CHRONIC KIDNEY DISEASE PREDICTION

Shipra Sonal
Shipra.sonal12@gmail.com

PROBLEM STATEMENT

- Build a model that can help to determine if a patient is suffering from kidney chronic disease or not
- Identify the factors causing chronic kidney disease.

Data Set Characteristics:	**Multivariate**	**Number of Instances:**	**400**	**Area:**	**N/A**
Attribute Characteristics:	Real	Number of Attributes:	25	Date Donated	2015-07-03
Associated Tasks:	Classification	Missing Values?	Yes		

DATA UNDERSTANDING

- Number of attributes: 25 (class included)
- 14 Nominal & 11 Numeric
- Missing value represented by '?'
- Few numeric fields have whitespaces, thus required cleaning

DETAILED STEPS WITH LOGICS

IMPORTING DATA

- Loading data in python

```
[4]:
# Import Data
input_filename = 'kidneyChronic.csv'
input_filepath = os.getcwd()+"/"+input_filename
# Replacing
Base_df = pd.read_csv(input_filepath,index_col=False,na_values=['?'])
Base_df
```

t[4]:

	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	1.020	1.0	0.0	NaN	normal	notpresent	notpresent	121.0	...	44	7800	5.2	yes	yes	no	good	no	no	ckd
1	7.0	50.0	1.020	4.0	0.0	NaN	normal	notpresent	notpresent	NaN	...	38	6000	NaN	no	no	no	good	no	no	ckd
2	62.0	80.0	1.010	2.0	3.0	normal	normal	notpresent	notpresent	423.0	...	31	7500	NaN	no	yes	no	poor	no	yes	ckd
3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	117.0	...	32	6700	3.9	yes	no	no	poor	yes	yes	ckd
4	51.0	80.0	1.010	2.0	0.0	normal	normal	notpresent	notpresent	106.0	...	35	7300	4.6	no	no	no	good	no	no	ckd
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

DATA CLEANING

- Replaced any escaped, literal and whitespaces character as per data analysis done in later portion of code

```
# handle escaped , literal and whitespaces as per data analysis done below
Base_df.replace(to_replace=[r"\\t|\\n|\\r", "\t|\n|\r",'(^\s+|\s+$)','\t'], value=['','','',''], regex=True, inplac
Base_df.replace("?",np.nan,inplace =True)

```

PREREQUISITE FOR EASY DATA HANDLING

Creating the numerical verison of class variable for easy analysis

6]:
```
1 Base_df['class_num'] =  np.where(Base_df["class"]=='ckd',1,0)
```

Creating a dictionary for field names and their descriptions/elaborated names for easy understanding

7]:
```
1 # Creating dict for field names and elaboration
2 Attributes_Description = { 'age' :'age' , 'bp' :'blood pressure' , 'sg' :'specific gravity' , 'al' :'albumin' , 'su
3
4
5 Attributes_Description
```

7]:
```
{'age': 'age',
 'bp': 'blood pressure',
 'sg': 'specific gravity',
 'al': 'albumin',
 'su': 'sugar',
 'rbc': 'red blood cells',
 'pc': 'pus cell',
 'pcc': 'pus cell clumps',
 'ba': 'bacteria',
 'bgr': 'blood glucose random',
```

EXPLORATORY DATA ANALYSIS

- Checking the missing value percentage for each attribute

```
]:  1 for i in (Base_df.columns):
    2     # count number of rows with missing values
    3     n_miss = Base_df[[i]].isnull().sum()
    4     perc = n_miss / Base_df.shape[0] * 100
    5     print('> %s , Missing: %d (%.1f%%)' % (i, n_miss, perc))
```

```
> age , Missing: 9 (2.2%)
> bp , Missing: 12 (3.0%)
> sg , Missing: 47 (11.8%)
> al , Missing: 46 (11.5%)
> su , Missing: 49 (12.2%)
> rbc , Missing: 152 (38.0%)
> pc , Missing: 65 (16.2%)
> pcc , Missing: 4 (1.0%)
> ba , Missing: 4 (1.0%)
> bgr , Missing: 44 (11.0%)
> bu , Missing: 19 (4.8%)
> sc , Missing: 17 (4.2%)
> sod , Missing: 87 (21.8%)
> pot , Missing: 88 (22.0%)
> hemo , Missing: 52 (13.0%)
> pcv , Missing: 71 (17.8%)
```

Missing value analysis is important to understand the nature of data.

Incase, any attribute has high missing values, It needs to be dropped

We will perform the missing value imputation as per the analysis

EXPLORATORY DATA ANALYSIS

- Checking the data types and value counts for each fields

```
1  Base_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 26 columns):
 #    Column      Non-Null Count  Dtype
---   ------      --------------  -----
 0    age         391 non-null    float64
 1    bp          388 non-null    float64
 2    sg          353 non-null    float64
 3    al          354 non-null    float64
 4    su          351 non-null    float64
 5    rbc         248 non-null    object
 6    pc          335 non-null    object
 7    pcc         396 non-null    object
 8    ba          396 non-null    object
 9    bgr         356 non-null    float64
 10   bu          381 non-null    float64
 11   sc          383 non-null    float64
 12   sod         313 non-null    float64
```

We will use this analysis to match the data type with nature of variable

EXPLORATORY DATA ANALYSIS

- EDA for categorical columns

```
[9]: 1 Base_df.describe(include=['O'])
```

[9]:

	rbc	pc	pcc	ba	pcv	wbcc	rbcc	htn	dm	cad	appet	pe	ane	class
count	248	335	396	396	329	294	269	398	398	398	399	399	399	400
unique	2	2	2	2	42	89	45	2	2	2	2	2	2	2
top	normal	normal	notpresent	notpresent	52	9800	5.2	no	no	no	good	no	no	ckd
freq	201	259	354	374	21	11	18	251	261	364	317	323	339	250

EXPLORATORY DATA ANALYSIS

- Analysed Distinct values and counts in categorical columns

```
]:  1  # Printing distinct values of Nominal fields
    2  for i in nominal_col:
    3      print('-'*100)
    4      print(Attributes_Description[i])
    5      print('-'*100)
    6      print(Base_df[i].value_counts())
```

```
----------------------------------------------------------------------------------------------------
specific gravity
----------------------------------------------------------------------------------------------------
1.020    106
1.010     84
1.025     81
1.015     75
1.005      7
Name: sg, dtype: int64
----------------------------------------------------------------------------------------------------
albumin
----------------------------------------------------------------------------------------------------
0.0    199
1.0     44
2.0     43
3.0     43
4.0     24
5.0      1
Name: al, dtype: int64
```

EXPLORATORY DATA ANALYSIS

- Analysed event rate for each category in detail

In [11]:

```
for i in nominal_col:
    temp = Base_df[[i, Target]].fillna('NA').groupby([i], as_index=False).sum().sort_values(by=Target, ascending=Fa
    temp1 = pd.DataFrame(Base_df[i].fillna('NA').value_counts() )
    temp1.columns = [i+'_value_count']
    temp.rename(columns = {'class_num': i+'_target_count'},inplace = True)
    temp = temp.merge(temp1, left_on=  temp[i] ,right_on=temp1.index).drop(columns='key_0')
    temp[i+'_target_%'] = temp[i+'_target_count']/temp[i+'_value_count']
    temp = temp[[i,i+'_value_count',i+'_target_count',i+'_target_%']]
    print('-'*100)
    print(Attributes_Description[i])
    print('-'*100)
    print(temp)
    print('*'*100)
    print(' '*100)
```

```
----------------------------------------------------------------------------------------------------
specific gravity
----------------------------------------------------------------------------------------------------
      sg  sg_value_count  sg_target_count  sg_target_%
0   1.01              84               84     1.000000
1  1.015              75               75     1.000000
2     NA              47               42     0.893617
3   1.02             106               31     0.292453
4  1.025              81               11     0.135802
5  1.005               7                7     1.000000
****************************************************************************************************

----------------------------------------------------------------------------------------------------
albumin
----------------------------------------------------------------------------------------------------
```

MISSING VALUE IMPUTATION FOR CATEGORICAL VARIABLES

- Observation from event rate
  - Due to different pattern for event rate in missing values, following attributes were treated with constant value for missing value imputation
    - al, su, rbc, pc, sg
  - Remaining variables were treated by mode value treatment for missing value imputation
    - pcc, ba, dm, cad, appet, pe

  We referred following blog for the topic Link

EXPLORATORY DATA ANALYSIS

- EDA for numerical columns

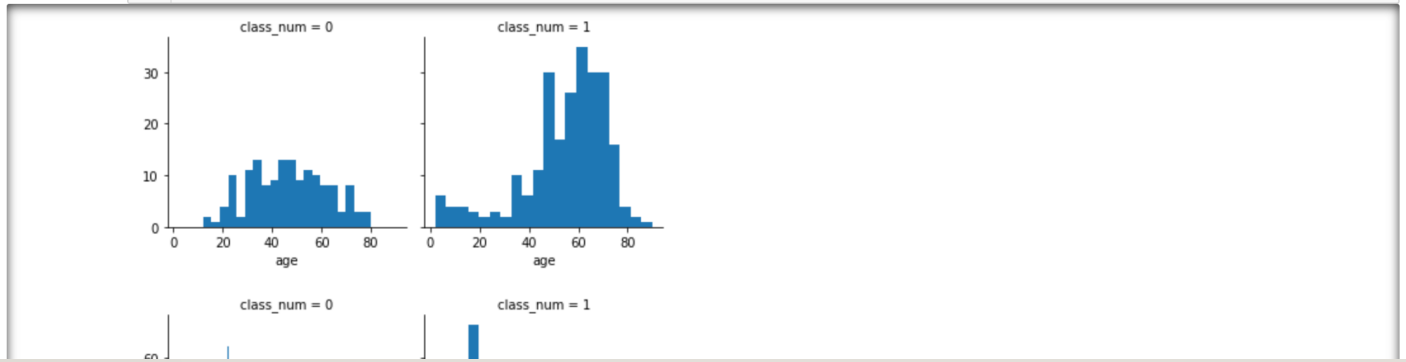
```
1 Base_df.describe()
```

	age	bp	sg	al	su	bgr	bu	sc	sod	pot	hemo	class_num
count	391.000000	388.000000	353.000000	354.000000	351.000000	356.000000	381.000000	383.000000	313.000000	312.000000	348.000000	400.000000
mean	51.483376	76.469072	1.017408	1.016949	0.450142	148.036517	57.425722	3.072454	137.528754	4.627244	12.526437	0.625000
std	17.169714	13.683637	0.005717	1.352679	1.099191	79.281714	50.503006	5.741126	10.408752	3.193904	2.912587	0.484729
min	2.000000	50.000000	1.005000	0.000000	0.000000	22.000000	1.500000	0.400000	4.500000	2.500000	3.100000	0.000000
25%	42.000000	70.000000	1.010000	0.000000	0.000000	99.000000	27.000000	0.900000	135.000000	3.800000	10.300000	0.000000
50%	55.000000	80.000000	1.020000	0.000000	0.000000	121.000000	42.000000	1.300000	138.000000	4.400000	12.650000	1.000000
75%	64.500000	80.000000	1.020000	2.000000	0.000000	163.000000	66.000000	2.800000	142.000000	4.900000	15.000000	1.000000
max	90.000000	180.000000	1.025000	5.000000	5.000000	490.000000	391.000000	76.000000	163.000000	47.000000	17.800000	1.000000

EXPLORATORY DATA ANALYSIS

- Visualising the numerical attributes for better understanding of data

In [16]:
```
for i in numerical_col:
    g = sns.FacetGrid(Base_df, col=Target)
    g.map(plt.hist, i, bins=20)
```



EXPLORATORY DATA ANALYSIS

- Analysed numerical data by binning values

In [19]:

```
for i in numerical_col:
    Base_df['Band'] = pd.cut(Base_df[i].round(-1), 11)
    temp = Base_df[['Band', Target]].groupby(['Band'], as_index=False).agg(
        count=(Target, 'count'), sum=(Target, sum),mean = (Target, np.average)).sort_values(by='Band', ascending=Tr

    print('-'*100)
    print(Attributes_Description[i])
    print('-'*100)
    print(temp)
    print('*'*100)
    print(' '*100)
Base_df.drop(['Band'], axis=1,inplace = True)
```

```
----------------------------------------------------------------------------------------------------
age
----------------------------------------------------------------------------------------------------
                Band  count  sum      mean
0     (-0.09, 8.182]      5    5  1.000000
1    (8.182, 16.364]      9    8  0.888889
2   (16.364, 24.545]     23    7  0.304348
3   (24.545, 32.727]     32   10  0.312500
4   (32.727, 40.909]     55   21  0.381818
5   (40.909, 49.091]      0    0       NaN
6   (49.091, 57.273]     71   47  0.661972
7   (57.273, 65.455]    115   82  0.713043
8   (65.455, 73.636]     61   47  0.770492
9   (73.636, 81.818]     19   14  0.736842
10   (81.818, 90.0]      1    1  1.000000
****************************************************************************************************
```

MISSING VALUE IMPUTATION IN NUMERICAL ATTRIBUTES

- We used mean imputation

Mean imputation in numerical fields

[22]:

```
1  # To calculate mean use imputer class
2  from sklearn.impute import SimpleImputer
3  imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
4
5  imputer = imputer.fit(train[numerical_col])
6  train[numerical_col] = imputer.transform(train[numerical_col])
7  test[numerical_col] = imputer.transform(test[numerical_col])
8
```

```
/usr/local/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:6: SettingWithCc
```

BINNING OF AGE

- Based on above EDAs, we have performed binning operation on Age

Binning Age variable

```
]:
1  # Binnig Age Variables
2  cut_bins = [0,10,40,60,1000]
3  train['Age_Band'] = pd.cut(train.age, bins=cut_bins)
4  test['Age_Band'] = pd.cut(test.age, bins=cut_bins)
5  Base_df['Age_Band'] = pd.cut(Base_df.age, bins=cut_bins)
6
7  i = 'age'
8
9  temp = train[['Age_Band', Target]].groupby(['Age_Band'], as_index=False).agg(
10     count=(Target, 'count'), sum=(Target, sum),mean = (Target, np.average)).sort_values(by='Age_Band', ascending=Tr
11 temp
```

```
/usr/local/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:3: SettingWithCopyWarning:
```

TRAIN TEST SPLIT

- Splitting data into train and test. Used 70-30 ratio

Train Test Split

21]:

```
from sklearn import model_selection
train, test = model_selection.train_test_split(Base_df,
                     test_size=0.3)

## print info
print("X_train shape:", train.drop(Target_cols,axis=1).shape, "\nX_test shape:", test.drop(Target_cols,axis=1).shap
print("y_train mean:", round(np.mean(train[Target_cols]),2), "\ny_test mean:", round(np.mean(test[Target_cols]),2))
print(train.shape[1], "features:", train.drop(Target_cols,axis=1).columns.to_list())
```

```
X_train shape: (280, 24)
X_test shape: (120, 24)
y_train mean: class_num    0.63
dtype: float64
y_test mean: class_num    0.62
dtype: float64
26 features: ['age', 'bp', 'sg', 'al', 'su', 'rbc', 'pc', 'pcc', 'ba', 'bgr', 'bu', 'sc', 'sod', 'pot', 'hemo', 'pc
v', 'wbcc', 'rbcc', 'htn', 'dm', 'cad', 'appet', 'pe', 'ane']
```

FEATURE CREATION – ONE HOT ENCODING

- Performed one hot encoding for categorical variables

```
from sklearn import preprocessing


enc = preprocessing.OneHotEncoder()

# 2. FIT
enc.fit(train[nominal_col].append(test[nominal_col]))

# 3. Transform
train[enc.get_feature_names(nominal_col)] = enc.transform(train[nominal_col]).toarray()
train.drop(nominal_col, axis=1,inplace = True)

test[enc.get_feature_names(nominal_col)] = enc.transform(test[nominal_col]).toarray()
test.drop(nominal_col, axis=1,inplace = True)

```

FEATURE CREATION – NORMALIZING NUMERICAL ATTRIBUTES

- Normalising all numerical features between 0-1 for feeding them into model

Normalising all features between 0-1 for feeding them into model

In [29]:

```
from sklearn import preprocessing
scaler = preprocessing.MinMaxScaler(feature_range=(0,1))
scaler = scaler.fit((train.drop(Target_cols, axis=1)))

X_train = scaler.transform(train.drop(Target_cols, axis=1))
train_scaled= pd.DataFrame(X_train, columns=train.drop(Target_cols, axis=1).columns, index=train.index)
train_scaled["Class_num"] = train.class_num
train_scaled.head()


X_test = scaler.transform(test.drop(Target_cols, axis=1))
test_scaled= pd.DataFrame(X_test, columns=test.drop(Target_cols, axis=1).columns, index=test.index)
test_scaled["Class_num"] = test.class_num
test_scaled.head()
```

Out[29]:

	bp	bgr	bu	sc	sod	pot	hemo	pcv	wbcc	rbcc	...	appet_poor	pe_no	pe_yes	ane_no	ane_yes	Age
36	0.153846	0.149573	0.085803	0.017219	0.883162	0.341463	0.493151	0.511111	0.316249	0.446059	...	0.0	1.0	0.0	1.0	0.0	
350	0.153846	0.134615	0.057722	0.006623	0.945017	0.560976	0.890411	0.755556	0.381443	0.406780	...	0.0	1.0	0.0	1.0	0.0	
160	0.076923	0.269231	0.117005	0.021192	0.979381	0.414634	0.534247	0.577778	0.371134	0.050847	...	1.0	0.0	1.0	1.0	0.0	
221	0.153846	0.482906	0.088924	0.015894	0.917526	0.682927	0.637629	0.661546	0.316249	0.446059	...	0.0	1.0	0.0	1.0	0.0	
4	0.230769	0.179487	0.076443	0.011921	0.913480	0.447934	0.582192	0.577778	0.262887	0.423729	...	0.0	1.0	0.0	1.0	0.0	

5 rows × 56 columns

In [32]:

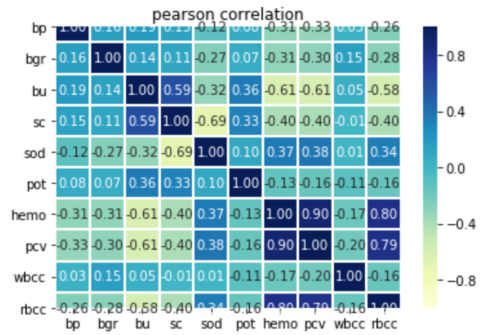
```
## Analysing correlation between variables for dropping highly correlated features
```

FEATURE SELECTION

- Analysing correlation between variables for dropping hughly correlated features

In [30]:
```
corr_matrix = Base_df[numerical_col].copy()
for col in corr_matrix.columns:
    if corr_matrix[col].dtype == "O":
        corr_matrix[col] = corr_matrix[col].factorize(sort=True)[0]
corr_matrix = corr_matrix.corr(method="pearson")
sns.heatmap(corr_matrix, vmin=-1., vmax=1., annot=True, fmt='.2f', cmap="YlGnBu", cbar=True, linewidths=1)
plt.title("pearson correlation")
```

Out[30]: Text(0.5, 1, 'pearson correlation')

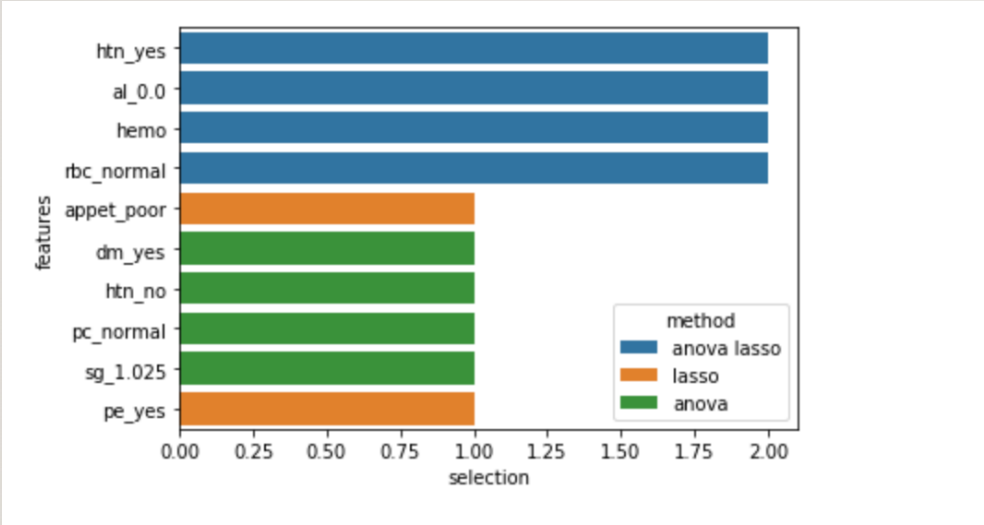


FEATURE SELECTION

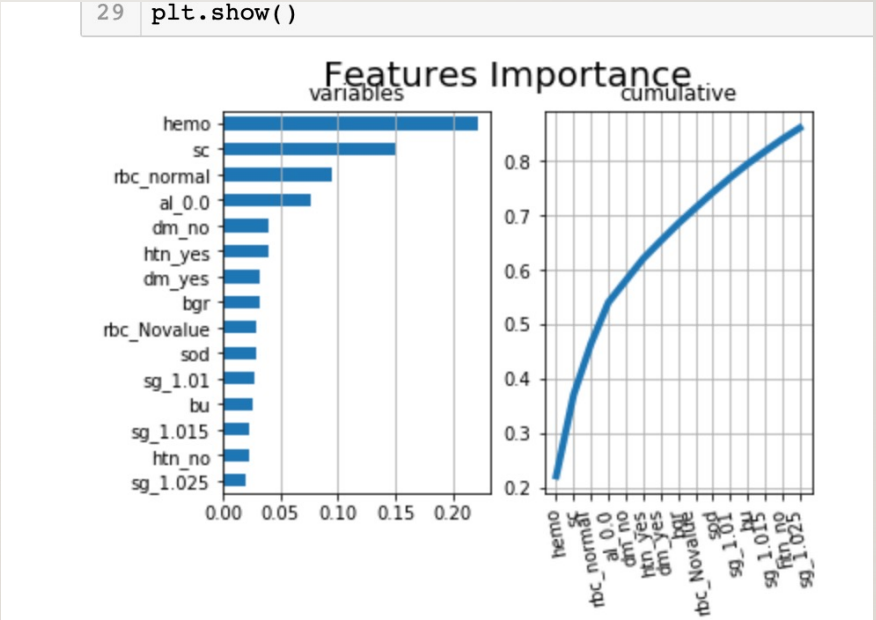
- Observations from Correlation analysis
  - High correlation between
    - hemo & PCV
    - hemo & rbcc
    - PCV & rbcc
- Analysed relation of these variables with target using Anova test
  - Hemo had the lowest P-value, hence dropped PCV and rbcc

FEATURE IMPORTANCE

- Using Anova and Lasso



Using Random Forest

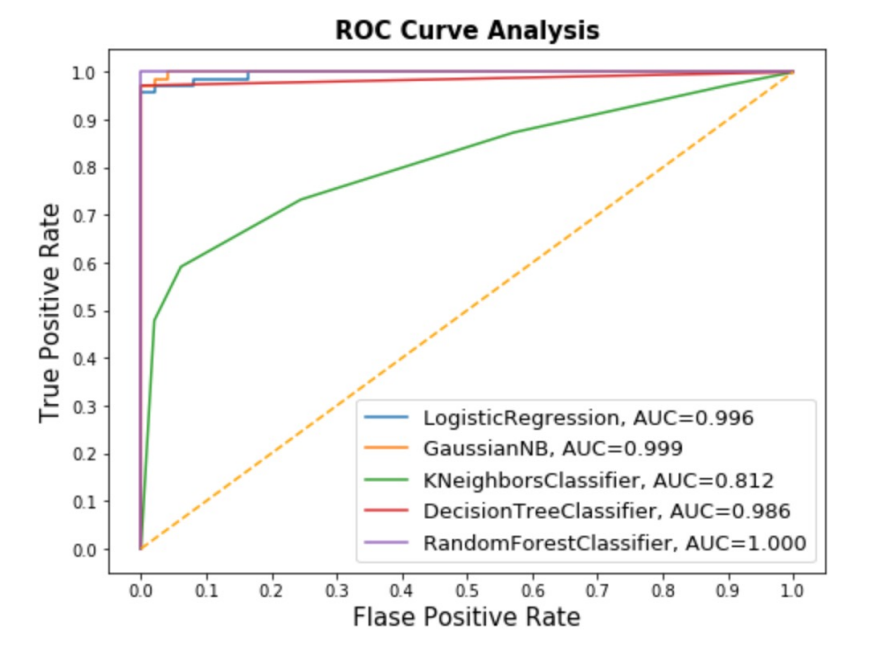


STUDIED THE PERFORMANCES FOR BASE MODELS

- Compared the results for following basic models using ROC and AUC curve which is created on results at different thresholds
  - LogisticRegression
  - GaussianNB
  - KNeighborsClassifier
  - RandomForestClassifier
  - DecisionTreeClassifier

STUDIED THE PERFORMANCES FOR BASE MODELS

- Output Graph



Observation: Random Forest is giving best performance on the data set followed by GuassianNB, Logistic Regression and Decision Tree

Note: Performance is based on Area Under Curve

ANALYZING THE ACCURACY FOR DIFFERENT MODELS WITHOUT HYPER PARAMETER TUNING

- Logictic Regression - 97.14%
- SVM - 63.93%
- KNN - 87.86%
- Gaussian Naive Bayes - 100.0%
- DecisionTreeClassifier – 100.0%
- Linear SVC - 82.5%
- Random Forest – 100.0%

ANALYZING METRICS FOR ALL MODELS ON TEST DATA

Logistic Regression

```
--------------------------------------------------------------------------------
LogisticRegression()
Confusion Matrix
 [[44  2]
 [ 0 74]]
Total Notckd or N:              46
Total ckd or P:                 74
Total Notckd Predicted or N^:   44
Total ckd Predicted or P^:      76
--------------------------------------------------------------------------------
Recall (all 1s predicted right): 1.0
Precision (confidence when predicting a 1): 0.97
Detail:
              precision    recall  f1-score   support

           0       1.00      0.96      0.98        46
           1       0.97      1.00      0.99        74

    accuracy                           0.98       120
   macro avg       0.99      0.98      0.98       120
weighted avg       0.98      0.98      0.98       120
```

SVM

```
--------------------------------------------------------------------------------
SVC()
Confusion Matrix
 [[ 0 46]
 [ 0 74]]
Total Notckd or N:              46
Total ckd or P:                 74
Total Notckd Predicted or N^:   0
Total ckd Predicted or P^:      120
--------------------------------------------------------------------------------
Recall (all 1s predicted right): 1.0
Precision (confidence when predicting a 1): 0.62
Detail:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        46
           1       0.62      1.00      0.76        74

    accuracy                           0.62       120
   macro avg       0.31      0.50      0.38       120
weighted avg       0.38      0.62      0.47       120
```

ANALYZING METRICS FOR ALL MODELS ON TEST DATA

KNN

```
--------------------------------------------------------------------------------
KNeighborsClassifier(n_neighbors=3)
Confusion Matrix
 [[43  3]
 [33 41]]
Total Notckd or N:              46
Total ckd or P:                 74
Total Notckd Predicted or N^:   76
Total ckd Predicted or P^:      44

/usr/local/anaconda3/lib/python3.7/site-packages/sklearn/metrics/_classifica
cision and F-score are ill-defined and being set to 0.0 in labels with no pr
meter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/anaconda3/lib/python3.7/site-packages/sklearn/metrics/_classifica
cision and F-score are ill-defined and being set to 0.0 in labels with no pr
meter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/anaconda3/lib/python3.7/site-packages/sklearn/metrics/_classifica
cision and F-score are ill-defined and being set to 0.0 in labels with no pr
meter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))

--------------------------------------------------------------------------------
Recall (all 1s predicted right): 0.55
Precision (confidence when predicting a 1): 0.93
Detail:
              precision    recall  f1-score   support

           0       0.57      0.93      0.70        46
           1       0.93      0.55      0.69        74

    accuracy                           0.70       120
   macro avg       0.75      0.74      0.70       120
weighted avg       0.79      0.70      0.70       120
```

Gaussian Naive Bayes

```
--------------------------------------------------------------------------------
GaussianNB()
Confusion Matrix
 [[46  0]
 [ 2 72]]
Total Notckd or N:              46
Total ckd or P:                 74
Total Notckd Predicted or N^:   48
Total ckd Predicted or P^:      72
--------------------------------------------------------------------------------
Recall (all 1s predicted right): 0.97
Precision (confidence when predicting a 1): 1.0
Detail:
              precision    recall  f1-score   support

           0       0.96      1.00      0.98        46
           1       1.00      0.97      0.99        74

    accuracy                           0.98       120
   macro avg       0.98      0.99      0.98       120
weighted avg       0.98      0.98      0.98       120
```

ANALYZING METRICS FOR ALL MODELS

DecisionTreeClassifier

```
----------------------------------------------------------------------
DecisionTreeClassifier()
Confusion Matrix
 [[46  0]
 [ 1 73]]
Total Notckd or N:               46
Total ckd or P:                   74
Total Notckd Predicted or N^:    47
Total ckd Predicted or P^:        73
----------------------------------------------------------------------
Recall (all 1s predicted right): 0.99
Precision (confidence when predicting a 1): 1.0
Detail:
              precision    recall  f1-score   support

           0       0.98      1.00      0.99        46
           1       1.00      0.99      0.99        74

    accuracy                           0.99       120
   macro avg       0.99      0.99      0.99       120
weighted avg       0.99      0.99      0.99       120

----------------------------------------------------------------------
```

Linear SVC

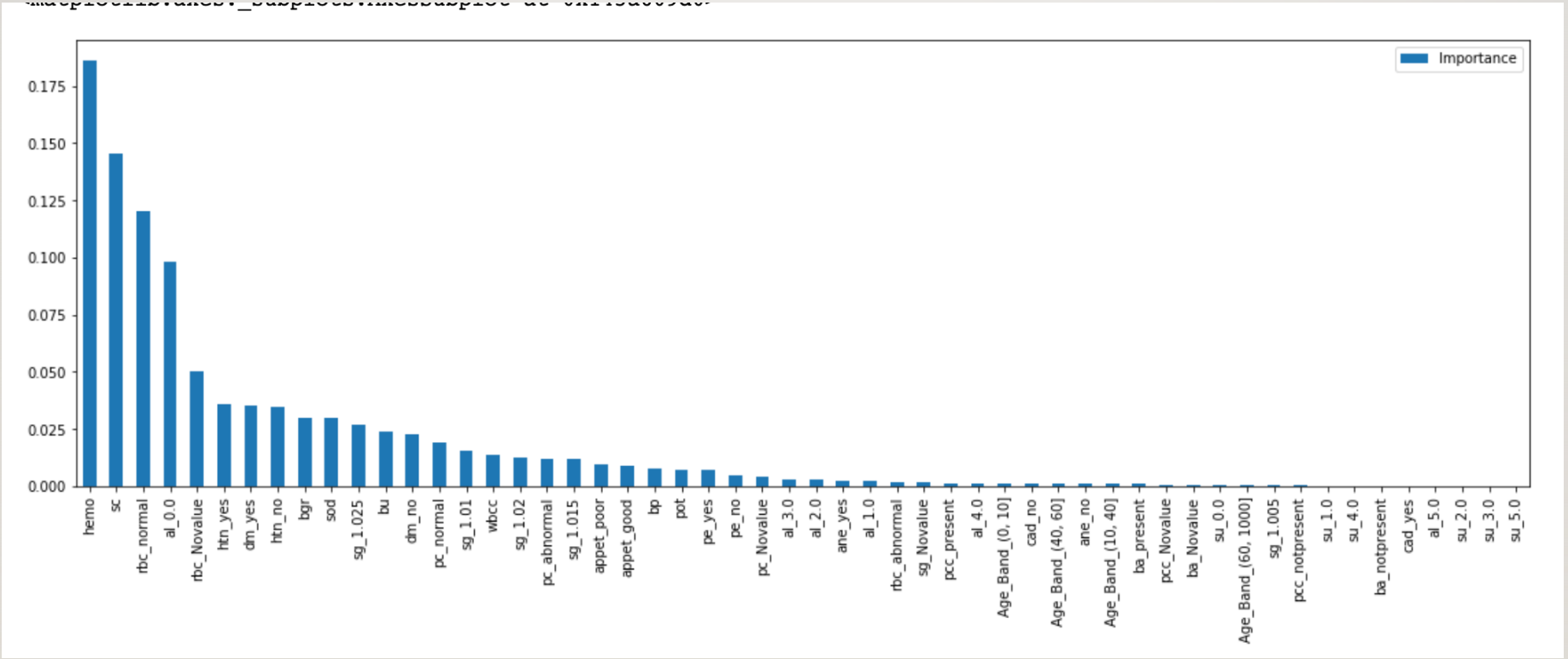
```
----------------------------------------------------------------------
LinearSVC()
Confusion Matrix
 [[46  0]
 [50 24]]
Total Notckd or N:               46
Total ckd or P:                   74
Total Notckd Predicted or N^:    96
Total ckd Predicted or P^:        24
----------------------------------------------------------------------
Recall (all 1s predicted right): 0.32
Precision (confidence when predicting a 1): 1.0
Detail:
              precision    recall  f1-score   support

           0       0.48      1.00      0.65        46
           1       1.00      0.32      0.49        74

    accuracy                           0.58       120
   macro avg       0.74      0.66      0.57       120
weighted avg       0.80      0.58      0.55       120
```

ANALYZING METRICS FOR ALL MODELS

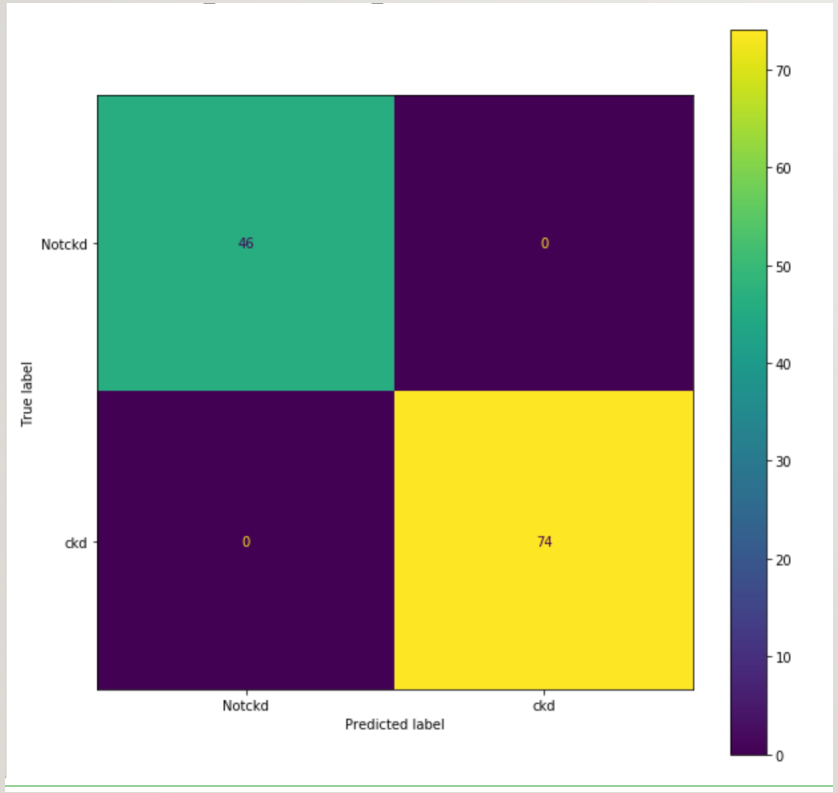
Random Forest – 100.0%

```
----------------------------------------------------------------------
RandomForestClassifier()
Confusion Matrix
 [[46  0]
 [ 0 74]]
Total Notckd or N:               46
Total ckd or P:                   74
Total Notckd Predicted or N^:    46
Total ckd Predicted or P^:        74
----------------------------------------------------------------------
Recall (all 1s predicted right): 1.0
Precision (confidence when predicting a 1): 1.0
Detail:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        46
           1       1.00      1.00      1.00        74

    accuracy                           1.00       120
   macro avg       1.00      1.00      1.00       120
weighted avg       1.00      1.00      1.00       120
```

ANALYZING FEATURE IMPORTANCE FROM RANDOM FOREST – SELECTED RF AS FINAL MODEL



TESTING RANDOM FOREST MODEL ON TEST DATA




```
Confusion Matrix
 [[46  0]
 [ 0 74]]
Total Notckd or N:                46
Total ckd or P:                   74
Total Notckd Predicted or N^:     46
Total ckd Predicted or P^:        74

: <sklearn.metrics._plot.confusion_matrix.Confu
```

FACTORS CAUSING CHRONIC KIDNEY DISEASE AS PER RANDOM FOREST MODEL

9]:

	Importance	cumulative_weight
hemo	0.209200	20.92%
sc	0.130929	34.01%
rbc_normal	0.096113	43.62%
al_0.0	0.085864	52.21%
bgr	0.052274	57.44%
htn_no	0.051880	62.63%
sod	0.038051	66.43%
htn_yes	0.035430	69.97%
rbc_Novalue	0.033645	73.34%
dm_yes	0.031389	76.48%
pc_normal	0.025399	79.02%
sg_1.025	0.022806	81.30%
dm_no	0.022512	83.55%
sg_1.01	0.020134	85.56%
bu	0.019901	87.55%

THANK YOU