EC-412 MACHINE LEARNING INNOVATIVE PROJECT

Topic: - Dietary Prediction for Patients with Chronic Kidney Disease (CKD) using Multi-Class Classification Algorithms.

Name - Aditya Agrawal
Roll No - 2K17/EC/008

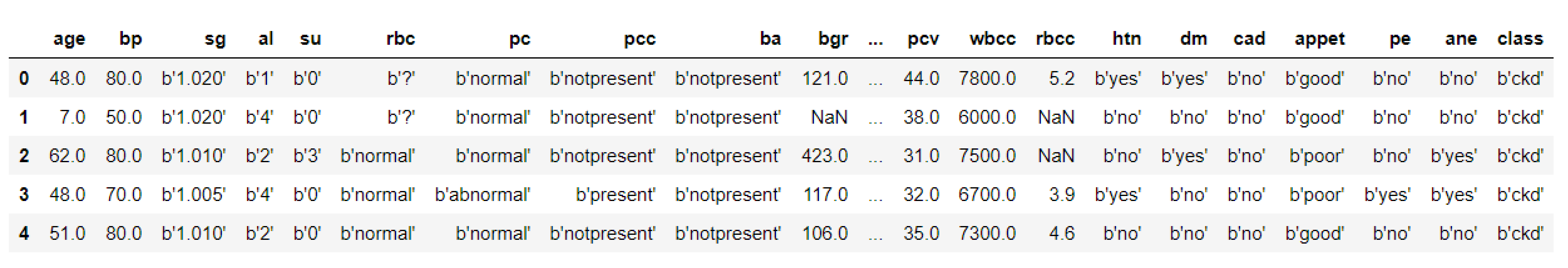
Submitted To: **-** Dr. N. Jayanthi

Methodology

- Reading, Pre-processing and Cleaning Dataset.
- Selecting and using top 5 features.
- K-fold Cross Validations.
- Classification
  - Decision Tree
  - Gaussian Naïve Bayes
  - Random Forest
  - Logistic Regression
  - K-Nearest Neighbours
- Compare Performance.
- Take input and predict diet.

DATASET

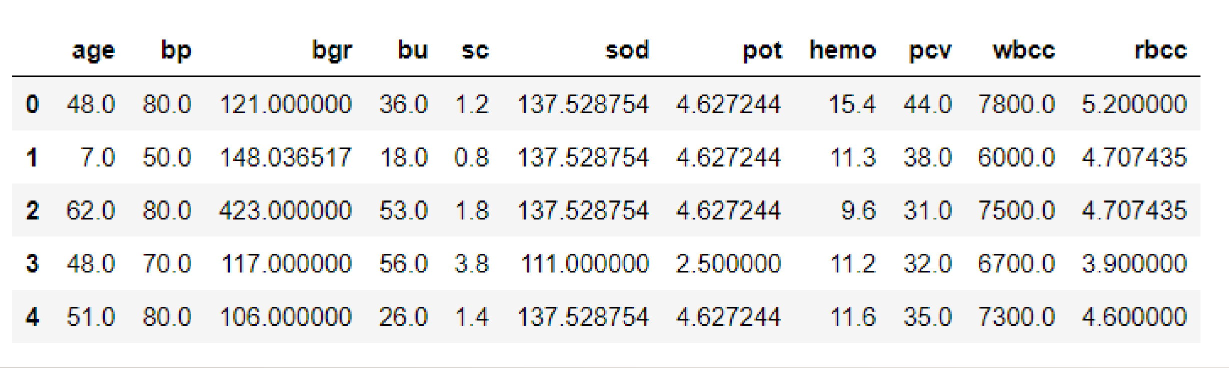
- Source – UCI Machine Learning Repository
- 400 instances, 25 attributes
- 11 numeric columns, 14 nominal columns



	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	...	pcv	wbcc	rbcc	htn	dm	cad	appet	pe	ane	class
0	48.0	80.0	b'1.020'	b'1'	b'0'	b'?'	b'normal'	b'notpresent'	b'notpresent'	121.0	...	44.0	7800.0	5.2	b'yes'	b'yes'	b'no'	b'good'	b'no'	b'no'	b'ckd'
1	7.0	50.0	b'1.020'	b'4'	b'0'	b'?'	b'normal'	b'notpresent'	b'notpresent'	NaN	...	38.0	6000.0	NaN	b'no'	b'no'	b'no'	b'good'	b'no'	b'no'	b'ckd'
2	62.0	80.0	b'1.010'	b'2'	b'3'	b'normal'	b'normal'	b'notpresent'	b'notpresent'	423.0	...	31.0	7500.0	NaN	b'no'	b'yes'	b'no'	b'poor'	b'no'	b'yes'	b'ckd'
3	48.0	70.0	b'1.005'	b'4'	b'0'	b'normal'	b'abnormal'	b'present'	b'notpresent'	117.0	...	32.0	6700.0	3.9	b'yes'	b'no'	b'no'	b'poor'	b'yes'	b'yes'	b'ckd'
4	51.0	80.0	b'1.010'	b'2'	b'0'	b'normal'	b'normal'	b'notpresent'	b'notpresent'	106.0	...	35.0	7300.0	4.6	b'no'	b'no'	b'no'	b'good'	b'no'	b'no'	b'ckd'

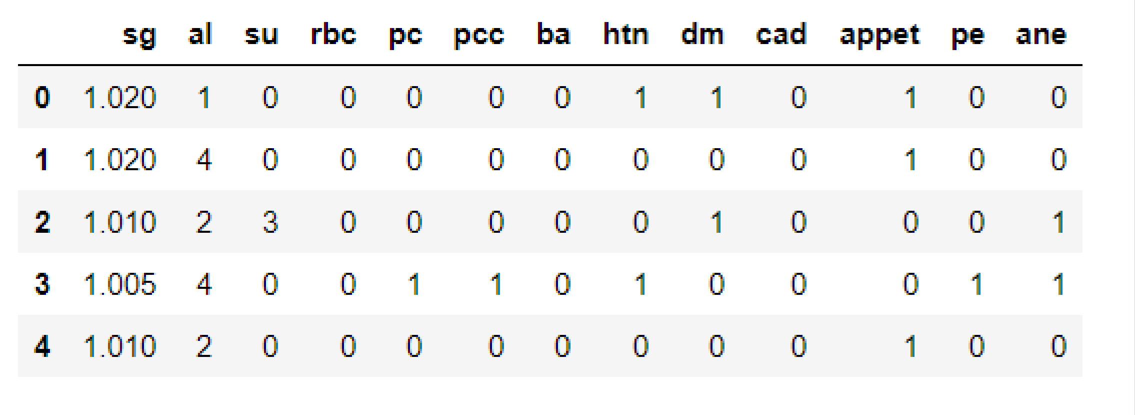
PROCESSING DATASET (1)

Replace NAN values with mean for numeric columns



	age	bp	bgr	bu	sc	sod	pot	hemo	pcv	wbcc	rbcc
0	48.0	80.0	121.000000	36.0	1.2	137.528754	4.627244	15.4	44.0	7800.0	5.200000
1	7.0	50.0	148.036517	18.0	0.8	137.528754	4.627244	11.3	38.0	6000.0	4.707435
2	62.0	80.0	423.000000	53.0	1.8	137.528754	4.627244	9.6	31.0	7500.0	4.707435
3	48.0	70.0	117.000000	56.0	3.8	111.000000	2.500000	11.2	32.0	6700.0	3.900000
4	51.0	80.0	106.000000	26.0	1.4	137.528754	4.627244	11.6	35.0	7300.0	4.600000

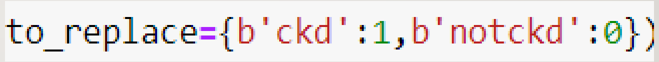
- Replace b'?' with mode values for nominal columns
- Replace all string with integers & floats.



	sg	al	su	rbc	pc	pcc	ba	htn	dm	cad	appet	pe	ane
0	1.020	1	0	0	0	0	0	1	1	0	1	0	0
1	1.020	4	0	0	0	0	0	0	0	0	1	0	0
2	1.010	2	3	0	0	0	0	0	1	0	0	0	1
3	1.005	4	0	0	1	1	0	1	0	0	0	1	1
4	1.010	2	0	0	0	0	0	0	0	0	1	0	0

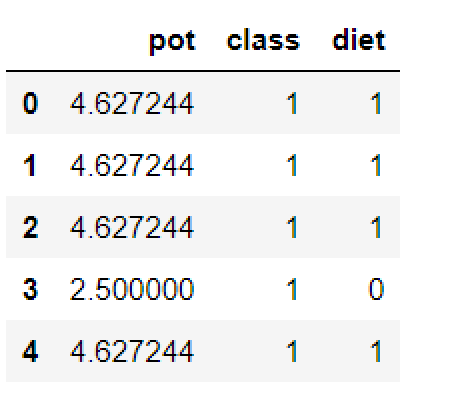
PROCESSING DATASET (11)

- Replace "class" column with integers (0,1)



```
to_replace={b'ckd':1,b'notckd':0})
```

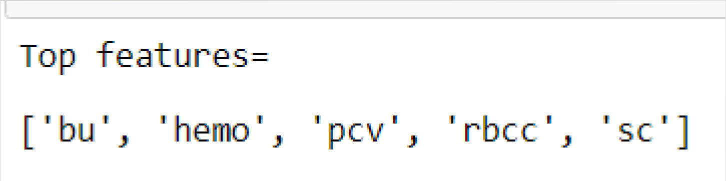
- Create new attribute "diet" based on potassium levels.
  - Pot<3.5 means "Low"
  - 3.5<=Pot<=5.0 means "Safe"
  - 5.0<=Pot<=6.0 means "Caution"
  - Pot>6.0 means "Danger"



	pot	class	diet
0	4.627244	1	1
1	4.627244	1	1
2	4.627244	1	1
3	2.500000	1	0
4	4.627244	1	1

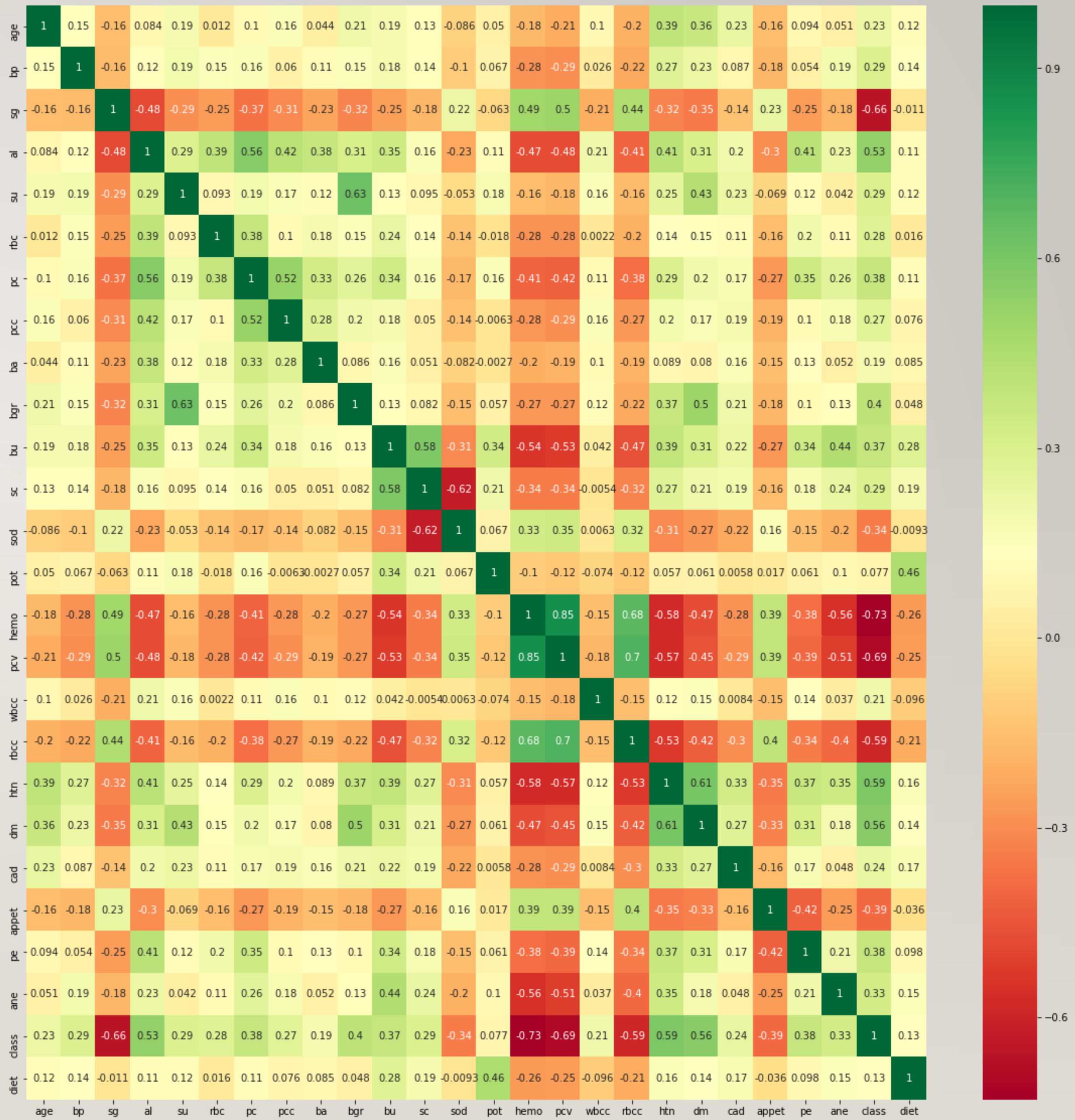
FEATURE SELECTION

- Plot heatmap of correlations.
- Select top 5 variables having high correlation with “diet”.



```
Top features=

['bu', 'hemo', 'pcv', 'rbcc', 'sc']
```

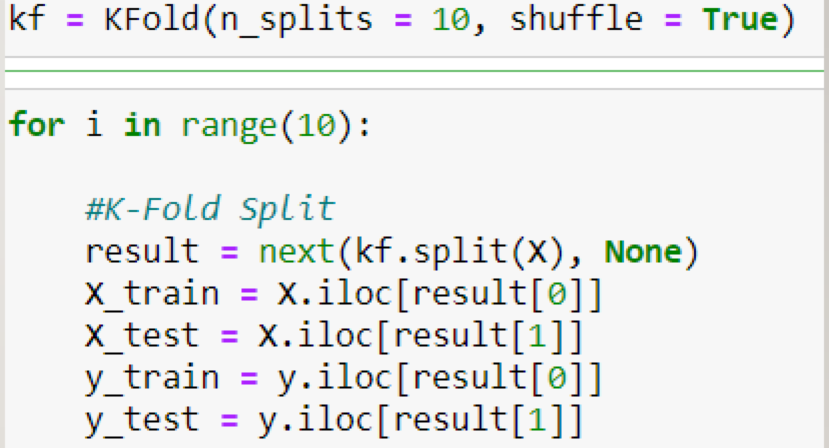
- X has 5 features
- y is “diet” column



	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	bu	sc	sod	pot	hemo	pcv	wbcc	rbcc	htn	dm	cad	appet	pe	ane	class	diet
age	1	0.15	-0.16	0.084	0.19	0.012	0.1	0.16	0.044	0.21	0.19	0.13	-0.086	0.05	-0.18	-0.21	0.1	-0.2	0.39	0.36	0.23	-0.16	0.094	0.051	0.23	0.12
bp	0.15	1	-0.16	0.12	0.19	0.15	0.16	0.06	0.11	0.15	0.18	0.14	-0.1	0.067	-0.28	-0.29	0.026	-0.22	0.27	0.23	0.087	-0.18	0.054	0.19	0.29	0.14
sg	-0.16	-0.16	1	-0.48	-0.29	-0.25	-0.37	-0.31	-0.23	-0.32	-0.25	-0.18	0.22	-0.063	0.49	0.5	-0.21	0.44	-0.32	-0.35	-0.14	0.23	-0.25	-0.18	-0.66	-0.011
al	0.084	0.12	-0.48	1	0.29	0.39	0.56	0.42	0.38	0.31	0.35	0.16	-0.23	0.11	-0.47	-0.48	0.21	-0.41	0.41	0.31	0.2	-0.3	0.41	0.23	0.53	0.11
su	0.19	0.19	-0.29	0.29	1	0.093	0.19	0.17	0.12	0.63	0.13	0.095	-0.053	0.18	-0.16	-0.18	0.16	-0.16	0.25	0.43	0.23	-0.069	0.12	0.042	0.29	0.12
rbc	0.012	0.15	-0.25	0.39	0.093	1	0.38	0.1	0.18	0.15	0.24	0.14	-0.14	-0.018	-0.28	-0.28	0.0022	-0.2	0.14	0.15	0.11	-0.16	0.2	0.11	0.28	0.016
pc	0.1	0.16	-0.37	0.56	0.19	0.38	1	0.52	0.33	0.26	0.34	0.16	-0.17	0.16	-0.41	-0.42	0.11	-0.38	0.29	0.2	0.17	-0.27	0.35	0.26	0.38	0.11
pcc	0.16	0.06	-0.31	0.42	0.17	0.1	0.52	1	0.28	0.2	0.18	0.05	-0.14	-0.0063	-0.28	-0.29	0.16	-0.27	0.2	0.17	0.19	-0.19	0.1	0.18	0.27	0.076
ba	0.044	0.11	-0.23	0.38	0.12	0.18	0.33	0.28	1	0.086	0.16	0.051	-0.082	-0.0027	-0.2	-0.19	0.1	-0.19	0.089	0.08	0.16	-0.15	0.13	0.052	0.19	0.085
bgr	0.21	0.15	-0.32	0.31	0.63	0.15	0.26	0.2	0.086	1	0.13	0.082	-0.15	0.057	-0.27	-0.27	0.12	-0.22	0.37	0.5	0.21	-0.18	0.1	0.13	0.4	0.048
bu	0.19	0.18	-0.25	0.35	0.13	0.24	0.34	0.18	0.16	0.13	1	0.58	-0.31	0.34	-0.54	-0.53	0.042	-0.47	0.39	0.31	0.22	-0.27	0.34	0.44	0.37	0.28
sc	0.13	0.14	-0.18	0.16	0.095	0.14	0.16	0.05	0.051	0.082	0.58	1	-0.62	0.21	-0.34	-0.34	-0.0054	-0.32	0.27	0.21	0.19	-0.16	0.18	0.24	0.29	0.19
sod	-0.086	-0.1	0.22	-0.23	-0.053	-0.14	-0.17	-0.14	-0.082	-0.15	-0.31	-0.62	1	0.067	0.33	0.35	0.0063	0.32	-0.31	-0.27	-0.22	0.16	-0.15	-0.2	-0.34	-0.0093
pot	0.05	0.067	-0.063	0.11	0.18	-0.018	0.16	-0.0063	-0.0027	0.057	0.34	0.21	0.067	1	-0.1	-0.12	-0.074	-0.12	0.057	0.061	0.0058	0.017	0.061	0.1	0.077	0.46
hemo	-0.18	-0.28	0.49	-0.47	-0.16	-0.28	-0.41	-0.28	-0.2	-0.27	-0.54	-0.34	0.33	-0.1	1	0.85	-0.15	0.68	-0.58	-0.47	-0.28	0.39	-0.38	-0.56	-0.73	-0.26
pcv	-0.21	-0.29	0.5	-0.48	-0.18	-0.28	-0.42	-0.29	-0.19	-0.27	-0.53	-0.34	0.35	-0.12	0.85	1	-0.18	0.7	-0.57	-0.45	-0.29	0.39	-0.39	-0.51	-0.69	-0.25
wbcc	0.1	0.026	-0.21	0.21	0.16	0.0022	0.11	0.16	0.1	0.12	0.042	-0.0054	0.0063	-0.074	-0.15	-0.18	1	-0.15	0.12	0.15	0.0084	-0.15	0.14	0.037	0.21	-0.096
rbcc	-0.2	-0.22	0.44	-0.41	-0.16	-0.2	-0.38	-0.27	-0.19	-0.22	-0.47	-0.32	0.32	-0.12	0.68	0.7	-0.15	1	-0.53	-0.42	-0.3	0.4	-0.34	-0.4	-0.59	-0.21
htn	0.39	0.27	-0.32	0.41	0.25	0.14	0.29	0.2	0.089	0.37	0.39	0.27	-0.31	0.057	-0.58	-0.57	0.12	-0.53	1	0.61	0.33	-0.35	0.37	0.35	0.59	0.16
dm	0.36	0.23	-0.35	0.31	0.43	0.15	0.2	0.17	0.08	0.5	0.31	0.21	-0.27	0.061	-0.47	-0.45	0.15	-0.42	0.61	1	0.27	-0.33	0.31	0.18	0.56	0.14
cad	0.23	0.087	-0.14	0.2	0.23	0.11	0.17	0.19	0.16	0.21	0.22	0.19	-0.22	0.0058	-0.28	-0.29	0.0084	-0.3	0.33	0.27	1	-0.16	0.17	0.048	0.24	0.17
appet	-0.16	-0.18	0.23	-0.3	-0.069	-0.16	-0.27	-0.19	-0.15	-0.18	-0.27	-0.16	0.16	0.017	0.39	0.39	-0.15	0.4	-0.35	-0.33	-0.16	1	-0.42	-0.25	-0.39	-0.036
pe	0.094	0.054	-0.25	0.41	0.12	0.2	0.35	0.1	0.13	0.1	0.34	0.18	-0.15	0.061	-0.38	-0.39	0.14	-0.34	0.37	0.31	0.17	-0.42	1	0.21	0.38	0.098
ane	0.051	0.19	-0.18	0.23	0.042	0.11	0.26	0.18	0.052	0.13	0.44	0.24	-0.2	0.1	-0.56	-0.51	0.037	-0.4	0.35	0.18	0.048	-0.25	0.21	1	0.33	0.15
class	0.23	0.29	-0.66	0.53	0.29	0.28	0.38	0.27	0.19	0.4	0.37	0.29	-0.34	0.077	-0.73	-0.69	0.21	-0.59	0.59	0.56	0.24	-0.39	0.38	0.33	1	0.13
diet	0.12	0.14	-0.011	0.11	0.12	0.016	0.11	0.076	0.085	0.048	0.28	0.19	-0.0093	0.46	-0.26	-0.25	-0.096	-0.21	0.16	0.14	0.17	-0.036	0.098	0.15	0.13	1

K-FOLD CROSS VALIDATION

- K=10
- Shuffle dataset randomly.
- Splits dataset into 10 groups
- Takes one group as test dataset
- Takes the remaining groups as train dataset
- Repeat till all groups used as test dataset
- Mean of evaluation scores for each fold is taken.



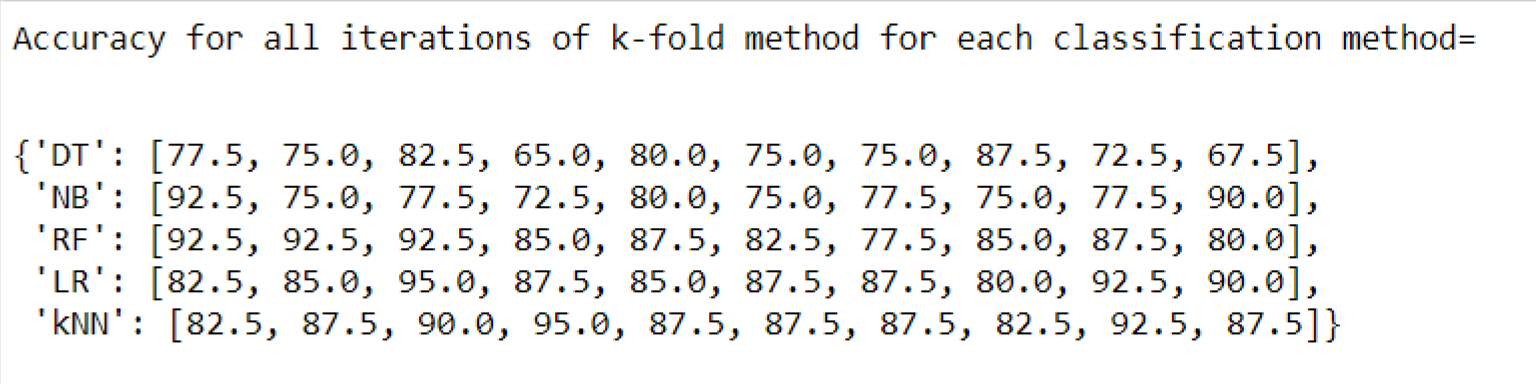
```
kf = KFold(n_splits = 10, shuffle = True)
```

```
for i in range(10):

    #K-Fold Split
    result = next(kf.split(X), None)
    X_train = X.iloc[result[0]]
    X_test = X.iloc[result[1]]
    y_train = y.iloc[result[0]]
    y_test = y.iloc[result[1]]
```

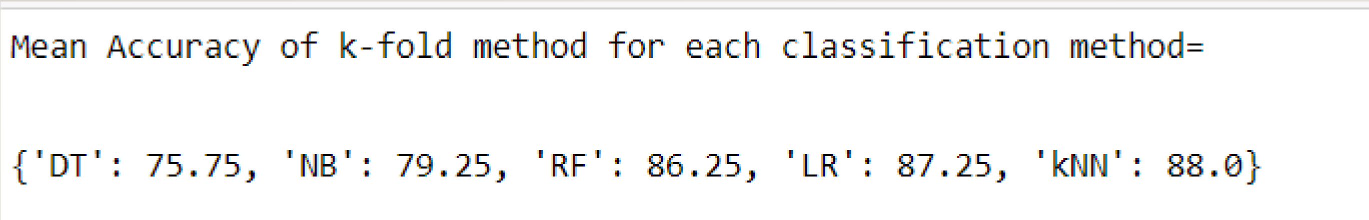
CLASSIFICATION MODELS

- Decision Tree
  - Tree = tree.DecisionTreeClassifier(criterion='entropy',random_state=0)
- Naïve Bayes
  - model=GaussianNB ()
- Random Forest
  - Model = RandomForestClassifier(random_state=0,n_estimators=50,criterion='entropy’)
- Logistic Regression
  - model=LogisticRegression (solver='lbfgs', random_state=0, multi_class='multinomial', max_iter=5000)
- K-Nearest Neighbours
  - classifier = KNeighborsClassifier(n_neighbors=15)

RESULTS (I)



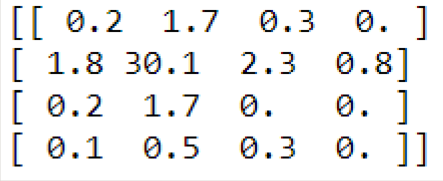
```
Accuracy for all iterations of k-fold method for each classification method=


{'DT': [77.5, 75.0, 82.5, 65.0, 80.0, 75.0, 75.0, 87.5, 72.5, 67.5],
 'NB': [92.5, 75.0, 77.5, 72.5, 80.0, 75.0, 77.5, 75.0, 77.5, 90.0],
 'RF': [92.5, 92.5, 92.5, 85.0, 87.5, 82.5, 77.5, 85.0, 87.5, 80.0],
 'LR': [82.5, 85.0, 95.0, 87.5, 85.0, 87.5, 87.5, 80.0, 92.5, 90.0],
 'kNN': [82.5, 87.5, 90.0, 95.0, 87.5, 87.5, 87.5, 82.5, 92.5, 87.5]}
```



```
Mean Accuracy of k-fold method for each classification method=


{'DT': 75.75, 'NB': 79.25, 'RF': 86.25, 'LR': 87.25, 'kNN': 88.0}
```

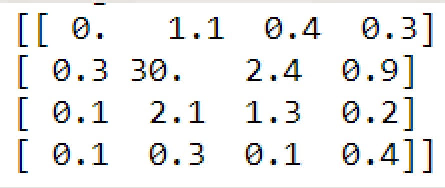
RESULTS (11)

Decision Tree



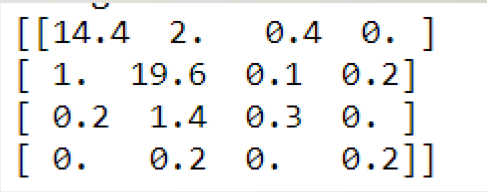
```
[[ 0.2  1.7  0.3  0. ]
 [ 1.8 30.1  2.3  0.8]
 [ 0.2  1.7  0.   0. ]
 [ 0.1  0.5  0.3  0. ]]
```

Naïve Bayes



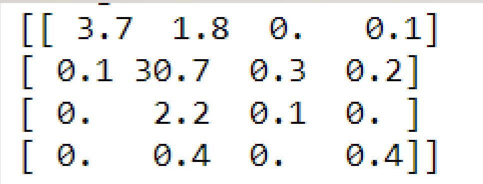
```
[[ 0.   1.1  0.4  0.3]
 [ 0.3 30.   2.4  0.9]
 [ 0.1  2.1  1.3  0.2]
 [ 0.1  0.3  0.1  0.4]]
```

Random Forest



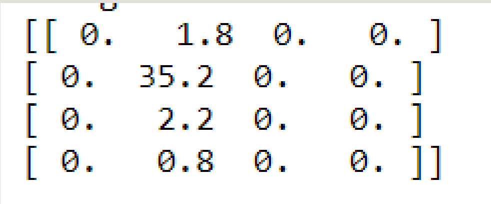
```
[[14.4  2.   0.4  0. ]
 [ 1.  19.6  0.1  0.2]
 [ 0.2  1.4  0.3  0. ]
 [ 0.   0.2  0.   0.2]]
```

Logistic Regression



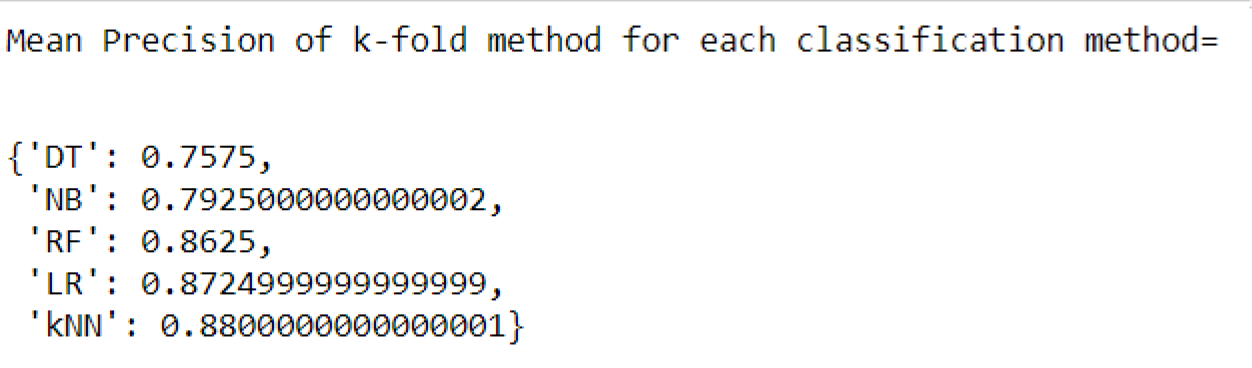
```
[[ 3.7  1.8  0.   0.1]
 [ 0.1 30.7  0.3  0.2]
 [ 0.   2.2  0.1  0. ]
 [ 0.   0.4  0.   0.4]]
```

k-Nearest Neighbours

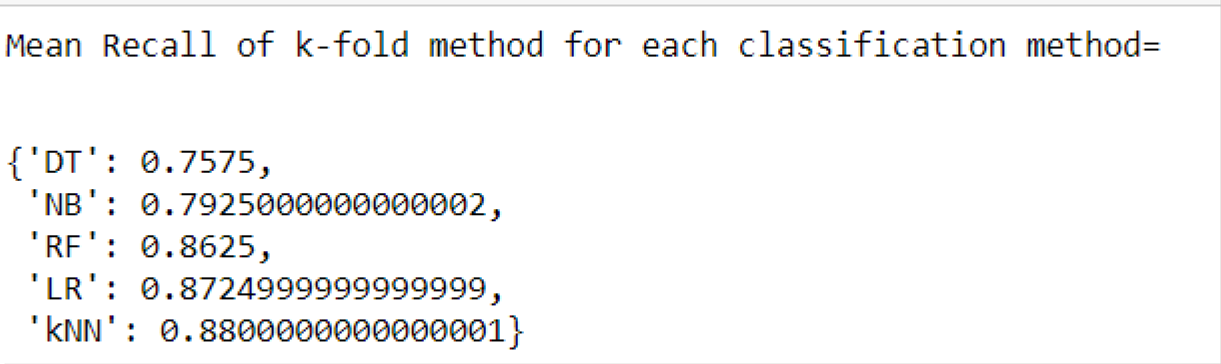


```
[[ 0.   1.8  0.   0. ]
 [ 0.  35.2  0.   0. ]
 [ 0.   2.2  0.   0. ]
 [ 0.   0.8  0.   0. ]]
```

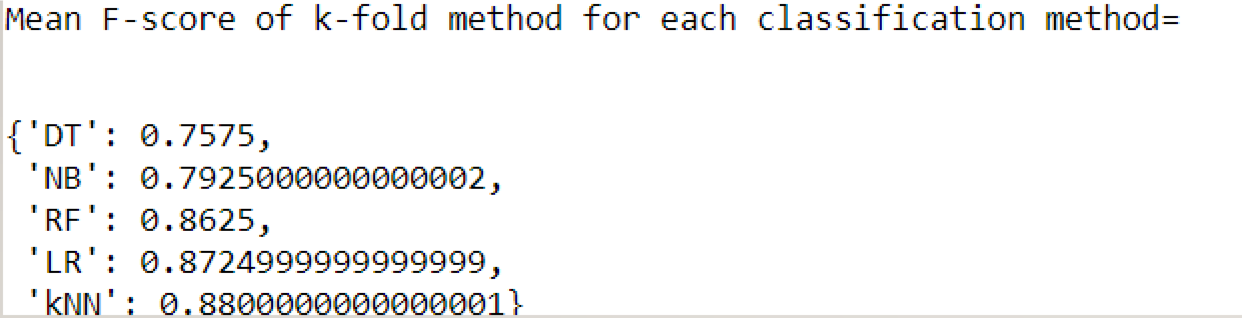
RESULTS (III)



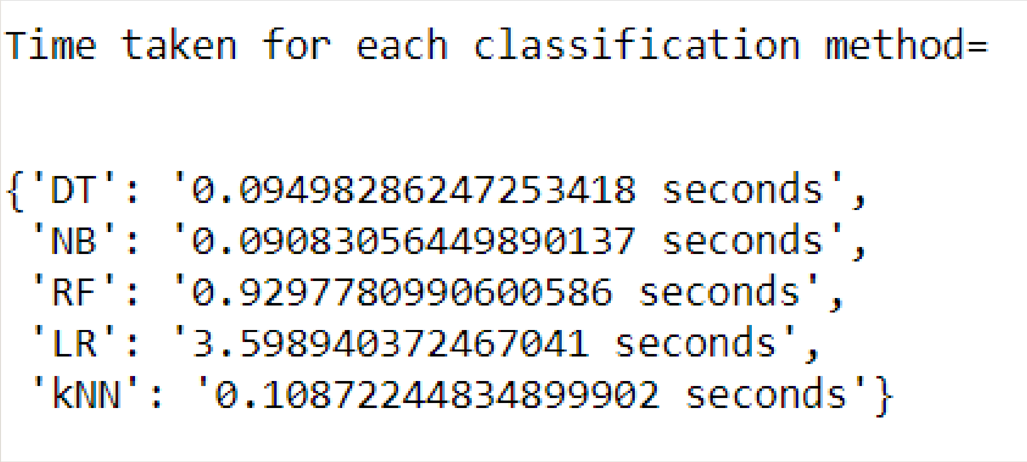
```
Mean Precision of k-fold method for each classification method=


{'DT': 0.7575,
 'NB': 0.7925000000000002,
 'RF': 0.8625,
 'LR': 0.8724999999999999,
 'kNN': 0.8800000000000001}
```



```
Mean Recall of k-fold method for each classification method=


{'DT': 0.7575,
 'NB': 0.7925000000000002,
 'RF': 0.8625,
 'LR': 0.8724999999999999,
 'kNN': 0.8800000000000001}
```

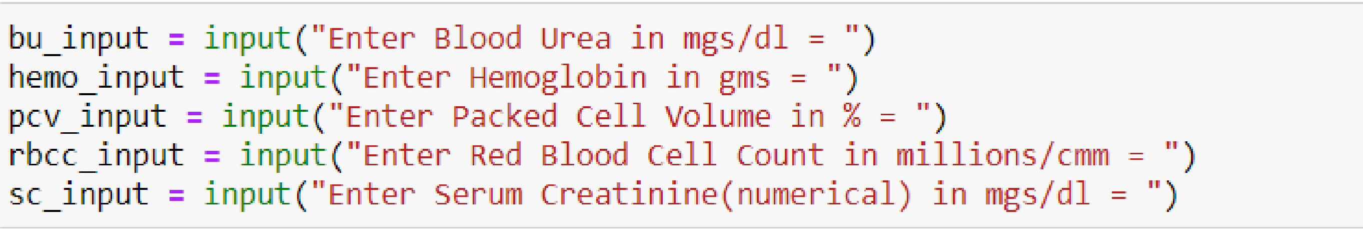


```
Mean F-score of k-fold method for each classification method=


{'DT': 0.7575,
 'NB': 0.7925000000000002,
 'RF': 0.8625,
 'LR': 0.8724999999999999,
 'kNN': 0.8800000000000001}
```

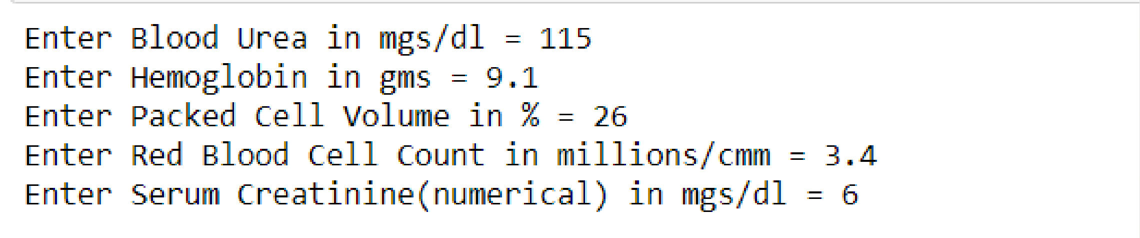


```
Time taken for each classification method=


{'DT': '0.09498286247253418 seconds',
 'NB': '0.09083056449890137 seconds',
 'RF': '0.9297780990600586 seconds',
 'LR': '3.598940372467041 seconds',
 'kNN': '0.10872244834899902 seconds'}
```

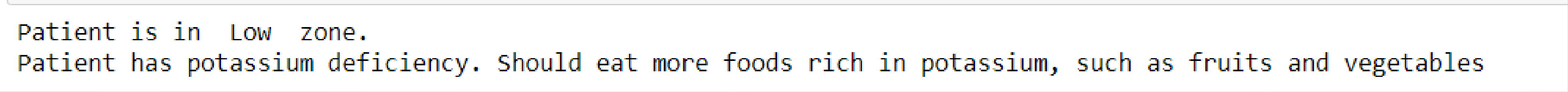
Taking Input and making prediction



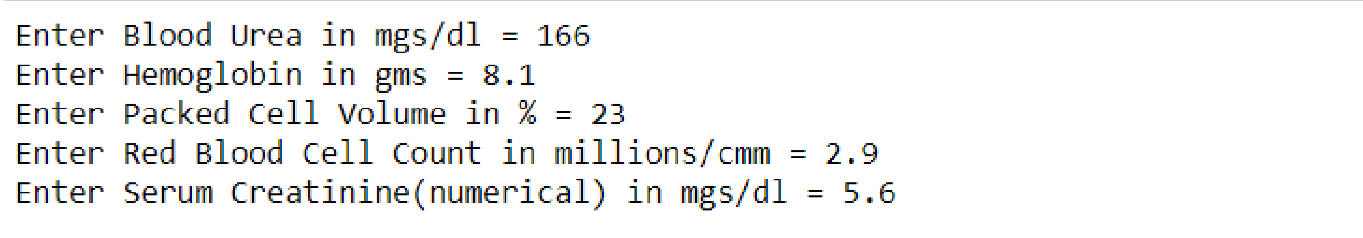
```
bu_input = input("Enter Blood Urea in mgs/dl = ")
hemo_input = input("Enter Hemoglobin in gms = ")
pcv_input = input("Enter Packed Cell Volume in % = ")
rbcc_input = input("Enter Red Blood Cell Count in millions/cmm = ")
sc_input = input("Enter Serum Creatinine(numerical) in mgs/dl = ")
```



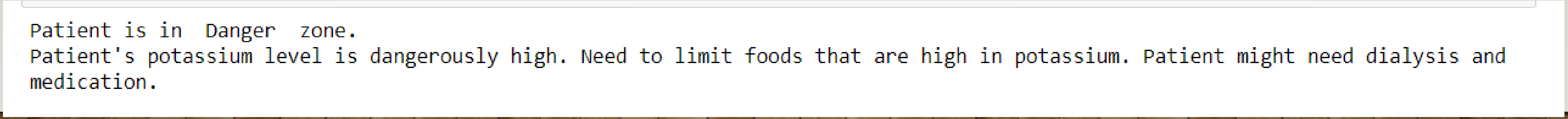
```
Enter Blood Urea in mgs/dl = 115
Enter Hemoglobin in gms = 9.1
Enter Packed Cell Volume in % = 26
Enter Red Blood Cell Count in millions/cmm = 3.4
Enter Serum Creatinine(numerical) in mgs/dl = 6
```



```
Patient is in  Low  zone.
Patient has potassium deficiency. Should eat more foods rich in potassium, such as fruits and vegetables
```



```
Enter Blood Urea in mgs/dl = 166
Enter Hemoglobin in gms = 8.1
Enter Packed Cell Volume in % = 23
Enter Red Blood Cell Count in millions/cmm = 2.9
Enter Serum Creatinine(numerical) in mgs/dl = 5.6
```



```
Patient is in  Danger  zone.
Patient's potassium level is dangerously high. Need to limit foods that are high in potassium. Patient might need dialysis and
medication.
```

CONCLUSION

K-nearest neighbours & Random Forest Classifier are the best algorithms for this problem of predicting diet for CKD patients.

- They have high accuracy
- Have small training time.

THANK YOU