Problem Statement: Predicting Chronic Kidney Disease (CKD) in patients

Objective: The objective is to develop a supervised machine learning classification model that can accurately predict whether a patient has chronic kidney disease (CKD) based on several medical parameters provided in a labelled dataset. This model will assist healthcare providers in early detection and timely intervention, ultimately improving patient outcomes.

Business Context: A hospital management team has tasked us with creating a predictive model to identify patients at risk of CKD. Early detection of CKD can significantly enhance treatment efficacy and patient care, while also potentially reducing healthcare costs. By leveraging historical patient data, we aim to build a model that can classify new patients as either having CKD or not, based on their medical parameters.

Data Description: The dataset contains various features that are used to predict the presence of CKD. These features might include:

- Age
- Blood Pressure
- Specific Gravity
- Albumin
- Sugar
- Red Blood Cells
- Pus Cell
- Serum Creatinine
- Hemoglobin
- Packed Cell Volume
- White Blood Cell Count
- Red Blood Cell Count
- Hypertension
- Diabetes Mellitus
- Coronary Artery Disease
- Appetite
- Pedal Edema
- Anemia

Approach:

1. Data Pre-processing:

   1.1 Handle missing values.

   1.2 Encode categorical variables (using the get_dummies method).

   1.3 Standardize numerical features.

2. Model Selection:

2.1 Split the data into training and testing sets.

2.2 Train various classification algorithms, such as:

- Logistic Regression
- Decision Trees
- Random Forest
- Support Vector Machine (SVM)
- k-Nearest Neighbors (k-NN)
- Naive Bayes
- Gradient Boosting

2.3 Hyperparameter Tuning:

2.3.1 Utilize techniques like GridSearchCV to tune the hyperparameters of each model.

2.3.2 Fine-tune hyperparameters to optimize model performance.

3. Model Evaluation:

3.1 Evaluate models using metrics such as accuracy, precision, recall, F1-score, and ROC-AUC.

3.2 Perform cross-validation to ensure model robustness. 3.3 Select the best-performing model based on evaluation metrics.

Best Model - Random Forest:

- Utilized hyperparameter tuning using GridSearchCV.
- Best parameters used were: RandomForestClassifier(criterion='entropy', max_depth=10, min_samples_split=5, n_estimators=50)

Model Evaluation Results:

- **Random Forest Classification Report and AUC & ROC Score:**
  - The classification report shows high precision, recall, and F1-score for both classes (0 and 1), indicating good performance.
  - The ROC AUC score is very high (close to 1.0), indicating excellent performance in distinguishing between positive and negative classes.

```
#Classification report
classification_rep = classification_report(y_test, test_predictions)
print("Classification Report:")
print(classification_rep)
```

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

           0       0.98      0.98      0.98        45
           1       0.99      0.99      0.99        75

    accuracy                           0.98       120
   macro avg       0.98      0.98      0.98       120
weighted avg       0.98      0.98      0.98       120
```

```
# ROC AUC score
test_probabilities = best_random_forest.predict_proba(X_test)[:, 1]
roc_auc = roc_auc_score(y_test, test_probabilities)
print(f"ROC AUC Score: {roc_auc}")
```

```
ROC AUC Score: 0.9997037037037038
```

- **Gaussian Naive Bayes Classification Report and AUC & ROC Score:**
  - The classification report also shows high precision, recall, and F1-score for both classes.
  - The ROC AUC score is perfect (1.0), indicating perfect performance in distinguishing between positive and negative classes.

```
[[45  0]
 [ 2 73]]
              precision    recall  f1-score   support

           0       0.96      1.00      0.98        45
           1       1.00      0.97      0.99        75

    accuracy                           0.98       120
   macro avg       0.98      0.99      0.98       120
weighted avg       0.98      0.98      0.98       120

ROC AUC Score:: 1.0
```

- **Decision Tree Classification Report and AUC & ROC Score:**
  - The classification report shows decent performance with lower precision, recall, and F1-score compared to Random Forest and Naive Bayes.
  - The ROC AUC score is good but slightly lower than the other models.

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

           0       0.90      0.96      0.92        45
           1       0.97      0.93      0.95        75

    accuracy                           0.94       120
   macro avg       0.93      0.94      0.94       120
weighted avg       0.94      0.94      0.94       120
```

```python
# ROC AUC score
test_probabilities = best_decision_tree.predict_proba(X_test)[:, 1]
roc_auc = roc_auc_score(y_test, test_probabilities)
print(f"ROC AUC Score: {roc_auc}")
```

```
ROC AUC Score: 0.9444444444444445
```

- **SVM Classification Report and AUC & ROC Score:**
  - The classification report and ROC AUC score indicate excellent performance, like Random Forest and Naive Bayes.

```python
#Classification report
classification_rep = classification_report(y_test, test_predictions)
print("Classification Report:")
print(classification_rep)
```

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

           0       0.96      1.00      0.98        45
           1       1.00      0.97      0.99        75

    accuracy                           0.98       120
   macro avg       0.98      0.99      0.98       120
weighted avg       0.98      0.98      0.98       120
```

```python
test_scores = best_svm_classifier.decision_function(X_test)
```

```python
# Calculating ROC AUC score
roc_auc = roc_auc_score(y_test, test_scores)
print(f"ROC AUC Score: {roc_auc}")
```

```
ROC AUC Score: 1.0
```

- **KNN Classification Report and AUC & ROC Score:**
  - The classification report shows good performance, but the ROC AUC score is slightly lower compared to Random Forest and Naive Bayes.

```
#Classification report
classification_rep = classification_report(y_test, test_predictions)
print("Classification Report:")
print(classification_rep)
```

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

           0       0.88      1.00      0.94        45
           1       1.00      0.92      0.96        75

    accuracy                           0.95       120
   macro avg       0.94      0.96      0.95       120
weighted avg       0.96      0.95      0.95       120
```

```
# ROC AUC score
test_probabilities = best_knn_neighbour.predict_proba(X_test)[:, 1]
roc_auc = roc_auc_score(y_test, test_probabilities)
print(f"ROC AUC Score: {roc_auc}")
```

```
ROC AUC Score: 0.9995555555555555
```

- **Logistic Regression Classification Report and AUC & ROC Score:**
  - The classification report and ROC AUC score indicate excellent performance, like Random Forest and Naive Bayes.

```
#Classification report
classification_rep = classification_report(y_test, test_predictions)
print("Classification Report:")
print(classification_rep)
```

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

           0       0.98      1.00      0.99        45
           1       1.00      0.99      0.99        75

    accuracy                           0.99       120
   macro avg       0.99      0.99      0.99       120
weighted avg       0.99      0.99      0.99       120
```

```
# ROC AUC score
test_probabilities = best_logistic_regression.predict_proba(X_test)[:, 1]
roc_auc = roc_auc_score(y_test, test_probabilities)
print(f"ROC AUC Score: {roc_auc}")
```

```
ROC AUC Score: 1.0
```

Conclusion:

In conclusion, after evaluating multiple classification models including Random Forest, Gaussian Naive Bayes, Support Vector Machine (SVM), KNN, and Logistic Regression, it is evident that Random Forest and Logistic Regression stand out as strong candidates for model deployment.

Random Forest demonstrates superior performance in terms of predictive accuracy, particularly in handling complex data with non-linear relationships. Its ability to effectively handle outliers and noise, along with providing valuable feature importance scores for interpretability, makes it a preferred choice.

Therefore, based on its robustness, predictive performance, and interpretability, **Random Forest** emerges as the recommended model for deployment in this scenario.