Machine Learning – Classification (Assignment)

Problem Statement or Requirement:

The client wants to develop a machine learning-based application that can accurately **predict the presence of Chronic Kidney Disease (CKD)** in patients based on multiple clinical and demographic parameters. Early detection of CKD is critical to enabling timely medical intervention and improving patient outcomes.

1. Problem Statement
Domain Selection - Machine Learning
Learning Selection - Supervised Learning
It is a Classification

2. Basic information of the Dataset
Rows = 399
Columns = 25

3. Preprocessing
This dataset contains **11 categorical columns** which is Nominal so I used **One-Hot-Encoder** to convert Categorical columns to Numerical Columns
We have also standardised the dataset using **StandardScaler** function to ensure there is **defined range** between data

4. Working out with different Algorithms to find the best model

Support Vector Machine

Hyper tuning Parameters grid:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	param_gamma	param_kernel	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
0	0.018402	0.009238	0.019347	0.00577	10	auto	linear	{'C': 10, 'gamma': 'auto', 'kernel': 'linear'}	0.984445	0.984436	0.984436	0.96875	0.984195	0.981252	0.006252	23
1	0.025279	0.010769	0.017068	0.003894	10	auto	rbf	{'C': 10, 'gamma': 'auto', 'kernel': 'rbf'}	0.96901	1	0.984305	0.984305	1	0.987524	0.011617	9
2	0.057689	0.003059	0.01468	0.002015	10	auto	poly	{'C': 10, 'gamma': 'auto', 'kernel': 'poly'}	1	0.984305	0.96875	0.984305	1	0.987472	0.0117	11
3	0.035888	0.010634	0.014354	0.002494	10	auto	sigmoid	{'C': 10, 'gamma': 'auto', 'kernel': 'sigmoid'}	0.984445	1	0.984305	0.96875	0.968508	0.981202	0.011745	33
4	0.020226	0.002409	0.01296	0.001076	10	scale	linear	{'C': 10, 'gamma': 'scale', 'kernel': 'linear'}	0.984445	0.984436	0.984436	0.96875	0.984195	0.981252	0.006252	23
5	0.020775	0.005514	0.014392	0.003578	10	scale	rbf	{'C': 10, 'gamma': 'scale', 'kernel': 'rbf'}	0.96901	1	0.984305	0.984305	1	0.987524	0.011617	9
6	0.022446	0.00298	0.013446	0.001496	10	scale	poly	{'C': 10, 'gamma': 'scale', 'kernel': 'poly'}	1	0.984305	0.96875	0.984305	1	0.987472	0.0117	11
7	0.014683	0.007503	0.012745	0.001339	10	scale	sigmoid	{'C': 10, 'gamma': 'scale', 'kernel': 'sigmoid'}	0.96901	1	0.984305	0.953307	0.968508	0.975026	0.015876	40
8	0.016037	0.003813	0.012252	0.001482	100	auto	linear	{'C': 100, 'gamma': 'auto', 'kernel': 'linear'}	0.984445	0.984436	0.984436	0.96875	0.984195	0.981252	0.006252	23
9	0.014451	0.003811	0.011855	0.001183	100	auto	rbf	{'C': 100, 'gamma': 'auto', 'kernel': 'rbf'}	0.984445	1	0.984305	0.984305	1	0.990611	0.007666	1
10	0.02201	0.002584	0.011118	0.001158	100	auto	poly	{'C': 100, 'gamma': 'auto', 'kernel': 'poly'}	0.984445	1	0.984305	0.984305	0.984195	0.98745	0.006275	13
11	0.014559	0.00456	0.011994	0.000427	100	auto	sigmoid	{'C': 100, 'gamma': 'auto', 'kernel': 'sigmoid'}	1	0.984436	0.984436	0.96875	0.984195	0.984363	0.009883	15
12	0.014761	0.003989	0.011073	0.000947	100	scale	linear	{'C': 100, 'gamma': 'scale', 'kernel': 'linear'}	0.984445	0.984436	0.984436	0.96875	0.984195	0.981252	0.006252	23
13	0.011453	0.001868	0.012184	0.001283	100	scale	rbf	{'C': 100, 'gamma': 'scale', 'kernel': 'rbf'}	0.984445	1	0.984305	0.984305	1	0.990611	0.007666	1
14	0.020295	0.001937	0.014346	0.001217	100	scale	poly	{'C': 100, 'gamma': 'scale', 'kernel': 'poly'}	0.984445	1	0.984305	0.984305	0.984195	0.98745	0.006275	13
15	0.01291	0.002514	0.01272	0.00132	100	scale	sigmoid	{'C': 100, 'gamma': 'scale', 'kernel': 'sigmoid'}	1	0.984436	0.984436	0.96875	0.984195	0.984363	0.009883	15
16	0.013433	0.004872	0.013129	0.001025	1000	auto	linear	{'C': 1000, 'gamma': 'auto', 'kernel': 'linear'}	0.984445	0.984436	0.984436	0.96875	0.984195	0.981252	0.006252	23
17	0.013355	0.002317	0.013419	0.000791	1000	auto	rbf	{'C': 1000, 'gamma': 'auto', 'kernel': 'rbf'}	0.984445	1	0.984305	0.984305	1	0.990611	0.007666	1
18	0.025254	0.003433	0.012653	0.000756	1000	auto	poly	{'C': 1000, 'gamma': 'auto', 'kernel': 'poly'}	0.95367	1	0.984305	0.984436	0.984195	0.981321	0.015102	17
19	0.015098	0.00537	0.011679	0.001144	1000	auto	sigmoid	{'C': 1000, 'gamma': 'auto', 'kernel': 'sigmoid'}	0.984445	0.984436	0.984436	0.96875	0.968508	0.978115	0.007745	34
20	0.012184	0.003327	0.011391	0.000546	1000	scale	linear	{'C': 1000, 'gamma': 'scale', 'kernel': 'linear'}	0.984445	0.984436	0.984436	0.96875	0.984195	0.981252	0.006252	23
21	0.011371	0.001488	0.01199	0.000796	1000	scale	rbf	{'C': 1000, 'gamma': 'scale', 'kernel': 'rbf'}	0.984445	1	0.984305	0.984305	1	0.990611	0.007666	1
22	0.019491	0.002713	0.0121	0.000796	1000	scale	poly	{'C': 1000, 'gamma': 'scale', 'kernel': 'poly'}	0.95367	1	0.984305	0.984436	0.984195	0.981321	0.015102	17
23	0.010982	0.003055	0.010868	0.00039	1000	scale	sigmoid	{'C': 1000, 'gamma': 'scale', 'kernel': 'sigmo...	0.984445	0.984436	0.984436	0.96875	0.968508	0.978115	0.007745	34
24	0.010862	0.002234	0.010965	0.000689	2000	auto	linear	{'C': 2000, 'gamma': 'auto', 'kernel': 'linear'}	0.984445	0.984436	0.984436	0.96875	0.984195	0.981252	0.006252	23
25	0.011962	0.004353	0.011827	0.001039	2000	auto	rbf	{'C': 2000, 'gamma': 'auto', 'kernel': 'rbf'}	0.984445	1	0.984305	0.984305	1	0.990611	0.007666	1
26	0.018879	0.001119	0.012498	0.001083	2000	auto	poly	{'C': 2000, 'gamma': 'auto', 'kernel': 'poly'}	0.95367	1	0.984305	0.984436	0.984195	0.981321	0.015102	17
27	0.010446	0.001262	0.011782	0.000902	2000	auto	sigmoid	{'C': 2000, 'gamma': 'auto', 'kernel': 'sigmoid'}	0.984445	0.984436	0.984436	0.96875	0.968508	0.978115	0.007745	34
28	0.010135	0.000757	0.011908	0.001148	2000	scale	linear	{'C': 2000, 'gamma': 'scale', 'kernel': 'linear'}	0.984445	0.984436	0.984436	0.96875	0.984195	0.981252	0.006252	23
29	0.011051	0.000822	0.01201	0.000425	2000	scale	rbf	{'C': 2000, 'gamma': 'scale', 'kernel': 'rbf'}	0.984445	1	0.984305	0.984305	1	0.990611	0.007666	1
30	0.018907	0.000229	0.01315	0.000598	2000	scale	poly	{'C': 2000, 'gamma': 'scale', 'kernel': 'poly'}	0.95367	1	0.984305	0.984436	0.984195	0.981321	0.015102	17
31	0.01151	0.002163	0.012616	0.000813	2000	scale	sigmoid	{'C': 2000, 'gamma': 'scale', 'kernel': 'sigmo...	0.984445	0.984436	0.984436	0.96875	0.968508	0.978115	0.007745	34
32	0.01047	0.00164	0.011488	0.000938	3000	auto	linear	{'C': 3000, 'gamma': 'auto', 'kernel': 'linear'}	0.984445	0.984436	0.984436	0.96875	0.984195	0.981252	0.006252	23
33	0.010629	0.000204	0.012834	0.000878	3000	auto	rbf	{'C': 3000, 'gamma': 'auto', 'kernel': 'rbf'}	0.984445	1	0.984305	0.984305	1	0.990611	0.007666	1
34	0.019506	0.000832	0.01207	0.00111	3000	auto	poly	{'C': 3000, 'gamma': 'auto', 'kernel': 'poly'}	0.95367	1	0.984305	0.984436	0.984195	0.981321	0.015102	17
35	0.010803	0.000742	0.012751	0.001019	3000	auto	sigmoid	{'C': 3000, 'gamma': 'auto', 'kernel': 'sigmoid'}	0.984445	0.984436	0.984436	0.96875	0.968508	0.978115	0.007745	34
36	0.009097	0.001311	0.012787	0.000776	3000	scale	linear	{'C': 3000, 'gamma': 'scale', 'kernel': 'linear'}	0.984445	0.984436	0.984436	0.96875	0.984195	0.981252	0.006252	23
37	0.010238	0.002137	0.011927	0.000491	3000	scale	rbf	{'C': 3000, 'gamma': 'scale',	0.984445	1	0.984305	0.984305	1	0.990611	0.007666	1
38	0.018455	0.00131	0.009421	0.000915	3000	scale	poly	{'C': 3000, 'gamma': 'scale', 'kernel': 'poly'}	0.95367	1	0.984305	0.984436	0.984195	0.981321	0.015102	17
39	0.009926	0.000801	0.010727	0.000398	3000	scale	sigmoid	{'C': 3000, 'gamma': 'scale', 'kernel': 'sigmo...	0.984445	0.984436	0.984436	0.96875	0.968508	0.978115	0.007745	34

Evaluation Metrics Report:

```
              precision    recall  f1-score   support

           0       0.97      1.00      0.98        32
           1       1.00      0.98      0.99        48

    accuracy                           0.99        80
   macro avg       0.98      0.99      0.99        80
weighted avg       0.99      0.99      0.99        80
```

ROC_AUC Score:

Decision Tree

Hyper tuning Parameters grid:

```
from sklearn.metrics import roc_auc_score

roc_auc_score(y_test,grid.predict_proba(x_test)[:,1])
```

```
0.9993489583333334
```

grid:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_criterion	param_splitter	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
0	0.003235	0.00086	0.009713	0.001125	gini	random	{'criterion': 'gini', 'splitter': 'random'}	0.96875	0.984305	0.907658	0.953307	0.984195	0.959643	0.028413	2
1	0.004018	0.001713	0.010908	0.001261	gini	best	{'criterion': 'gini', 'splitter': 'best'}	0.95367	0.890137	0.922956	0.95241	0.968254	0.937485	0.027873	5
2	0.004414	0.000813	0.008617	0.000626	entropy	random	{'criterion': 'entropy', 'splitter': 'random'}	0.95367	0.968452	0.984436	0.96875	0.968508	0.968763	0.009732	1
3	0.004293	0.00047	0.0106	0.000448	entropy	best	{'criterion': 'entropy', 'splitter': 'best'}	0.96901	0.891049	0.937949	0.920683	0.968254	0.937389	0.029595	6
4	0.003563	0.001407	0.010228	0.002093	log_loss	random	{'criterion': 'log_loss', 'splitter': 'random'}	0.96901	0.953307	0.922956	0.9375	0.952586	0.947072	0.015645	3
5	0.00371	0.000608	0.008687	0.001288	log_loss	best	{'criterion': 'log_loss', 'splitter': 'best'}	0.96901	0.90625	0.953591	0.936147	0.968254	0.94665	0.023483	4

Evaluation Metrics Report:

```
              precision    recall  f1-score   support

           0       1.00      0.97      0.98        32
           1       0.98      1.00      0.99        48

    accuracy                           0.99        80
   macro avg       0.99      0.98      0.99        80
weighted avg       0.99      0.99      0.99        80
```

ROC_AUC Score:

```
from sklearn.metrics import roc_auc_score

roc_auc_score(y_test,grid.predict_proba(x_test)[:,1])
```

```
0.984375
```

Random Forest

Hyper tuning Parameters grid:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_criterion	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
0	0.236706	0.009368	0.018948	0.001431	gini	{'criterion': 'gini'}	0.984445	0.952916	0.984436	0.984305	1	0.98122	0.015389	2
1	0.242593	0.009106	0.018219	0.001255	entropy	{'criterion': 'entropy'}	0.984445	0.952916	0.984436	0.984305	0.984049	0.97803	0.012558	3
2	0.225827	0.020249	0.019445	0.002277	log_loss	{'criterion': 'log_loss'}	0.984445	0.984305	0.968974	0.96875	1	0.981295	0.011645	1

Evaluation Metrics Report:

```
              precision    recall  f1-score   support

           0       1.00      0.97      0.98        32
           1       0.98      1.00      0.99        48

    accuracy                           0.99        80
   macro avg       0.99      0.98      0.99        80
weighted avg       0.99      0.99      0.99        80
```

ROC_AUC Score:

```
from sklearn.metrics import roc_auc_score

roc_auc_score(y_test,grid.predict_proba(x_test)[:,1])
```

```
1.0
```

Logistic Regression

Hyper tuning Parameters grid:

mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_multi_class	param_penalty	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
0.001306	0.000401	0	0	auto	l1	lbfgs	{'multi_class': 'auto', 'penalty': 'l1', 'solv...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	22
0.008803	0.001386	0.00937	0.002663	auto	l1	liblinear	{'multi_class': 'auto', 'penalty': 'l1', 'solv...	0.95367	1	0.968974	0.96875	0.968508	0.971981	0.015178	17
0.001702	0.0004	0	0	auto	l1	newton-cg	{'multi_class': 'auto', 'penalty': 'l1', 'solv...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	22
0.001302	0.000597	0	0	auto	l1	newton-cholesky	{'multi_class': 'auto', 'penalty': 'l1', 'solv...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	22
0.000702	0.000399	0	0	auto	l1	sag	{'multi_class': 'auto', 'penalty': 'l1', 'solv...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	22
0.000603	0.000492	0	0	multinomial	None	liblinear	{'multi_class': 'multinomial', 'penalty': 'Non...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	22
0.0004	0.000489	0	0	multinomial	None	newton-cg	{'multi_class': 'multinomial', 'penalty': 'Non...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	22
0.000504	0.000451	0	0	multinomial	None	newton-cholesky	{'multi_class': 'multinomial', 'penalty': 'Non...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	22
0.000826	0.000415	0	0	multinomial	None	sag	{'multi_class': 'multinomial', 'penalty': 'Non...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	22
0.000202	0.000404	0	0	multinomial	None	saga	{'multi_class': 'multinomial', 'penalty': 'Non...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	22

Evaluation Metrics Report:

```
              precision    recall  f1-score   support

           0       0.97      1.00      0.98        32
           1       1.00      0.98      0.99        48

    accuracy                           0.99        80
   macro avg       0.98      0.99      0.99        80
weighted avg       0.99      0.99      0.99        80
```

ROC_AUC Score:

```
from sklearn.metrics import roc_auc_score

roc_auc_score(y_test,grid.predict_proba(x_test)[:,1])
```

```
1.0
```

K-Nearest Neighbor

Hyper tuning Parameters grid:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_algorithm	param_weights	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
0	0.005845	0.002815	0.188478	0.011524	auto	uniform	{'algorithm': 'auto', 'weights': 'uniform'}	0.923168	0.968974	0.938259	0.968974	0.968508	0.953577	0.019269	1
1	0.00446	0.000542	0.191545	0.016062	auto	distance	{'algorithm': 'auto', 'weights': 'distance'}	0.923168	0.968974	0.938259	0.968974	0.968508	0.953577	0.019269	1
2	0.005321	0.001406	0.013454	0.001234	ball_tree	uniform	{'algorithm': 'ball_tree', 'weights': 'uniform'}	0.923168	0.968974	0.938259	0.968974	0.968508	0.953577	0.019269	1
3	0.003346	0.000546	0.008332	0.000522	ball_tree	distance	{'algorithm': 'ball_tree', 'weights': 'distance'}	0.923168	0.968974	0.938259	0.968974	0.968508	0.953577	0.019269	1
4	0.003623	0.000794	0.01229	0.0016	kd_tree	uniform	{'algorithm': 'kd_tree', 'weights': 'uniform'}	0.923168	0.968974	0.938259	0.968974	0.968508	0.953577	0.019269	1
5	0.003247	0.000232	0.009234	0.001232	kd_tree	distance	{'algorithm': 'kd_tree', 'weights': 'distance'}	0.923168	0.968974	0.938259	0.968974	0.968508	0.953577	0.019269	1
6	0.002103	0.000665	0.214797	0.004446	brute	uniform	{'algorithm': 'brute', 'weights': 'uniform'}	0.923168	0.968974	0.938259	0.968974	0.968508	0.953577	0.019269	1
7	0.003453	0.000468	0.051256	0.081017	brute	distance	{'algorithm': 'brute', 'weights': 'distance'}	0.923168	0.968974	0.938259	0.968974	0.968508	0.953577	0.019269	1

Evaluation Metrics Report:

```
              precision    recall  f1-score   support

           0       0.97      1.00      0.98        32
           1       1.00      0.98      0.99        48

    accuracy                           0.99        80
   macro avg       0.98      0.99      0.99        80
weighted avg       0.99      0.99      0.99        80
```

ROC_AUC Score:

```
from sklearn.metrics import roc_auc_score

roc_auc_score(y_test,grid.predict_proba(x_test)[:,1])
```

```
0.9993489583333334
```

Naïve Bayes

Evaluation Metrics Report:

```
              precision    recall  f1-score   support

           0       0.67      1.00      0.80        32
           1       1.00      0.67      0.80        48

    accuracy                           0.80        80
   macro avg       0.83      0.83      0.80        80
weighted avg       0.87      0.80      0.80        80
```

ROC AUC Score:

```python
from sklearn.metrics import roc_auc_score

roc_auc_score(y_test,classifier.predict_proba(x_test)[:,1])
```

```
0.8776041666666666
```

5. Best Model

The Best Model I have selected is "**Random Forest**" algorithm

Reason

The classification model has demonstrated excellent performance across all evaluation metrics. Notably, the **ROC-AUC score is 1.0**, which indicates a perfect separation between the positive and negative classes. This suggests that the model has learned the patterns in the data extremely well and is highly effective at predicting Chronic Kidney Disease (CKD) outcomes.

All other metrics such as accuracy, precision, recall, and F1-score are also consistently high, reinforcing the fact that this is a well-generalized and robust model suitable for deployment or further validation.