Chronic Kidney Disease (CKD) Predictive Model

SUMMARY

Objective: Build a reliable model to predict CKD presence from routine clinical parameters.

Best Model: LogisticRegression with a preprocessing pipeline (imputation, scaling, one-hot encoding).

PROBLEM STATEMENT & DATASET:

Problem: Predict CKD (Yes/No) to aid early detection and resource allocation.

Dataset: 399 rows, 28 columns, target = classification.

Numeric features	Categorical features
16	**11**

Cleaning Dataset:

1. Yes/No Variants.
2. **Preprocessing Pipeline:** numeric and standardization; categorical → one-hot encoding.
3. **Modeling:** trained Logistic Regression, SVC (RBF), and Random Forest on stratified train/test split.
4. **Evaluation:** computed Accuracy, Precision, Recall, F1, ROC AUC on held-out test set; inspected confusion matrix and ROC curve.
5. **Selection:** chose the simplest model with Best Performance and best clinical interpretability.

RESULTS:

Model Comparison

model	accuracy	precision	recall	f1	ROC_AUC
LogisticRegression	0.99	1.0	0.98	0.9899	1.0000
SVC	0.99	1.0	0.98	0.9899	1.0000
RandomForest	0.9875	1.0	0.98	0.9899	0.9993

CONFUSION MATRIX:

	Pred 0	Pred 1
Actual 0	51	0
Actual 1	1	81

FINAL REPORT [Logistic Regression]:

```
print("The f1_macro value for best parameter {}:".format(grid.best_params_), f1_macro)
```

```
The f1_macro value for best parameter {'penalty': 'l2', 'solver': 'newton-cg'}: 0.9924946382275899
```

```
[24]: print("The Confusion Matrix:\n", cm)
```

```
The Confusion Matrix:
 [[51  0]
 [ 1 81]]
```

```
[25]: print("The Report:\n", clf_report)
```

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

       False       0.98      1.00      0.99        51
        True       1.00      0.99      0.99        82

    accuracy                           0.99       133
   macro avg       0.99      0.99      0.99       133
weighted avg       0.99      0.99      0.99       133
```

Grid Table [Logistic Regression]:

```
[27]: table = pd.DataFrame.from_dict(re)

[28]: table
```

[28]:

enalty	param_solver	params	split0_test_score	split1_test_score	split2_test_score	split3_test_score	split4_test_score	mean_test_score	std_test_score	rank_test_score
l2	newton-cg	{'penalty': 'l2', 'solver': 'newton-cg'}	0.981569	0.981014	0.981217	1.000000	1.000000	0.988760	0.009179	1
l2	lbfgs	{'penalty': 'l2', 'solver': 'lbfgs'}	0.981569	0.981014	0.981217	1.000000	1.000000	0.988760	0.009179	1
l2	liblinear	{'penalty': 'l2', 'solver': 'liblinear'}	0.963284	0.981233	0.962573	0.981217	0.981217	0.973905	0.008965	4
l2	saga	{'penalty': 'l2', 'solver': 'saga'}	0.981569	0.981233	0.981217	0.981217	0.981217	0.981291	0.000139	3

FUTURE PREDICTIONS: Future_Prediction = [1]

```
print(input_data)
```

```
    age    bp  al  su    bgr   bu   sc    sod  pot  hrmo  ...  rbc_normal  \
0  45.0  80.0   2   0  150.0  35.0  1.2  135.0  4.5  12.0  ...           0

   pc_normal  pcc_present  ba_present  htn_yes  dm_yes  cad_yes  appet_yes  \
0          0            0           1        0       0        1          0

   pe_yes  ane_yes
0       1        0

[1 rows x 27 columns]
```

```
# Predict with the trained model
Future_Prediction = grid.predict(input_data)
print("Future_Prediction = {}".format(Future_Prediction))
```

```
Future_Prediction = [ True]
```

Why Choosing Logistic Regression:

We chose **Logistic Regression** because the **CKD prediction is a binary classification** problem. Logistic Regression is interpretable, **provides probabilistic outputs, handles both numerical and categorical data efficiently**, and works well with relatively small medical datasets. Most importantly, it allows clinicians to understand the impact of each medical parameter on the disease outcome, which is essential in healthcare applications.

Unlike complex models (RandomForest, Neural Networks), logistic regression is **less prone to overfitting** on small datasets.

SAVED MODEL:

```
[39]: #Save our model
import pickle
# Suppose you trained your model as 'grid' using GridSearchCV
grid.fit(X_train, y_train)
filename="finalized_model_CKD_LogisticRegression_classification.sav" # file name has been created and saved here the trained model.

[41]: # Here we have to insert the already created model.
pickle.dump(grid,open(filename,'wb')) # Here the 'wb' is "Write Binary".

# With this we have saved the model.

[42]: # Next step is to load the saved model.
loaded_model=pickle.load(open("finalized_model_CKD_LogisticRegression_classification.sav",'rb'))
#Here we have to check with the user input in real time.
# Example prediction with input_data
result = loaded_model.predict(input_data)

C:\Anaconda3\Lib\site-packages\sklearn\utils\validation.py:2732: UserWarning: X has feature names, but LogisticRegression was fitted without feature n
ames
  warnings.warn(

[43]: result

[43]: array([ True])
```

DEPLOYMENT & HANDOVER:

- **Model:** finalized_model_CKD_LogisticRegression_classification.sav

Algorithms:

- **Metrics:** model comparison, test report, confusion matrix
- **Figure:** ROC curve