PROBLEM STATEMENT

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.

1.) Identify your problem statement

2.) Tell basic info about the dataset (Total number of rows, columns)

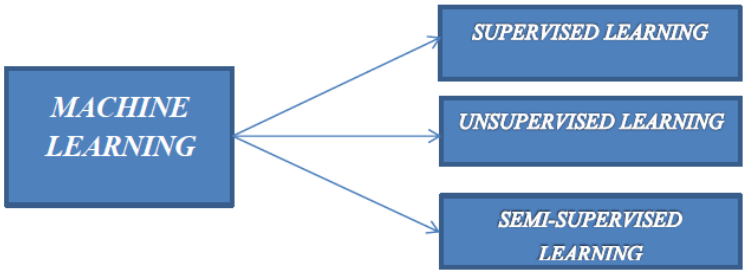
3.) Mention the pre-processing method if you're doing any (like converting string to number – nominal data)

4.) Develop a good model with good evaluation metric. You can use any machine learning algorithm; you can create many models. Finally, you have to come up with final model.

5.) All the research values of each algorithm should be documented. (You can make tabulation or screenshot of the results.)

6.) Mention your final model, justify why u have chosen the same.

Algorithm Identification



1) In machine learning, the requirement should be clear in the given dataset. Input and output are well defined.

Then it is **SUPERVISED LEARNING**

2) Under supervised learning, there are 2 types: Classification and Regression.

i)Classification - It has categorical
ii)Regression - It has numerical

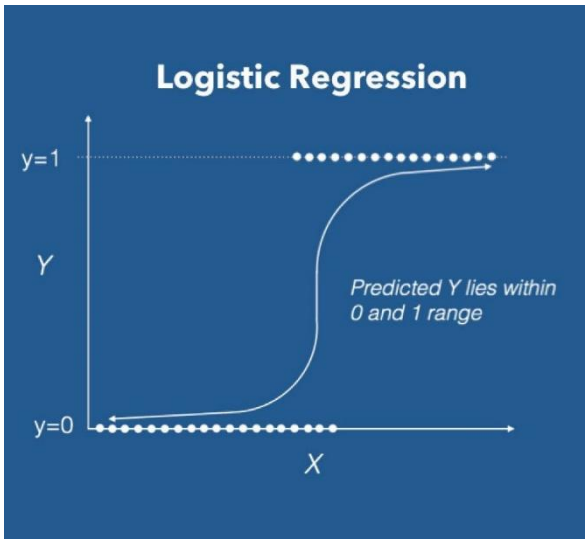
An output parameter has Categorical value, then it is Classification

3)In regression algorithm, using **Logistic Classification, Support vector machine, Decision tree and Random forest** on the model

Logistic Classification

A) **Logistic Classification** – The logistic classification model (or logit model) is a binary classification model in which the conditional probability of one of the two possible realizations of the output variable is assumed to be equal to a linear combination of the input variables, transformed by the logistic function.

MATPLOT-(**Logistic Classification)**



1. Importing the Libraries - (input)

```
import numpy as np # numpy is numerical
import matplotlib.pyplot as plt # matplotlip is a plot the points in graph
import pandas as pd #used for data analysis
```

2. Loaded the excel into Jupyter Notebook - (input)

```
# Reading the Dataset
dataset = pd.read_csv('CKD.csv')
```

3. Analysing & Visualization the given Data - (output)

```
3]: dataset
```

3]:

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	...	pcv	wc	rc	htn	dm	cad	appet	pe
0	2.000000	76.459948	c	3.0	0.0	normal	abnormal	notpresent	notpresent	148.112676	...	38.868902	8408.191126	4.705597	no	no	no	yes	yes
1	3.000000	76.459948	c	2.0	0.0	normal	normal	notpresent	notpresent	148.112676	...	34.000000	12300.000000	4.705597	no	no	no	yes	poor
2	4.000000	76.459948	a	1.0	0.0	normal	normal	notpresent	notpresent	99.000000	...	34.000000	8408.191126	4.705597	no	no	no	yes	poor
3	5.000000	76.459948	d	1.0	0.0	normal	normal	notpresent	notpresent	148.112676	...	38.868902	8408.191126	4.705597	no	no	no	yes	poor
4	5.000000	50.000000	c	0.0	0.0	normal	normal	notpresent	notpresent	148.112676	...	36.000000	12400.000000	4.705597	no	no	no	yes	poor
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
394	51.492308	70.000000	a	0.0	0.0	normal	normal	notpresent	notpresent	219.000000	...	37.000000	9800.000000	4.400000	no	no	no	yes	poor
395	51.492308	70.000000	c	0.0	2.0	normal	normal	notpresent	notpresent	220.000000	...	27.000000	8408.191126	4.705597	yes	yes	no	yes	poor
396	51.492308	70.000000	c	3.0	0.0	normal	normal	notpresent	notpresent	110.000000	...	26.000000	9200.000000	3.400000	yes	yes	no	poor	poor
397	51.492308	90.000000	a	0.0	0.0	normal	normal	notpresent	notpresent	207.000000	...	38.868902	8408.191126	4.705597	yes	yes	no	yes	poor
398	51.492308	80.000000	a	0.0	0.0	normal	normal	notpresent	notpresent	100.000000	...	53.000000	8500.000000	4.900000	no	no	no	yes	poor

4. Create the dummies Bec string value is can't find ml authorisms

(input) (eg):Help to create dataset

```
: dataset=pd.get_dummies(dataset,drop_first=True)
```

```
: dataset
```

(Output)

	age	bp	al	su	bgr	bu	sc	sod	pot	hrmo	...	pc_normal	pcc_present	ba_present	htn_yes	dm_\
0	2.000000	76.459948	3.0	0.0	148.112676	57.482105	3.077356	137.528754	4.627244	12.518156	...	0	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	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	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	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	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	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	1	
396	51.492308	70.000000	3.0	0.0	110.000000	115.000000	6.000000	134.000000	2.700000	9.100000	...	1	0	0	1	
397	51.492308	90.000000	0.0	0.0	207.000000	80.000000	6.800000	142.000000	5.500000	8.500000	...	1	0	0	1	
398	51.492308	80.000000	0.0	0.0	100.000000	49.000000	1.000000	140.000000	5.000000	16.300000	...	1	0	0	0	

399 rows × 28 columns

5. Separate the Dataset input and output column

Input column

- **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

Output column

- classification_yes

(input)

```
indep=dataset[['age','bp','al','su','bgr','bu','sc','sod','pot','hrmo','pc_normal','pcc_present','ba_present','htn_yes','dm_yes',
dep=dataset['classification_yes']
```

a)Independent Input (Output)

```
indep
```

	age	bp	al	su	bgr	bu	sc	sod	pot	
0	2.000000	76.459948	3.0	0.0	148.112676	57.482105	3.077356	137.528754	4.627244	12.5
1	3.000000	76.459948	2.0	0.0	148.112676	22.000000	0.700000	137.528754	4.627244	10.7
2	4.000000	76.459948	1.0	0.0	99.000000	23.000000	0.600000	138.000000	4.400000	12.0
3	5.000000	76.459948	1.0	0.0	148.112676	16.000000	0.700000	138.000000	3.200000	8.1
4	5.000000	50.000000	0.0	0.0	148.112676	25.000000	0.600000	137.528754	4.627244	11.8
...	...	...	...	...	...	...	...	...	...	...
394	51.492308	70.000000	0.0	0.0	219.000000	36.000000	1.300000	139.000000	3.700000	12.5
395	51.492308	70.000000	0.0	2.0	220.000000	68.000000	2.800000	137.528754	4.627244	8.7
396	51.492308	70.000000	3.0	0.0	110.000000	115.000000	6.000000	134.000000	2.700000	9.1
397	51.492308	90.000000	0.0	0.0	207.000000	80.000000	6.800000	142.000000	5.500000	8.5
398	51.492308	80.000000	0.0	0.0	100.000000	49.000000	1.000000	140.000000	5.000000	16.3

399 rows × 19 columns

b)Dependent Output (Output)

```
dep
```

```
0      1
1      1
2      1
3      1
4      1
      ..
394    1
395    1
396    1
397    1
398    0
Name: classification_yes, Length: 399, dtype: uint8
```

6. Split the Dataset 8% training data 2% test data

```
#split into training set and test
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(indep, dep, test_size = 1/3, random_state = 0)
```

7.Logistic classifier model & formulaeg: y=W1*w2*w3*w4*w5*x1+b0 for this equation we got value for b1 and bo

(input)

```
from sklearn.linear_model import LogisticRegression
classifier=LogisticRegression(random_state=0)
classifier.fit(X_train, y_train)
```

8.Visualization the accuracy

(input)

```
from sklearn.metrics import classification_report
clf_report = classification_report(y_test, y_pred)
```

```
print(clf_report)
```

(Output)

```
              precision    recall  f1-score   support

           0       1.00      0.96      0.98        51
           1       0.98      1.00      0.99        82

    accuracy                           0.98       133
   macro avg       0.99      0.98      0.98       133
weighted avg       0.99      0.98      0.98       133
```

7. Prediction

(input)/ (Output)

```
]: age_input=float(input("age:"))
   bp_input=float(input("bp:"))
   al_input=float(input("al:"))
   su_input=float(input("su:"))
   bgr_input=float(input("bgr:"))
   bu_input=float(input("bu:"))
   sc_input=float(input("sc:"))
   sod_input=float(input("sod:"))
   pot_input=float(input("pot:"))
   hrmo_input=float(input("hrmo:"))
   pc_normal_input=int(input("pc normal 0 or 1:"))
   pcc_present_input=int(input("pcc present 0 or 1:"))
   ba_present_input=int(input("ba present 0 or 1:"))
   htn_yes_input=int(input("htn yes 0 or 1:"))
   dm_yes_input=int(input("dm yes 0 or 1:"))
   cad_yes_input=int(input("cad yes 0 or 1:"))
   appet_yes_input=int(input("appet yes 0 or 1:"))
   pe_yes_input=int(input("pe yes 0 or 1:"))
   ane_yes_input=int(input("ane yes 0 or 1:"))
```

```
age:2
bp:76.45
al:3
su:0
bgr:148.12
```

8. Future Prediction

(input)/ (Output)

```
: Future_Prediction=grid.predict([[age_input,bp_input,al_input,su_input,bgr_input,bu_input,sc_input,sod_input,pot_input,hrmo_inpu
  print("Future_Prediction={}".format(Future_Prediction))
```

```
Future_Prediction=[1]
```

9. Finding the accuracy Grid

(input)/ (Output)

```
# print best parameter after tuning
#print(grid.best_params_)
re=grid.cv_results_
#print(re)
grid_predictions = grid.predict(X_test)


from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, grid_predictions)



# print classification report
from sklearn.metrics import classification_report
clf_report = classification_report(y_test, grid_predictions)
```

A)Conclusion for Logistic Classification

```
from sklearn.metrics import f1_score
f1_macro=f1_score(y_test,grid_predictions,average='weighted')
print("The f1_macro value for best parameter {}:".format(grid.best_params_),f1_macro)
```

```
The f1_macro value for best parameter {'penalty': 'l2', 'solver': 'lbfgs'}: 0.98490261799076
```

The Logistic Classification use **F1** value **(0.98)**

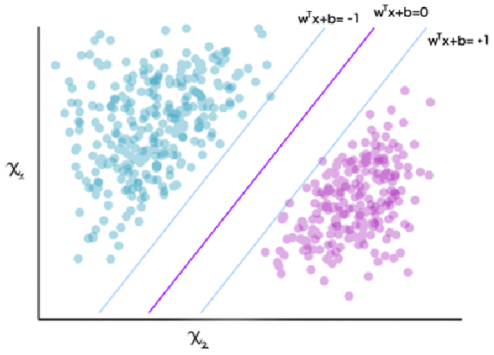
Support Vector Machine- linear Classification

B) **Support Vector Machine- linear** - SVM or Support Vector Machine is a linear model for classification and regression problems. It can solve linear and non-linear problems and work well for many practical problems. The idea of SVM is simple: The algorithm creates a line or a hyperplane which separates the data into classes..

MATPLOT-(**Support Vector Machine**)

1. Finding the R2 Value in Standard Scaler - import SVM

(input)/ (Output)



2.Visualization the accuracy

(input)/ (Output)

```
: print(clf_report)
```

	precision	recall	f1-score	support
0	0.94	1.00	0.97	51
1	1.00	0.96	0.98	82
accuracy			0.98	133
macro avg	0.97	0.98	0.98	133
weighted avg	0.98	0.98	0.98	133

(Output)

2.Finding the accuracy Grid

```
from sklearn.model_selection import GridSearchCV

param_grid = {'kernel':['linear','rbf','poly','sigmoid'],
              'gamma':['auto','scale'],
              'C':[10,100,1000,2000,3000]}



grid = GridSearchCV(SVC(), param_grid, refit = True, verbose = 3,n_jobs=-1,scoring='f1_weighted')

# fitting the model for grid search
grid.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 40 candidates, totalling 200 fits

GridSearchCV(estimator=SVC(), n_jobs=-1
```

B)Conclusion for Support Vector Machine - linear

```
from sklearn.metrics import f1_score
f1_macro=f1_score(y_test,grid_predictions,average='weighted')
print("The f1_macro value for best parameter {}:".format(grid.best_params_),f1_macro)
```

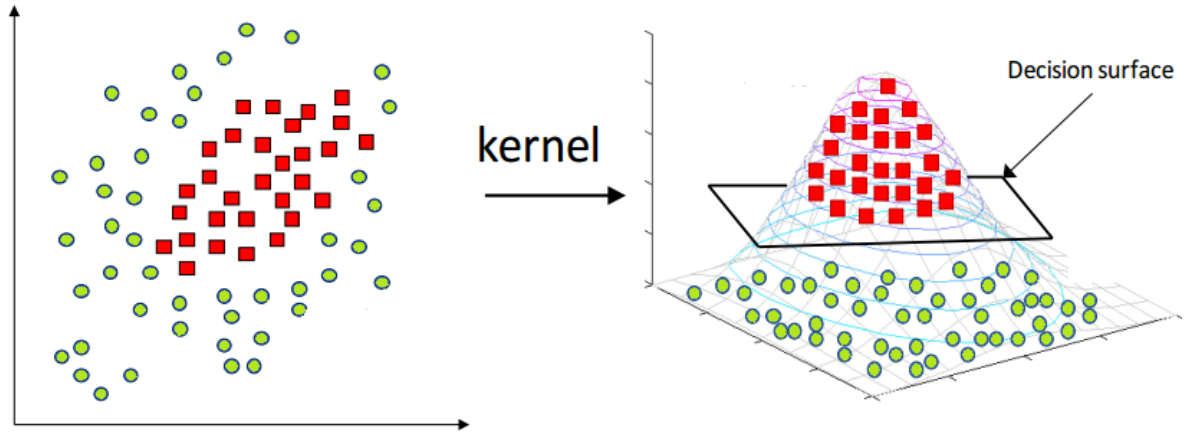
```
The f1_macro value for best parameter {'C': 10, 'gamma': 'auto', 'kernel': 'poly'}: 0.955283779067923
```

The **Support Vector Machine- linear** use **F1** value =**0.95**

Support Vector Machine Kernel Classification

B) Support Vector Machine Kernel "Kernel" is used due to set of mathematical functions used in Support Vector Machine provides the window to manipulate the data. So, Kernel Function generally transforms the training set of data so that a non-linear decision surface is able to transformed to a linear equation in a higher number of dimension spaces

MATPLOT-(**Support Vector Machine Kernel**)



2.Visualization the accuracy

(input/Output)

```
print(clf_report)
```

```
              precision    recall  f1-score   support

           0       0.68      0.92      0.78        51
           1       0.94      0.73      0.82        82

    accuracy                           0.80       133
   macro avg       0.81      0.83      0.80       133
weighted avg       0.84      0.80      0.81       133
```

(Output)

3.Finding the accuracy Grid

```
# print best parameter after tuning
#print(grid.best_params_)
re=grid.cv_results_
#print(re)
grid_predictions = grid.predict(X_test)


from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, grid_predictions)



# print classification report
from sklearn.metrics import classification_report
clf_report = classification_report(y_test, grid_predictions)
```

```
print(grid.best_params_)
```

B)Conclusion for Support Vector Machine- Kernel

```
from sklearn.metrics import f1_score
f1_macro=f1_score(y_test,grid_predictions,average='weighted')
print("The f1_macro value for best parameter {}:".format(grid.best_params_),f1_macro)
```

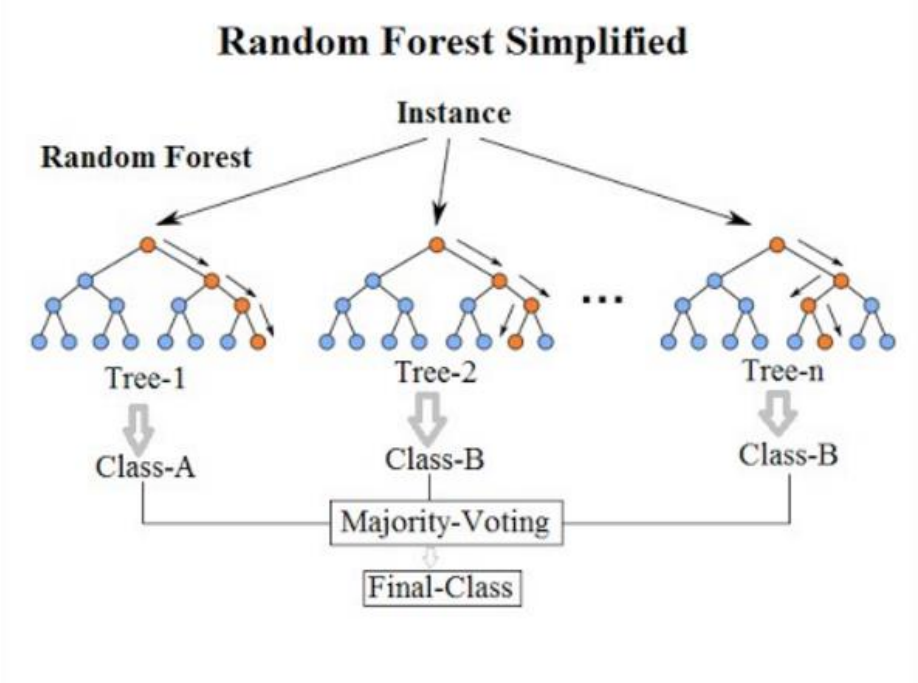
```
The f1_macro value for best parameter {'C': 1000, 'gamma': 'scale', 'kernel': 'rbf'}: 0.9625928174473452
```

The **Support Vector Machine- Kernel** use **F1** value =0.96

Random Forest Classification

D)Random Forest Classification The random forest is a classification algorithm consisting of many decisions trees. It uses bagging and feature randomness when building each individual tree to try to create an uncorrelated forest of trees whose prediction by committee is more accurate than that of any individual tree.

MATPLOT-(**Random Forest Classification**)



2.Visualization the accuracy

(input/Output)

```
print(clf_report)
```

```
              precision    recall  f1-score   support

           0       0.96      1.00      0.98        51
           1       1.00      0.98      0.99        82

    accuracy                           0.98       133
   macro avg       0.98      0.99      0.98       133
weighted avg       0.99      0.98      0.99       133
```

2.Finding the accuracy Grid

```
]: # print best parameter after tuning
   #print(grid.best_params_)
   re=grid.cv_results_
   #print(re)
   grid_predictions = grid.predict(X_test)


   from sklearn.metrics import confusion_matrix
   cm = confusion_matrix(y_test, grid_predictions)



   # print classification report
   from sklearn.metrics import classification_report
   clf_report = classification_report(y_test, grid_predictions)
```

```
]: print(grid.best_params_)
```

```
{'criterion': 'entropy', 'max_features': 'sqrt', 'n_estimators': 100}
```

B)Conclusion for Random Forest Classification

```
:
from sklearn.metrics import f1_score
f1_macro=f1_score(y_test,grid_predictions,average='weighted')
print("The f1_macro value for best parameter {}:".format(grid.best_params_),f1_macro)
```

```
The f1_macro value for best parameter {'criterion': 'entropy', 'max_features': 'sqrt', 'n_estimators': 100}: 0.9924946382275899
```

The **Random Forest Classification** use **F1** value =0.99

Decision Tree Classification

E) Decision Tree Classification A decision tree is a graphical representation of all possible solutions to a decision based on certain conditions. On each step or node of a decision tree, used for classification, we try to form a condition on the features to separate all the labels or classes contained in the dataset to the fullest purity

MATPLOT-(**Decision Tree Classification)**

2.Visualization the accuracy

(input/Output)

```
print(clf_report)
```

```
              precision    recall  f1-score   support

           0       0.92      0.96      0.94        51
           1       0.97      0.95      0.96        82

    accuracy                           0.95       133
   macro avg       0.95      0.96      0.95       133
weighted avg       0.96      0.95      0.96       133
```

(Output)

2.Finding the accuracy Grid

```
# print best parameter after tuning
#print(grid.best_params_)
re=grid.cv_results_
#print(re)
grid_predictions = grid.predict(X_test)


from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, grid_predictions)



# print classification report
from sklearn.metrics import classification_report
clf_report = classification_report(y_test, grid_predictions)
```

```
print(grid.best_params_)
```

```
{'criterion': 'gini', 'max_features': 'sqrt', 'splitter': 'random'}
```

B)Conclusion for Decision Tree Classification

```
from sklearn.metrics import f1_score
f1_macro=f1_score(y_test,grid_predictions,average='weighted')
print("The f1_macro value for best parameter {}:".format(grid.best_params_),f1_macro)
```

```
The f1_macro value for best parameter {'criterion': 'gini', 'max_features': 'sqrt', 'splitter': 'random'}: 0.9398496240601504
```

The **Decision Tree Classification** use **F1** value =0.93

8. FINAL MODEL

Using regression algorithms, to calculate the accuracy value to know the best performance of a model.

Algorithm	Accuracy
Logistic Classification	0.98
Support Vector Machine- Kernel	0.96
Support Vector Machine- linear	0.95
Random Forest Classification	0.99
Decision Tree Classification	0.93

Tabulation of accuracy value using Regression Algorithms

The final machine learning best method of Classification:

Decision Tree Classification accuracy (`criterion': 'gini', 'max_features': 'sqrt', 'splitter': 'random'`)

1. =**0.93**

Thank You

M.Ezhilarasan form hope