Problem Statement or Requirement:

A requirement from the Hospital, Management asked us to create a predictive model which will predict the Chronic Kidney Disease (CKD) based on the several parameters. The Client has provided the dataset of the same

CKD Prediction – Model Development Report

1) Problem statement

Build a supervised ML model that predicts **Chronic Kidney Disease (CKD)** (yes/no) from routine clinical and lab parameters to enable early triage and follow-up.

2) Dataset overview

- **Rows × columns: 399 × 25**
- **Target:** classification (yes/no)
- **Class distribution:** yes=249 (62.4%), no=150 (37.6%) → **mild imbalance**

```
dataset=pd.read_csv("CKD.csv")
dataset['classification'].value_counts()

classification
yes    249
no     150
Name: count, dtype: int64
```

[4]:

	age	bp	al	su	bgr	bu	sc	sod	pot	hrmo	...	pc_normal	pcc_present	ba_present	ht
0	2.000000	76.459948	3.0	0.0	148.112676	57.482105	3.077356	137.528754	4.627244	12.518156	...	0	0	0	
1	3.000000	76.459948	2.0	0.0	148.112676	22.000000	0.700000	137.528754	4.627244	10.700000	...	1	0	0	
2	4.000000	76.459948	1.0	0.0	99.000000	23.000000	0.600000	138.000000	4.400000	12.000000	...	1	0	0	
3	5.000000	76.459948	1.0	0.0	148.112676	16.000000	0.700000	138.000000	3.200000	8.100000	...	1	0	0	
4	5.000000	50.000000	0.0	0.0	148.112676	25.000000	0.600000	137.528754	4.627244	11.800000	...	1	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
394	51.492308	70.000000	0.0	0.0	219.000000	36.000000	1.300000	139.000000	3.700000	12.500000	...	1	0	0	
395	51.492308	70.000000	0.0	2.0	220.000000	68.000000	2.800000	137.528754	4.627244	8.700000	...	1	0	0	
396	51.492308	70.000000	3.0	0.0	110.000000	115.000000	6.000000	134.000000	2.700000	9.100000	...	1	0	0	
397	51.492308	90.000000	0.0	0.0	207.000000	80.000000	6.800000	142.000000	5.500000	8.500000	...	1	0	0	
398	51.492308	80.000000	0.0	0.0	100.000000	49.000000	1.000000	140.000000	5.000000	16.300000	...	1	0	0	

399 rows × 28 columns

3) Data preprocessing

- **Missing tokens → NaN:** replaced ? / NA / na / NaN / nan / "" with NaN.

- **Imputation:**
  - **Numeric:** median (fit on train, applied to test)
  - **Categorical:** most-frequent value (fit on train, applied to test)
- **Encoding:** one-hot encoding (train on train set; **test columns aligned** to train to handle unseen categories).
- **Scaling:**
  - Applied **StandardScaler** for models that benefit from scaling (**Logistic Regression, kNN, GaussianNB, SVM**).
  - **Not** applied for tree/boosting models (**Random Forest, Decision Tree**).
- **Evaluation protocol: Stratified** train/test split (test size ≈ 1/3 → 133 records). Model selection via **5-fold stratified CV**.
  Primary metric: **ROC AUC**; also report Accuracy, Precision, Recall, F1, PR-AUC on the hold-out test set.

4) Experiments & results

Models evaluated

- SVM (RBF/Linear grid)
- Logistic Regression
- Random Forest
- Decision Tree
- k-Nearest Neighbors (kNN)
- Gaussian Naive Bayes
- Xgboost

CKD Prediction – Model Development Report

Model	Best CV R	Best Params (key)	Test Accu	Precision	Recall (pos)	F1 (pos)	ROC-AUC	Test PR-AUC
SVM (grid)	—	kernel={rbf,linear}, C∈{0.5,1,3,10}, γ∈{scale,0.1,0.01}	0.98	1	0.98	0.99	0.988	—
Logistic Regression	1	C=0.1, solver=lbfgs, penalty=l2	0.970	1.000	0.952	0.975	1.000	1.000
Random Forest	1	n_estimators=200, max_depth=None, max_features=√, min_samples_split=2, min_samples_leaf=1	0.985	0.988	0.988	0.988	1.000	1.000
Decision Tree	0.971	criterion=gini, max_depth=None, min_samples_split=10, min_samples_leaf=2	0.970	0.976	0.976	0.976	0.976	0.977
kNN	1	n_neighbors=11, p=1, weights=uniform	0.932	1.000	0.892	0.943	0.993	0.995
GaussianNB	1	var_smoothing=1e-12	0.977	0.988	0.976	0.982	0.990	0.988
Xgboost		colsample_bytree: 0.8, learning_rate: 0.05,	0.925	0.958	0.920	0.939	0.992	0.995

SVM(RBF)

```
Fitting 5 folds for each of 24 candidates, totalling 120 fits
[[49  1]
 [ 1 82]]
              precision    recall  f1-score   support

           0       0.98      0.98      0.98        50
           1       0.99      0.99      0.99        83

    accuracy                           0.98       133
   macro avg       0.98      0.98      0.98       133
weighted avg       0.98      0.98      0.98       133

Accuracy : 0.9849624060150376
Precision: 0.9879518072289156
Recall   : 0.9879518072289156
F1       : 0.9879518072289156
ROC AUC  : 0.9990361445783132

Best params: {'C': 0.5, 'gamma': 'scale', 'kernel': 'rbf'}
```

- Logistic Regression

```
Fitting 5 folds for each of 26 candidates, totalling 130 fits

===== LogisticRegression =====
Best CV score ( roc_auc ): 1.0
Best params: {'C': np.float64(0.1), 'penalty': 'l2', 'solver': 'lbfgs'}

Confusion matrix:
 [[50  0]
 [ 4 79]]

Classification report:
               precision    recall  f1-score   support

           0       0.93      1.00      0.96        50
           1       1.00      0.95      0.98        83

    accuracy                           0.97       133
   macro avg       0.96      0.98      0.97       133
weighted avg       0.97      0.97      0.97       133

Accuracy : 0.9699248120300752
Precision: 1.0
Recall   : 0.9518072289156626
F1       : 0.9753086419753086
ROC AUC  : 0.9995180722891566
PR  AUC  : 0.9997165131112689
Fitting 5 folds for each of 324 candidates, totalling 1620 fits
```

- Random Forest

```
Fitting 5 folds for each of 324 candidates, totalling 1620 fits

===== RandomForest =====
Best CV score ( roc_auc ): 1.0
Best params: {'max_depth': None, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}

Confusion matrix:
 [[49  1]
 [ 1 82]]

Classification report:
               precision    recall  f1-score   support

           0       0.98      0.98      0.98        50
           1       0.99      0.99      0.99        83

    accuracy                           0.98       133
   macro avg       0.98      0.98      0.98       133
weighted avg       0.98      0.98      0.98       133

Accuracy : 0.9849624060150376
Precision: 0.9879518072289156
Recall   : 0.9879518072289156
F1       : 0.9879518072289156
ROC AUC  : 0.9995180722891566
PR  AUC  : 0.9997114101846285
```

- Decision Tree

```
Fitting 5 folds for each of 162 candidates, totalling 810 fits

===== DecisionTree =====
Best CV score ( roc_auc ): 0.9711051693404634
Best params: {'criterion': 'gini', 'max_depth': None, 'min_samples_leaf': 2, 'min_samples_split': 10}

Confusion matrix:
 [[48  2]
 [ 2 81]]

Classification report:
               precision    recall  f1-score   support

           0       0.96      0.96      0.96        50
           1       0.98      0.98      0.98        83

    accuracy                           0.97       133
   macro avg       0.97      0.97      0.97       133
weighted avg       0.97      0.97      0.97       133

Accuracy : 0.9699248120300752
Precision: 0.9759036144578314
Recall   : 0.9759036144578314
F1       : 0.9759036144578314
ROC AUC  : 0.9756626506024098
PR  AUC  : 0.9771266510446963
```

- k-Nearest Neighbors (kNN)

```
Fitting 5 folds for each of 60 candidates, totalling 300 fits

===== kNN =====
Best CV score ( roc_auc ): 1.0
Best params: {'n_neighbors': 11, 'p': 1, 'weights': 'uniform'}

Confusion matrix:
 [[50  0]
 [ 9 74]]

Classification report:
              precision    recall  f1-score   support

           0       0.85      1.00      0.92        50
           1       1.00      0.89      0.94        83

    accuracy                           0.93       133
   macro avg       0.92      0.95      0.93       133
weighted avg       0.94      0.93      0.93       133

Accuracy : 0.9323308270676691
Precision: 1.0
Recall   : 0.891566265060241
F1       : 0.9426751592356688
ROC AUC  : 0.993012048192771
PR  AUC  : 0.9948934245906536
```

- Gaussian Naive Bayes

```
Fitting 5 folds for each of 7 candidates, totalling 35 fits

===== GaussianNB =====
Best CV score ( roc_auc ): 1.0
Best params: {'var_smoothing': np.float64(1e-12)}

Confusion matrix:
 [[49  1]
 [ 2 81]]

Classification report:
              precision    recall  f1-score   support

           0       0.96      0.98      0.97        50
           1       0.99      0.98      0.98        83

    accuracy                           0.98       133
   macro avg       0.97      0.98      0.98       133
weighted avg       0.98      0.98      0.98       133

Accuracy : 0.9774436090225563
Precision: 0.9878048780487805
Recall   : 0.9759036144578314
F1       : 0.9818181818181818
ROC AUC  : 0.9897590361445783
PR  AUC  : 0.9878101465936793
```

- Xgboost

```
Fitting 5 folds for each of 32 candidates, totalling 160 fits

===== XGBoost (gbtree, small grid) =====
Best CV score ( roc_auc ): 0.9987394957983193
Best params: {'colsample_bytree': 0.8, 'learning_rate': 0.05, 'max_depth': 3, 'min_child_weight': 1, 'subsample': 0.8}

Confusion matrix:
 [[14  1]
 [ 2 23]]

Classification report:
              precision    recall  f1-score   support

           0       0.88      0.93      0.90        15
           1       0.96      0.92      0.94        25

    accuracy                           0.93        40
   macro avg       0.92      0.93      0.92        40
weighted avg       0.93      0.93      0.93        40

Accuracy : 0.925
Precision: 0.9583333333333334
Recall   : 0.92
F1       : 0.9387755102040817
ROC AUC  : 0.992
PR  AUC  : 0.995194871794872
```

6) Final model & justification

Selected model: Random Forest Classifier

Best params (from grid): n_estimators=200, max_depth=None, max_features='sqrt', min_samples_split=2, min_samples_leaf=1 (with class_weight='balanced')

Test performance (hold-out):

Accuracy: 0.985
Precision (pos): 0.988
Recall (pos): 0.988
F1 (pos): 0.988
ROC-AUC / PR-AUC: ~0.9995 / 0.9997
Confusion matrix: [[49, 1], [1, 82]] → only 1 false negative, which is critical for CKD screening.
Why this model:
Best overall balance of Precision, Recall, and F1 on the test set, with near-perfect AUCs.
Low clinical risk: extremely few false negatives (missed CKD), while keeping false positives low.
Stable & robust: 5-fold CV ROC-AUC ≈ 1.0, minimal variance across folds.
Why not the others:
SVM: strong (Acc 0.98, Rec 0.98) but more FN than RF (2 vs 1), and needs feature scaling + more tuning.
Logistic Regression: excellent AUC but lower recall (0.952) vs RF, risking more missed CKD cases.
Decision Tree: interpretable but weaker overall metrics; can overfit as a single tree.
kNN: lower recall (0.892) and accuracy on this dataset.
GaussianNB: very good, but slightly behind RF on recall/F1.
XGBoost (quick config tested): high AUC, but the tested setup underperformed RF on accuracy/recall; RF gave the best balance here.

- Balanced errors (confusion matrix [[49, 1], [1, 82]])—very low false negatives, which is crucial for CKD screening.