AI-PROJECT 2
ASSIGNMENT-CLASSIFICATION ALGORITHM

Ai | Subramani

AI PREDICTION

1. 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.

2. Dataset

File: **CKD.csv**

- [399 rows x 25 columns] dataset

```
25 Columns:
Index(['age', 'bp', 'sg', 'al', 'su', 'rbc', 'pc', 'pcc', 'ba', 'bgr', 'bu',
       'sc', 'sod', 'pot', 'hrmo', 'pcv', 'wc', 'rc', 'htn', 'dm', 'cad',
       'appet', 'pe', 'ane', 'classification'],
      dtype='object')
```

3. Domain Prediction: Machine Learning

- The predicted value is numeric yes or No (1 or 0).
- Data is number

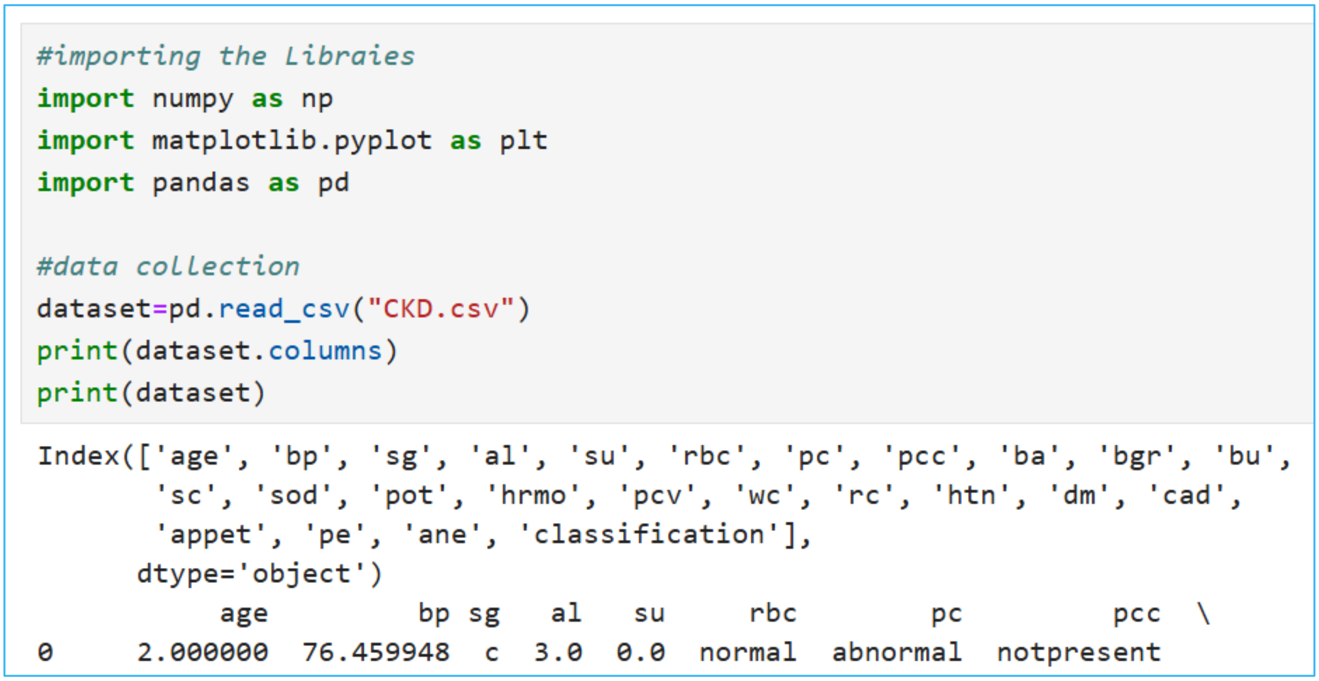
4. Learning Prediction: Supervised Learning

- Requirement is clear
- Both input and output data are available.

5. Algorithm Prediction: Classification

- Prediction is a yes/no, so this is a classification problem.
- We have 24 inputs and 1 output.
- Since there is **more than one input**, we can predict using the following classification algorithms:
  1. Logistic Regression
  2. Support Vector Classification (SVM)
  3. Decision Tree Classification
  4. **Random Forest Classification (F1 macro value=0.9917 | roc_auc score = 1.0)**
  5. K Nearest Neighbour Classification
  6. Naive Bayes (GaussianNB)

1.Data Collection:



```
#importing the Libraies
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

#data collection
dataset=pd.read_csv("CKD.csv")
print(dataset.columns)
print(dataset)
```

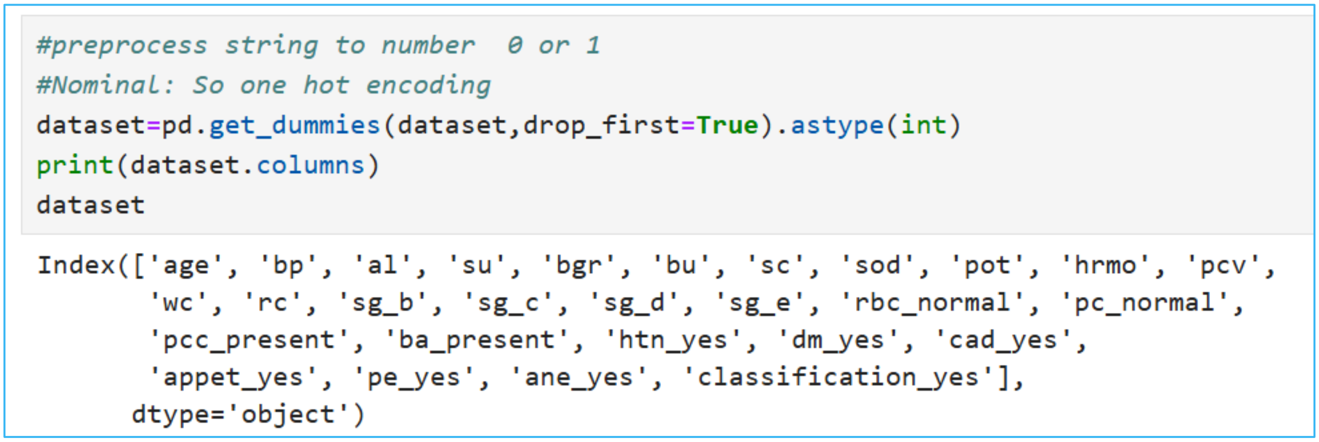
```
Index(['age', 'bp', 'sg', 'al', 'su', 'rbc', 'pc', 'pcc', 'ba', 'bgr', 'bu',
       'sc', 'sod', 'pot', 'hrmo', 'pcv', 'wc', 'rc', 'htn', 'dm', 'cad',
       'appet', 'pe', 'ane', 'classification'],
      dtype='object')
          age         bp sg   al   su     rbc        pc         pcc  \
0    2.000000  76.459948  c  3.0  0.0  normal  abnormal  notpresent
```

2. Data Preprocessing:

Categorical: **Nominal:** So **one hot encoding**

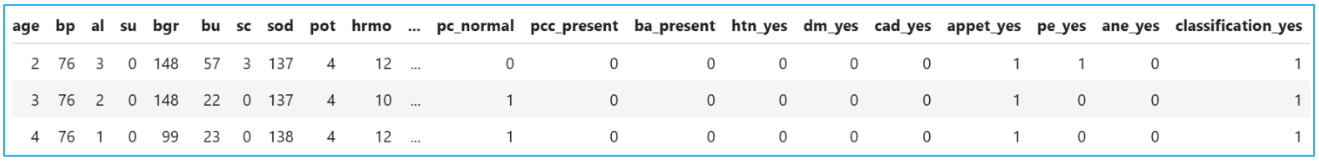
Converted string into number

Columns	String	Number
sg	a,b,c,d,e	1,0
rcb	Normal, abnormal	1,0
pc	Normal, abnormal	1,0
pcc	present , notpresent	1,0
ba	present , notpresent	1,0
htn	Yes,no	1,0
dm	Yes,no	1,0
cad	Yes,no	1,0
appet	Yes,poor	1,0
pe	Yes,poor	1,0
ane	Yes,no	1,0
classification	Yes,no	1,0



```
#preprocess string to number  0 or 1
#Nominal: So one hot encoding
dataset=pd.get_dummies(dataset,drop_first=True).astype(int)
print(dataset.columns)
dataset
```

```
Index(['age', 'bp', 'al', 'su', 'bgr', 'bu', 'sc', 'sod', 'pot', 'hrmo', 'pcv',
       'wc', 'rc', 'sg_b', 'sg_c', 'sg_d', 'sg_e', 'rbc_normal', 'pc_normal',
       'pcc_present', 'ba_present', 'htn_yes', 'dm_yes', 'cad_yes',
       'appet_yes', 'pe_yes', 'ane_yes', 'classification_yes'],
      dtype='object')
```



age	bp	al	su	bgr	bu	sc	sod	pot	hrmo	...	pc_normal	pcc_present	ba_present	htn_yes	dm_yes	cad_yes	appet_yes	pe_yes	ane_yes	classification_yes
2	76	3	0	148	57	3	137	4	12	...	0	0	0	0	0	0	1	1	0	1
3	76	2	0	148	22	0	137	4	10	...	1	0	0	0	0	0	1	0	0	1
4	76	1	0	99	23	0	138	4	12	...	1	0	0	0	0	0	1	0	0	1

399 rows × 28 columns

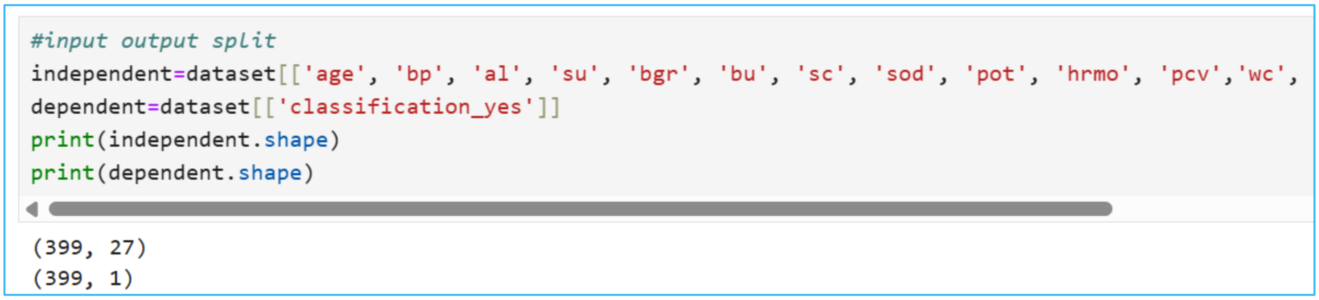
5. Split input & output:

Split X input:

independent=dataset[['age', 'bp', 'al', 'su', 'bgr', 'bu', 'sc', 'sod', 'pot', 'hrmo', 'pcv','wc', 'rc', 'sg_b', 'sg_c', 'sg_d', 'sg_e', 'rbc_normal', 'pc_normal','pcc_present', 'ba_present', 'htn_yes', 'dm_yes', 'cad_yes','appet_yes', 'pe_yes', 'ane_yes'**]]**
399 rows × 27 columns

Split y output:

dependent=dataset[['classification_yes'**]]**
399 rows × 1 columns



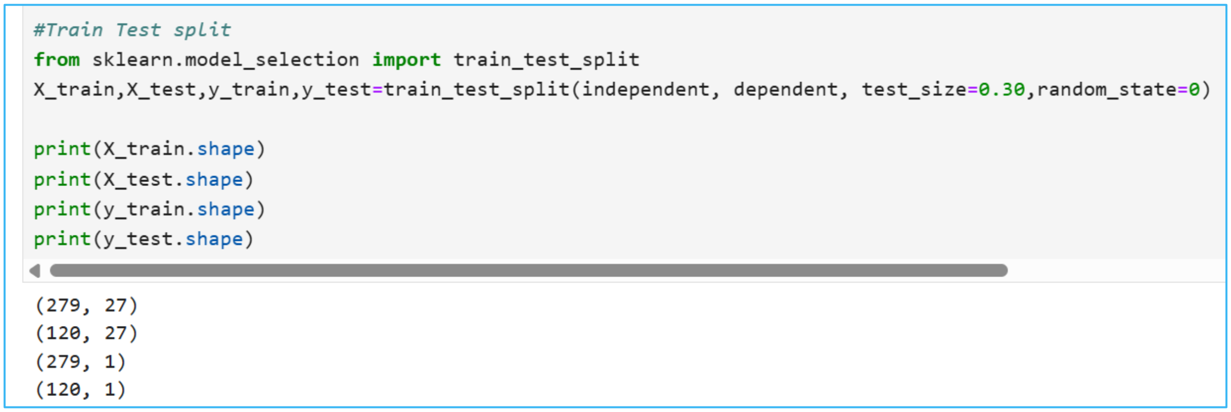
```
#input output split
independent=dataset[['age', 'bp', 'al', 'su', 'bgr', 'bu', 'sc', 'sod', 'pot', 'hrmo', 'pcv','wc',
dependent=dataset[['classification_yes']]
print(independent.shape)
print(dependent.shape)
```

```
(399, 27)
(399, 1)
```

6. Split Training Set(70%) and Test Set(30%):

70% dataset X training & y training set : X *279 rows × 27 columns* | y *279 rows × 1 column*

30% dataset X test set & y test set: X *120 rows × 27 columns* | y *120 rows × 1 column*



```
#Train Test split
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(independent, dependent, test_size=0.30,random_state=0)

print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

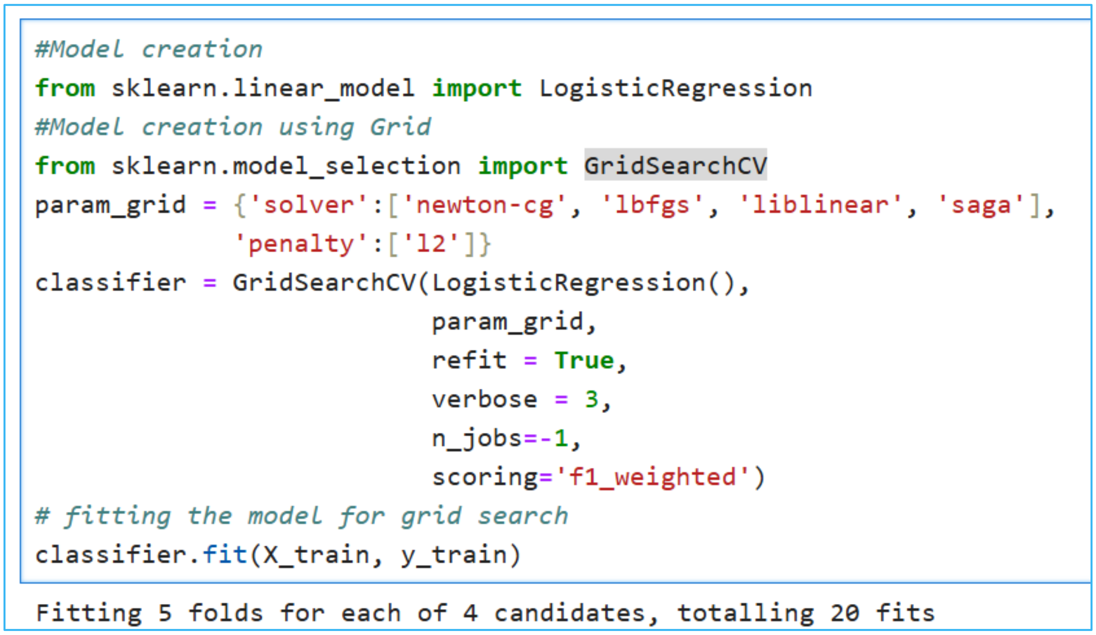
```
(279, 27)
(120, 27)
(279, 1)
(120, 1)
```

7. Model Creation

Create the Model: Using 70% of the data as the training set (X_train, y_train) with the following algorithms.

GridSearchCV is a tool in scikit-learn used to find the best hyperparameters for a model.

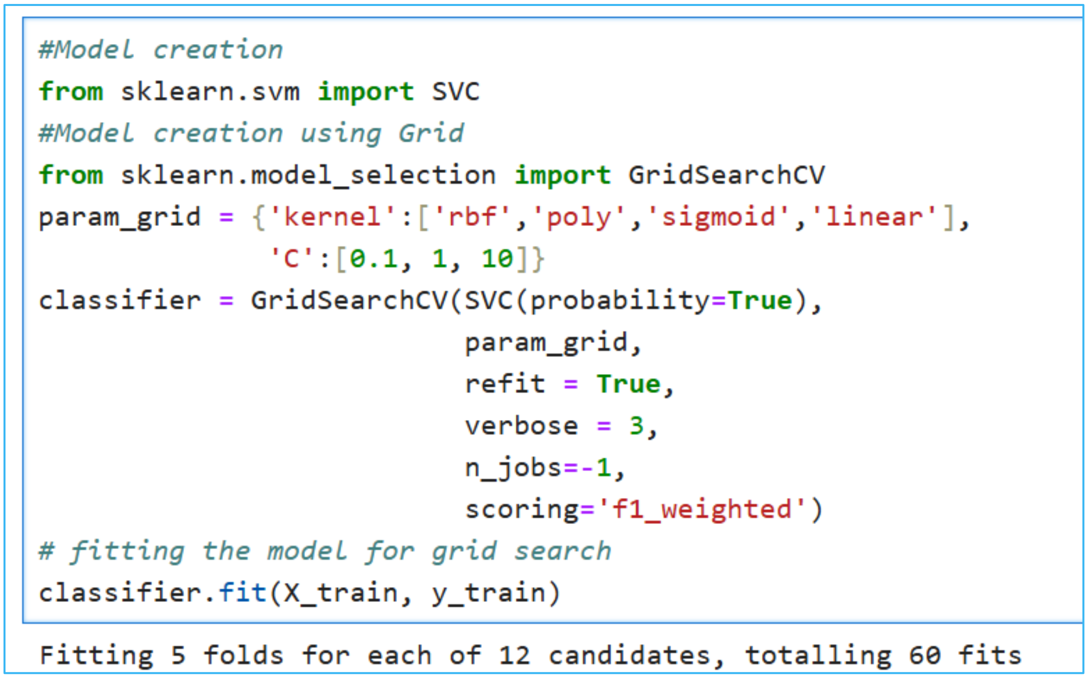
7.1 Logistic Regression:



```
#Model creation
from sklearn.linear_model import LogisticRegression
#Model creation using Grid
from sklearn.model_selection import GridSearchCV
param_grid = {'solver':['newton-cg', 'lbfgs', 'liblinear', 'saga'],
             'penalty':['l2']}
classifier = GridSearchCV(LogisticRegression(),
                          param_grid,
                          refit = True,
                          verbose = 3,
                          n_jobs=-1,
                          scoring='f1_weighted')
# fitting the model for grid search
classifier.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 4 candidates, totalling 20 fits
```

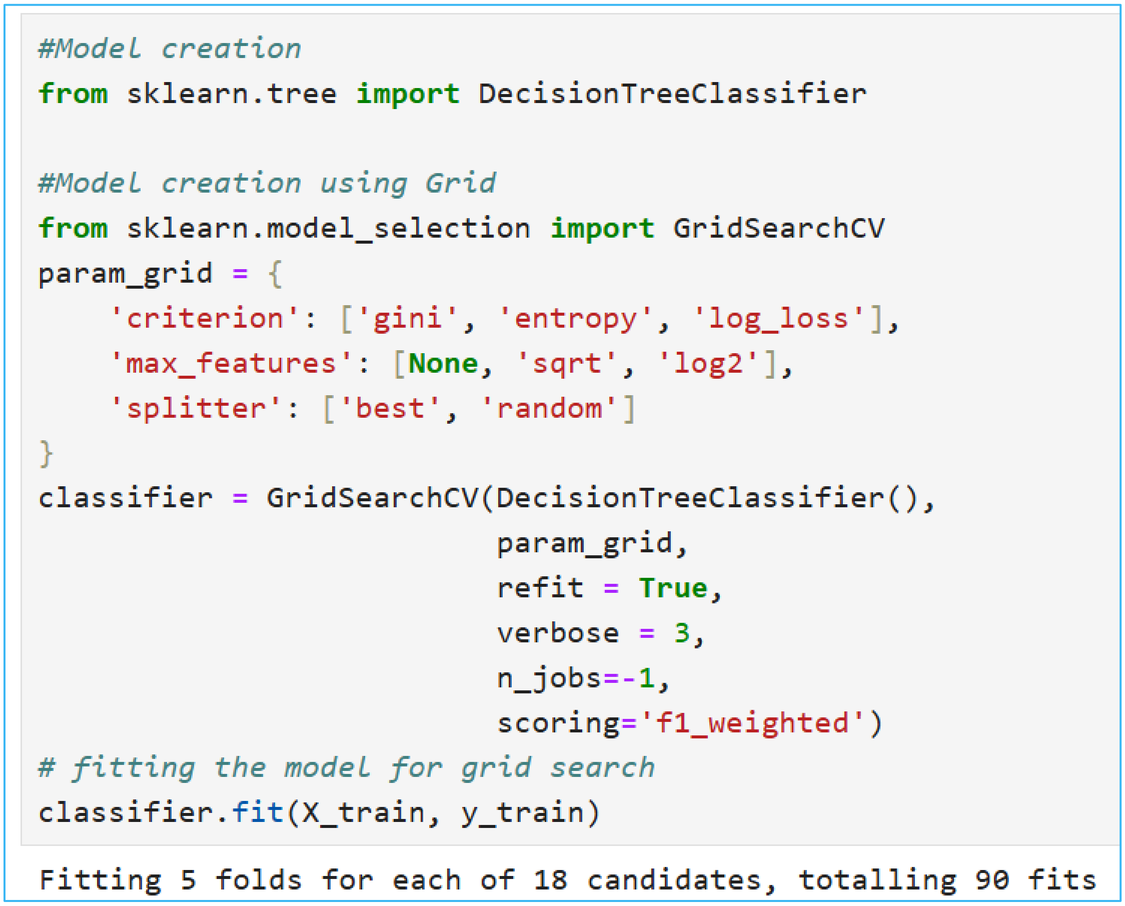
7.2 Support Vector Classification (SVM)



```
#Model creation
from sklearn.svm import SVC
#Model creation using Grid
from sklearn.model_selection import GridSearchCV
param_grid = {'kernel':['rbf','poly','sigmoid','linear'],
              'C':[0.1, 1, 10]}
classifier = GridSearchCV(SVC(probability=True),
                          param_grid,
                          refit = True,
                          verbose = 3,
                          n_jobs=-1,
                          scoring='f1_weighted')
# fitting the model for grid search
classifier.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
```

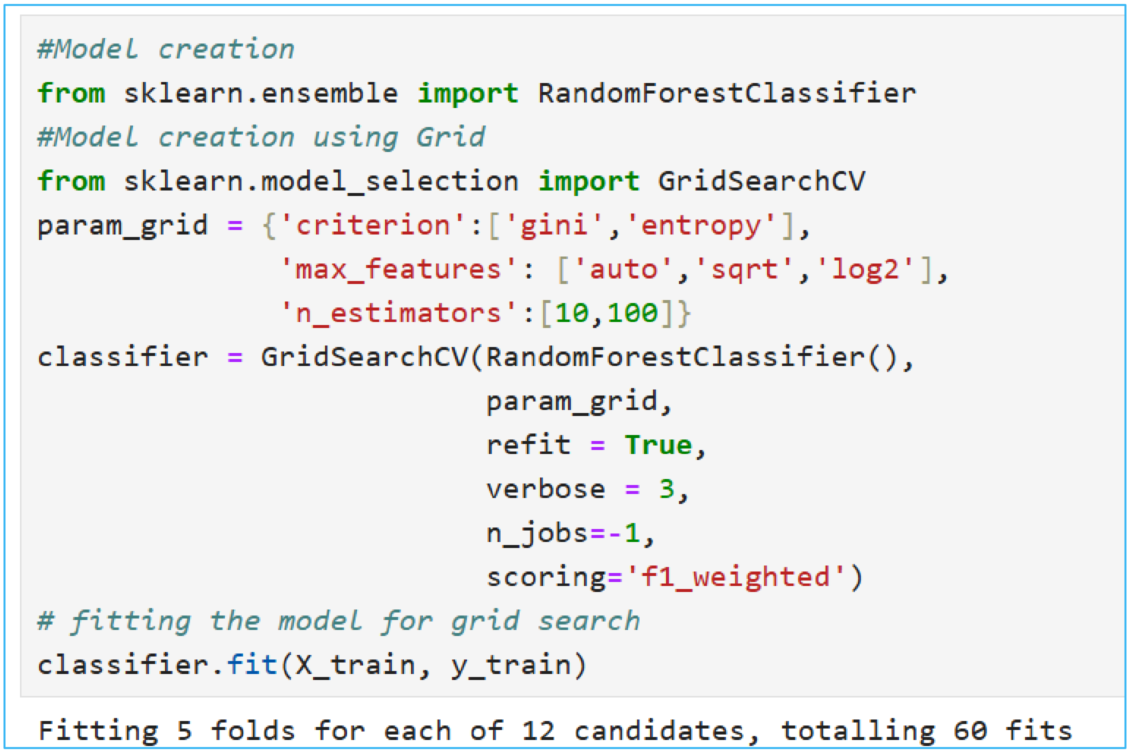
7.3 Decision Tree Classification



```
#Model creation
from sklearn.tree import DecisionTreeClassifier

#Model creation using Grid
from sklearn.model_selection import GridSearchCV
param_grid = {
    'criterion': ['gini', 'entropy', 'log_loss'],
    'max_features': [None, 'sqrt', 'log2'],
    'splitter': ['best', 'random']
}
classifier = GridSearchCV(DecisionTreeClassifier(),
                          param_grid,
                          refit = True,
                          verbose = 3,
                          n_jobs=-1,
                          scoring='f1_weighted')
# fitting the model for grid search
classifier.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 18 candidates, totalling 90 fits
```

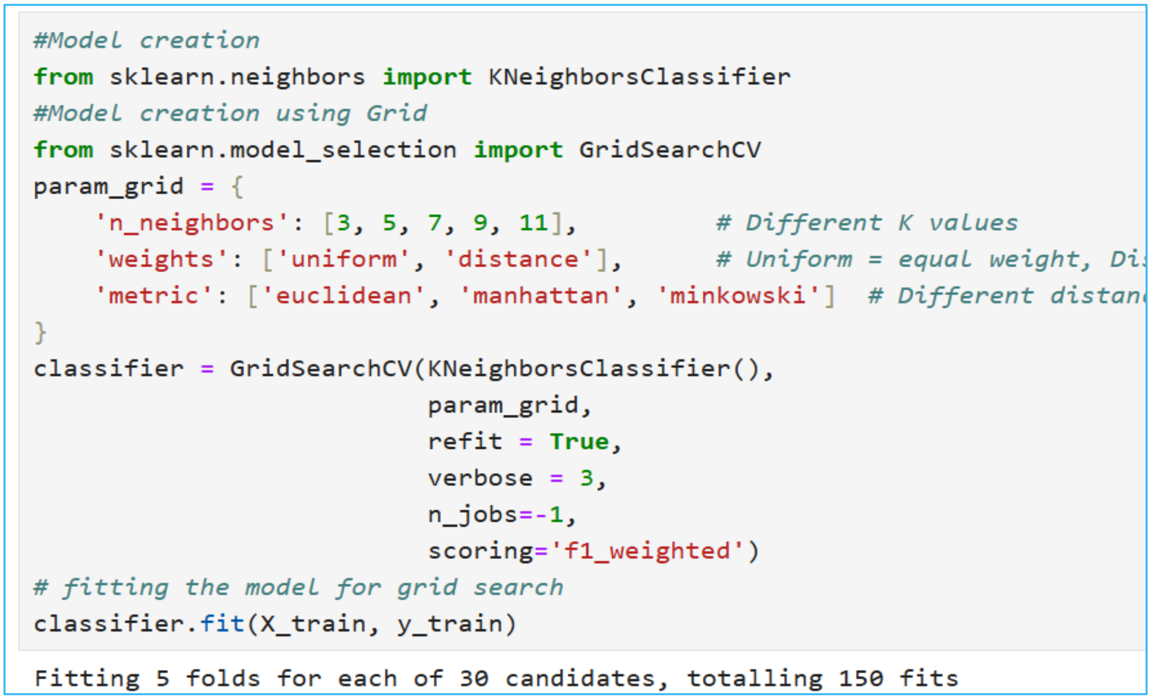
7.4 Random Forest Classification



```
#Model creation
from sklearn.ensemble import RandomForestClassifier
#Model creation using Grid
from sklearn.model_selection import GridSearchCV
param_grid = {'criterion':['gini','entropy'],
              'max_features': ['auto','sqrt','log2'],
              'n_estimators':[10,100]}
classifier = GridSearchCV(RandomForestClassifier(),
                          param_grid,
                          refit = True,
                          verbose = 3,
                          n_jobs=-1,
                          scoring='f1_weighted')
# fitting the model for grid search
classifier.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
```

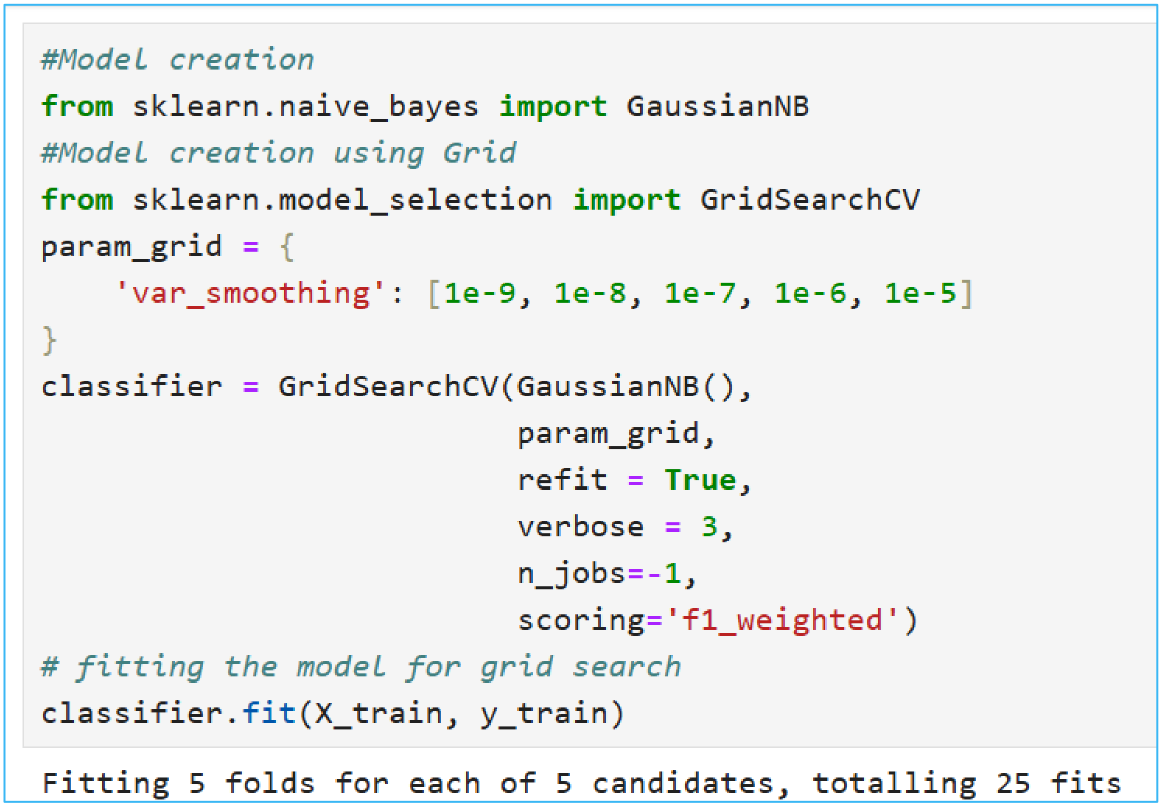
7.5 K-Nearest Neighbour Classification



```
#Model creation
from sklearn.neighbors import KNeighborsClassifier
#Model creation using Grid
from sklearn.model_selection import GridSearchCV
param_grid = {
    'n_neighbors': [3, 5, 7, 9, 11],         # Different K values
    'weights': ['uniform', 'distance'],      # Uniform = equal weight, Di
    'metric': ['euclidean', 'manhattan', 'minkowski']  # Different distan
}
classifier = GridSearchCV(KNeighborsClassifier(),
                          param_grid,
                          refit = True,
                          verbose = 3,
                          n_jobs=-1,
                          scoring='f1_weighted')
# fitting the model for grid search
classifier.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 30 candidates, totalling 150 fits
```

7.6 Naive Bayes (GaussianNB)



```
#Model creation
from sklearn.naive_bayes import GaussianNB
#Model creation using Grid
from sklearn.model_selection import GridSearchCV
param_grid = {
    'var_smoothing': [1e-9, 1e-8, 1e-7, 1e-6, 1e-5]
}
classifier = GridSearchCV(GaussianNB(),
                          param_grid,
                          refit = True,
                          verbose = 3,
                          n_jobs=-1,
                          scoring='f1_weighted')
# fitting the model for grid search
classifier.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits
```

8. Evaluation Metrics: Confusion Matrix (Best Model: Random Forest result below)

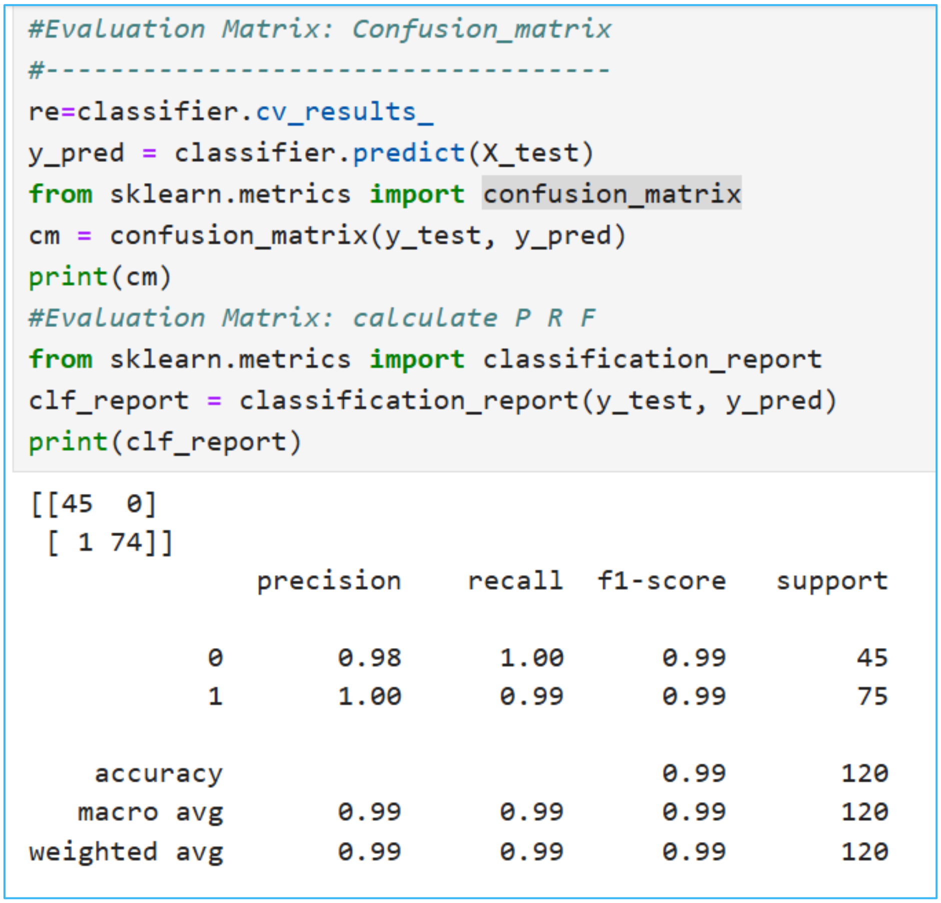
Each model will be evaluated on the **30% test set** (**`x_test`, `y_test`**) using the following metrics:

- **Confusion Matrix** – shows the counts of **True Positives (TP=45), True Negatives (TN=75), False Positives (FP=1), and False Negatives (FN=0)**, which helps to visualize classification performance.
- **Classification Report** – provides detailed metrics for each class:
  • **Precision** – proportion of correctly predicted positives among all predicted positives.
  • **Recall** – proportion of correctly predicted positives among all actual positives.
  • **F1-Score** – harmonic mean of precision and recall.

  Additionally, the report includes:
  • **Macro Average (macro avg = 0.9917)** – the unweighted mean of metrics across all classes (treats all classes equally, regardless of their size).
  • **Weighted Average (weighted avg = 0.99)** – the mean of metrics weighted by the number of instances in each class (better when classes are imbalanced).

- **Accuracy Score** (0.99)– measures the overall proportion of correctly classified instances out of total predictions.

Best Confusion Matrix: Random Forest

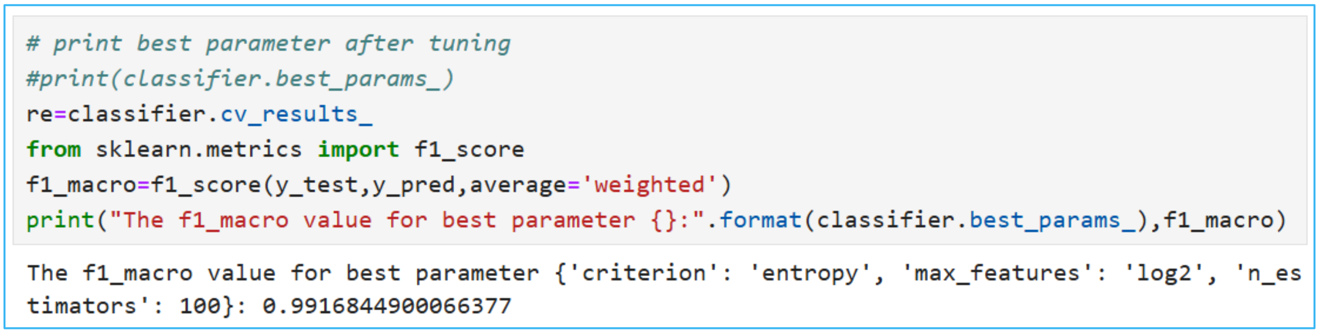


```
#Evaluation Matrix: Confusion_matrix
#-----------------------------------
re=classifier.cv_results_
y_pred = classifier.predict(X_test)
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
#Evaluation Matrix: calculate P R F
from sklearn.metrics import classification_report
clf_report = classification_report(y_test, y_pred)
print(clf_report)
```

```
[[45  0]
 [ 1 74]]
              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
```

Best Macro Average: 0.9917 in Random Forest

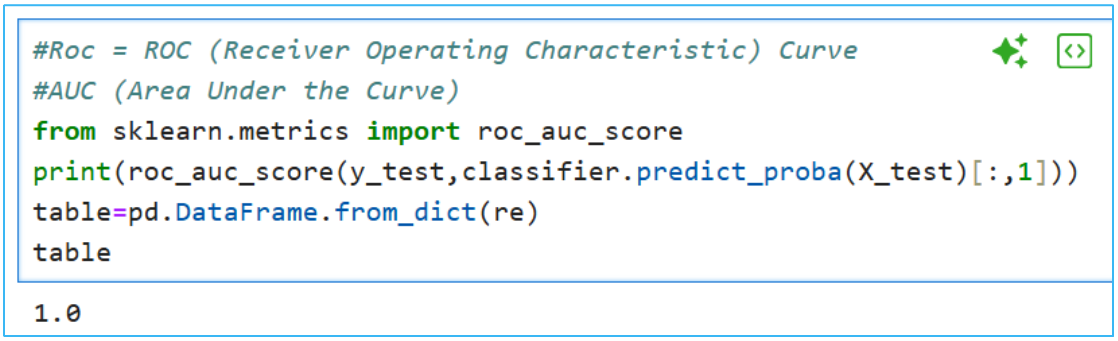
Best Parameter: Random Forest {'criterion': 'entropy', 'max_features': 'log2', 'n_estimators': 100}



```
# print best parameter after tuning
#print(classifier.best_params_)
re=classifier.cv_results_
from sklearn.metrics import f1_score
f1_macro=f1_score(y_test,y_pred,average='weighted')
print("The f1_macro value for best parameter {}:".format(classifier.best_params_),f1_macro)
```

```
The f1_macro value for best parameter {'criterion': 'entropy', 'max_features': 'log2', 'n_es
timators': 100}: 0.9916844900066377
```

Best roc_auc_score: 0.1 in Random Forest



```
#Roc = ROC (Receiver Operating Characteristic) Curve
#AUC (Area Under the Curve)
from sklearn.metrics import roc_auc_score
print(roc_auc_score(y_test,classifier.predict_proba(X_test)[:,1]))
table=pd.DataFrame.from_dict(re)
table
```

```
1.0
```

6. Model Compared with Different Classification Algorithm

(Confusion Matrix: F1-macro score and roc_auc score)

Best Model is Random Forest Classification

Best Model f1_macro value: 0.9917

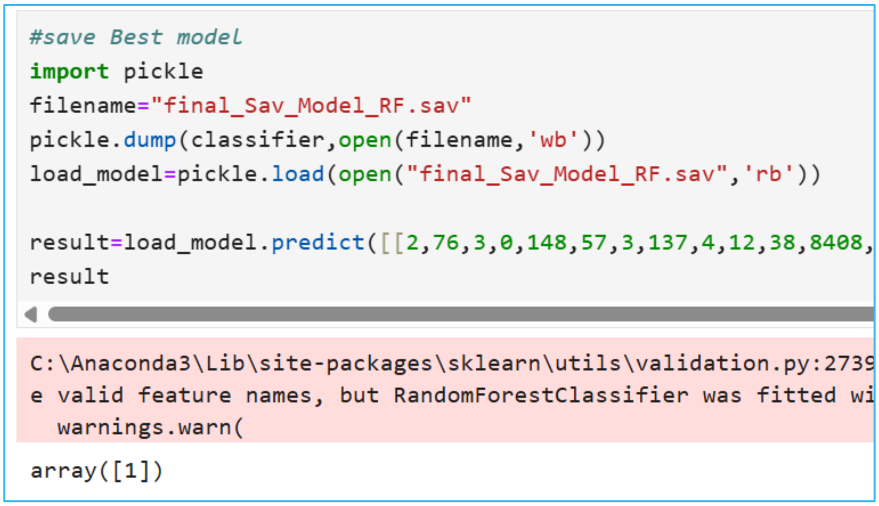
Best Model roc_auc score value: 1.0

Best Model parameter: {'criterion': 'entropy', 'max_features': 'log2', 'n_estimators': 100}

Algorithm	Best Parameter	f1_macro value	roc_auc score
Logistic Regression	{'penalty': 'l2', 'solver': 'liblinear'}	0.9749	0.9979
Support Vector Classification (SVM)	{'C': 0.1, 'kernel': 'linear'}	0.9916	0.9991
Decision Tree Classification	{'criterion': 'entropy', 'max_features': None, 'splitter': 'random'}	0.9750	0.9755
Random Forest Classification	{'criterion': 'entropy', 'max_features': 'log2', 'n_estimators': 100}	**0.9917**	**1.0**
K Nearest Neighbour Classification	{'metric': 'manhattan', 'n_neighbors': 9, 'weights': 'distance'}	0.7863	0.8642
Naive Bayes (GaussianNB)	{'var_smoothing': 1e-09}	0.9834	1.0

7. Save the Best Model

Compared to all models, **Random Forest Classification** achieved the highest **f1_macro value** of 0.9917 and **roc_auc score** value: 1.0, making it the best model to save.



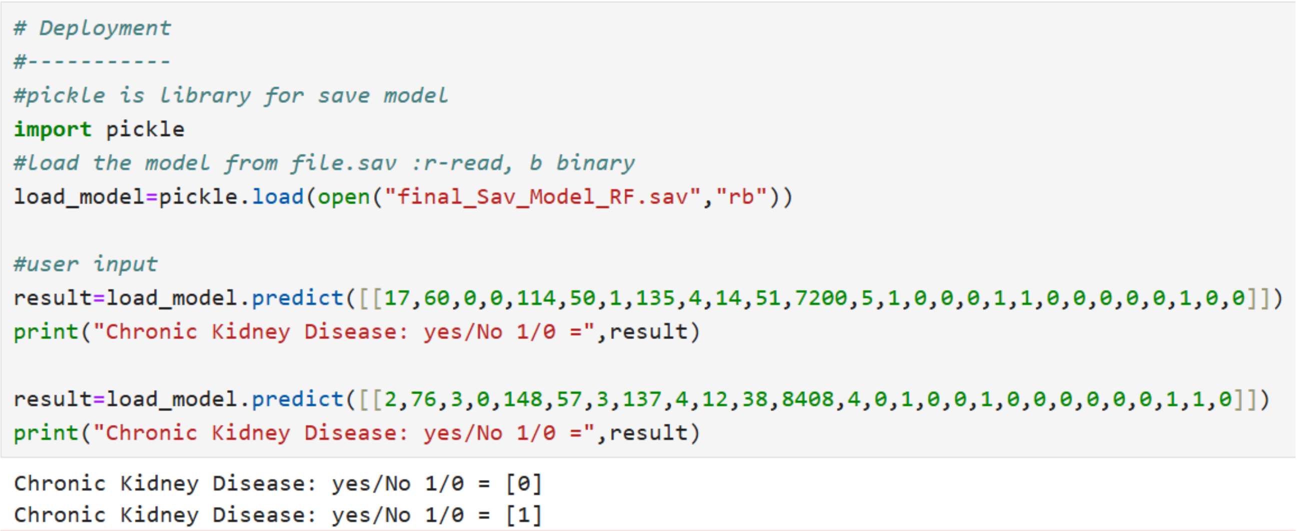
```
#save Best model
import pickle
filename="final_Sav_Model_RF.sav"
pickle.dump(classifier,open(filename,'wb'))
load_model=pickle.load(open("final_Sav_Model_RF.sav",'rb'))

result=load_model.predict([[2,76,3,0,148,57,3,137,4,12,38,8408,
result
```

```
C:\Anaconda3\Lib\site-packages\sklearn\utils\validation.py:2739
e valid feature names, but RandomForestClassifier was fitted wi
  warnings.warn(
```

```
array([1])
```

8. Deployment / Implement

For the deployment process, load the best **Random Forest Classification** model (which achieved the highest **f1_macro value** of 0.9917 and **roc_auc score** value: 1.0) and take user input to predict the **Chronic Kidney Disease: yes/No**.



```
# Deployment
#-----------
#pickle is library for save model
import pickle
#load the model from file.sav :r-read, b binary
load_model=pickle.load(open("final_Sav_Model_RF.sav","rb"))

#user input
result=load_model.predict([[17,60,0,0,114,50,1,135,4,14,51,7200,5,1,0,0,0,1,1,0,0,0,0,0,1,0,0]])
print("Chronic Kidney Disease: yes/No 1/0 =",result)

result=load_model.predict([[2,76,3,0,148,57,3,137,4,12,38,8408,4,0,1,0,0,1,0,0,0,0,0,0,1,1,0]])
print("Chronic Kidney Disease: yes/No 1/0 =",result)
```

```
Chronic Kidney Disease: yes/No 1/0 = [0]
Chronic Kidney Disease: yes/No 1/0 = [1]
```

9. Call to Action

The final model will serve as the call to action.