy_score(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)
report = classification_report(y_test, y_pred)
print("Accuracy:", acc)
print("Classification report:\n", report)
```

```
Accuracy: 0.635593220338983
Classification report:
              precision    recall  f1-score   support

           0       0.69      0.77      0.73        75
           1       0.50      0.40      0.44        43

    accuracy                           0.64       118
   macro avg       0.60      0.58      0.59       118
weighted avg       0.62      0.64      0.62       118
```

RandomForest

Random Forest is an ensemble learning method that combines multiple decision trees to improve the model's performance. It can handle both categorical and continuous input features, and it is robust to outliers and noisy data. So i tried setting the outlier threshold higher to see if we can see better accuracy to my surprise there was no significant difference outside the randomness factor

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

clf = RandomForestClassifier(n_estimators=100)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
acc = accuracy_score(y_test, y_pred)
print("Accuracy:", acc)
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.7372881355932204
              precision    recall  f1-score   support

           0       0.79      0.80      0.79        75
           1       0.64      0.63      0.64        43

    accuracy                           0.74       118
   macro avg       0.72      0.71      0.71       118
weighted avg       0.74      0.74      0.74       118
```

LogisticRegression

Logistic Regression is a simple yet effective linear model that can work well for many classification problems. if seams to provide the most consistent results without much tweaking for our dataset that's why we are currently sticking with it for our AI model

In [ ]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

clf = LogisticRegression()
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
acc = accuracy_score(y_test, y_pred)
print("Accuracy:", acc)
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.7966101694915254
              precision    recall  f1-score   support

           0       0.81      0.89      0.85        75
           1       0.77      0.63      0.69        43

    accuracy                           0.80       118
   macro avg       0.79      0.76      0.77       118
weighted avg       0.79      0.80      0.79       118
```

Radial Basis Function

Classification issues with nonlinear decision boundaries can be effectively handled by the SVM algorithm using an RBF kernel. The RBF kernel makes it possible to separate classes that are not linearly separable in the original feature space.

the model preforms way better than sigmoid kernel and NearestNeighbor but still subpar to logisticRegression and RandomForest i thought it would be nice to try it as well since i found it recommended on the web and in the feature if we find that we have overfitting problems when we test with real world data we can try switching to it (currently the model doesn't look overfitted)

In [ ]:

```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

clf = SVC(kernel='rbf')
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
acc = accuracy_score(y_test, y_pred)
print("Accuracy:", acc)
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.788135593220339
              precision    recall  f1-score   support

           0       0.78      0.93      0.85        75
           1       0.82      0.53      0.65        43

    accuracy                           0.79       118
   macro avg       0.80      0.73      0.75       118
weighted avg       0.79      0.79      0.78       118
```