Project Report

Health Analysis

INFO 7390

Advance Data Science & Architecture

Professor: Srikanth Krishnamurthy

By: Team 5

Akash Jagtap

Jerin Rajan

Nitin Prince Reuben

Contents

Abstract 4
Introduction 5
Kidney Disease 6
Dataset 6
Exploratory Data Analysis 6
Data Cleaning 23
Feature Engineering 28
Algorithm 30
Logistic Regression 31
Random Forest Classifier 34
Gradient Naïve Bayes 37
Bernoulli Naïve Bayes 40
Support Vector Classifier 43
Stochastic Gradient Decent 46
K Nearest Neighbor 49
Feature Selection 52
Boruta 52
RFE 54
Summary Matrix 58
Liver Disease 59
Dataset 59
Exploratory Data Analysis 59
Data Cleaning 67
Feature Engineering 71
Algorithm 72
Logistic Regression 73
Random Forest Classifier 76
Gradient Naïve Bayes 79
Bernoulli Naïve Bayes 82
Support Vector Classifier 85
Stochastic Gradient Decent 88
K Nearest Neighbor 91
Summary Matrix 94

Diabetes 95
Dataset 95
Exploratory Data Analysis 95
Data Cleaning 101
Feature Engineering 103
Algorithm 104
Logistic Regression 105
Random Forest Classifier 108
Gradient Naïve Bayes 111
Bernoulli Naïve Bayes 114
Support Vector Classifier 117
Stochastic Gradient Decent 119
K Nearest Neighbor 123
Summary Matrix 126
Confusion Matrix 127
F1 Score 128
Precision Score 128
Recall Score 128
Accuracy 129
Docker 130
AWS 131
Flask 132
Airflow & Lambda 162
Future Scope 179

Abstract

Health is the ability of a biological system to acquire, convert, allocate, distribute, and utilize the energy with maximum efficiency. The World Health Organization (WHO) defined human health in a broader sense in its 1948 constitution as "a state of complete physical, mental, and social well-being and not merely the absence of disease or infirmity." This definition has been subject to controversy, as lacking operational value, the ambiguity in developing cohesive health strategies, and because of the problem created by use of the word "complete", which makes it practically impossible to achieve. Other definitions have been proposed, among which a recent definition that correlates health and personal satisfaction.

But to keep your health monitored is the most important thing to do that, we don’t have many applications. So, in an ever growing billion-dollar industry if not trillion, it is important to make provisions available to people to check their own health status even if they don’t have any knowledge about biology or Health Science.

Introduction

With the increase in the number of disease and fatality caused by it, it is important for us to understand to be aware of our health always. So, we are have come up with a solution for it.

We have found a solution for it. We have come up with a model which will help people to be aware of their personal health.

We are covering Liver and Kidney Diseases. Anyone can feed their regular health checkup stats and can find out details whether they are suffering from certain diseases or not.

Kidney Disease

Dataset

You can find the dataset in the below mentioned link.

https://www.kaggle.com/mansoordaku/ckdisease

Exploratory Data Analysis

Let's import the csv file and check how data is distributed.

```
: df1.shape
: (400, 26)
```

Our data contains 26 features and has 400 rows of data.

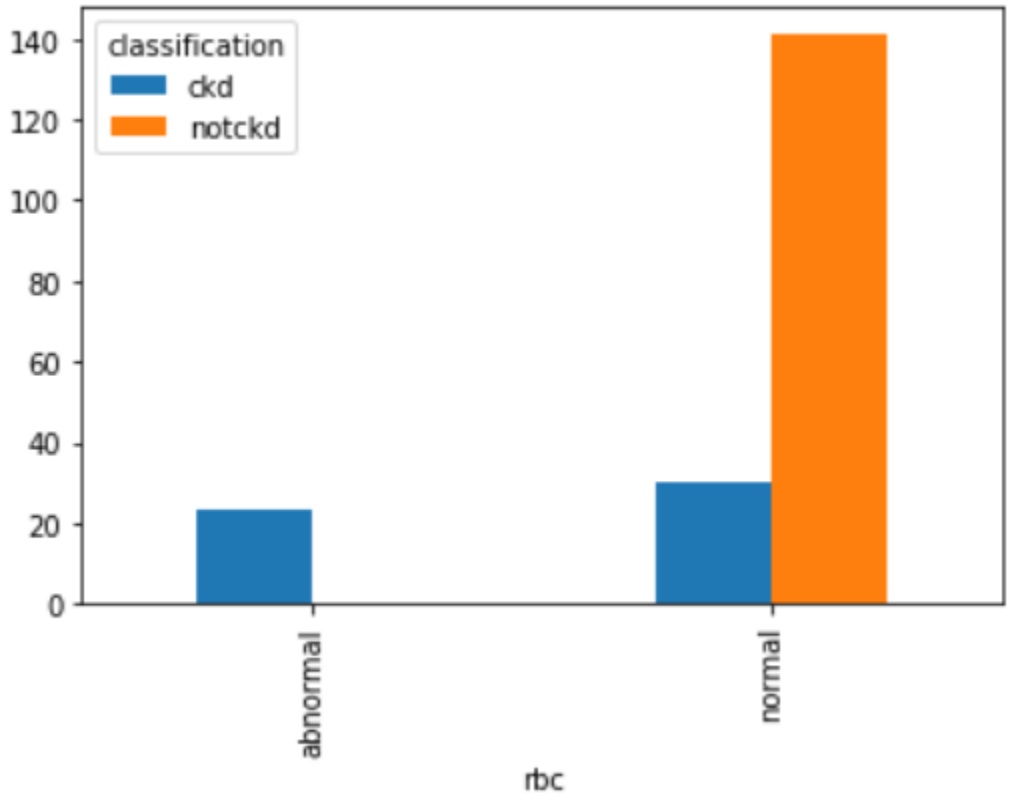
```
df1.isnull().sum()
```

```
id                  0
age                 9
bp                 12
sg                 47
al                 46
su                 49
rbc               152
pc                 65
pcc                 4
ba                  4
bgr                44
bu                 19
sc                 17
sod                87
pot                88
hemo               52
pcv                70
wc                105
rc                130
htn                 2
dm                  2
cad                 2
appet               1
pe                  1
ane                 1
classification      0
dtype: int64
```

There are lot of missing values in our dataset. It is important for us to handle them before making our model.

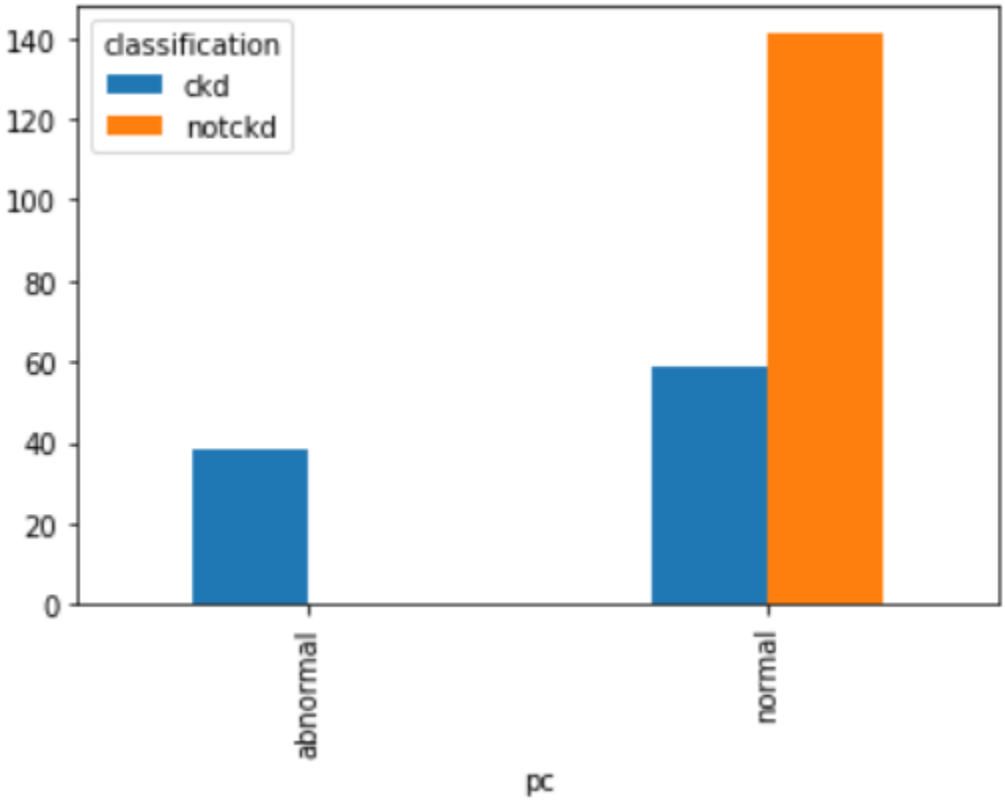
There are lots of features in our dataset which has string data. We are performing count plot on every feature.

Red Blood cell



Patients with abnormal red blood cell will suffer from kidney disease. There are chances of having kidney disease even if you have normal red blood cell count.

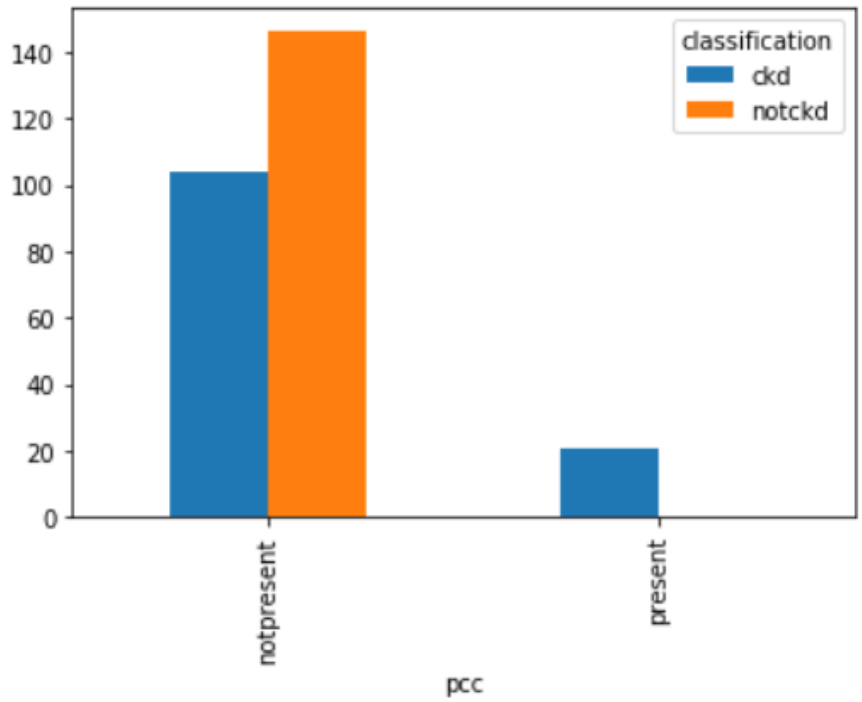
Packed Cell



Patients with abnormal packed cell are prone to kidney disease. There are chances of having kidney disease even if you have normal packed cell count.

Prothombin Complex Concentrate

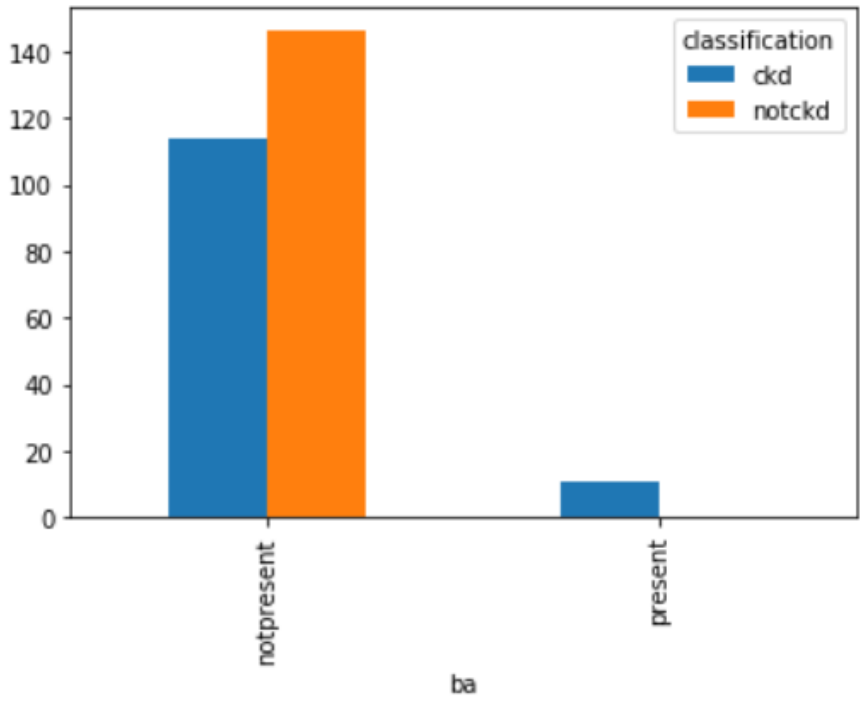
```
d = df1.groupby(['pcc', 'classification']).size().unstack(level=1)
d = d / d.count()
p = d.plot(kind='bar')
```



People who lack Prothombin Complex Concentrate have less chances of gettin kidney disease and people who have Prothombin Complex Concentrate have high chances of getting kidney disease. But in our dataset, there are more entries of people not having Prothombin Complex Concentrate.

Barium

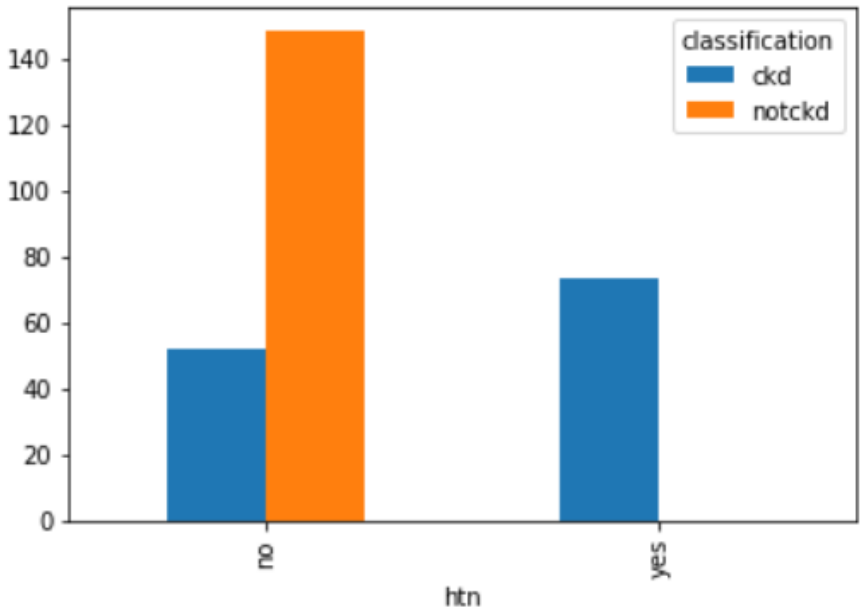
```
d = df1.groupby(['ba', 'classification']).size().unstack(level=1)
d = d / d.count()
p = d.plot(kind='bar')
```



People who lack barium have less chances of getting kidney disease and people who have barium have high chances of getting kidney disease. But in our dataset, there are more entries of people not having barium.

Hypertension

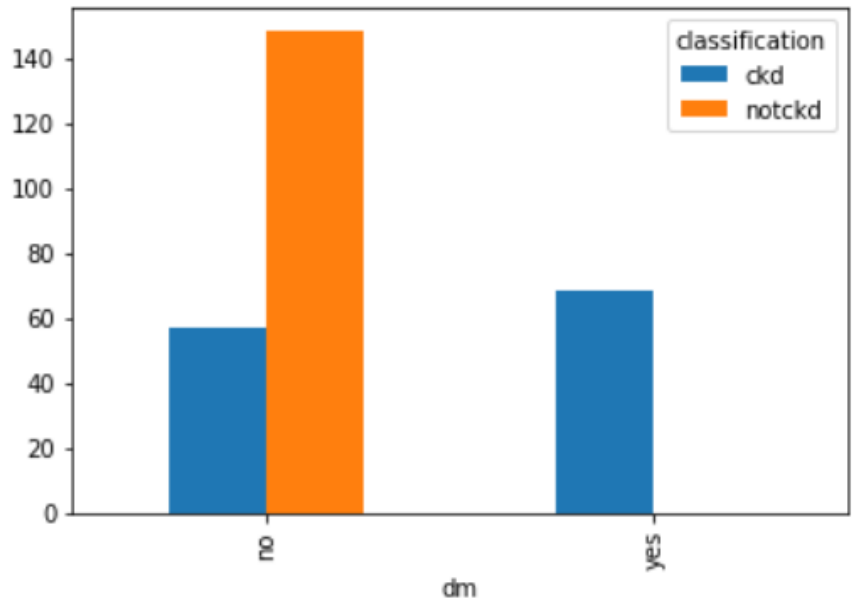
```
d = df1.groupby(['htn', 'classification']).size().unstack(level=1)
d = d / d.count()
p = d.plot(kind='bar')
```



People who have hypertension, they will have kidney disease. People who don't have hypertension have chances of having kidney disease.

Diabetes Mellitus

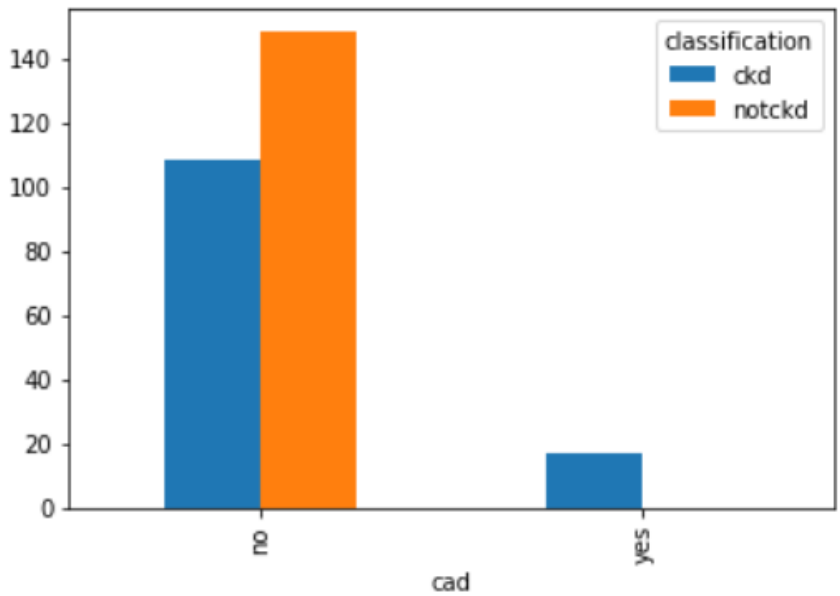
```
d = df1.groupby(['dm', 'classification']).size().unstack(level=1)
d = d / d.count()
p = d.plot(kind='bar')
```



People who lack diabetes mellitus have less chances of getting kidney disease. People who have diabetes mellitus have very high chances of getting kidney disease.

Coronary Artery Disease

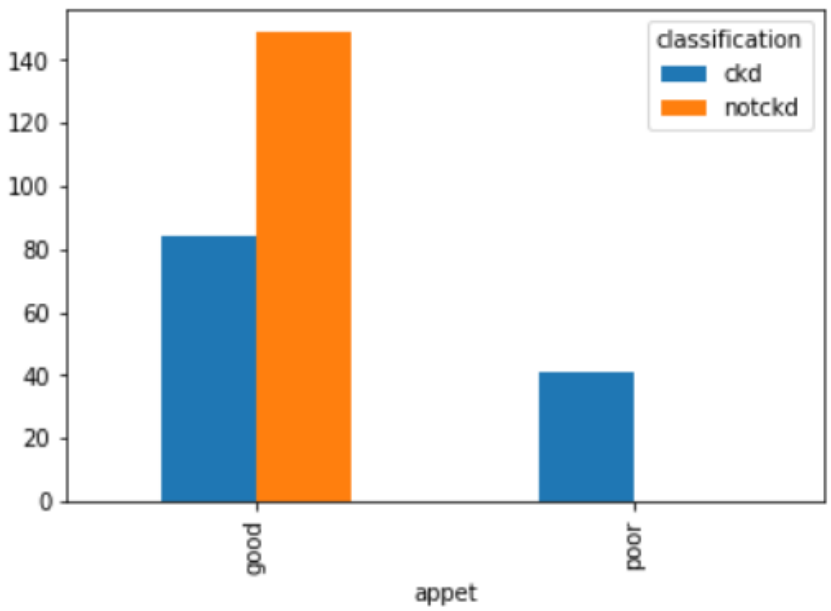
```
d = df1.groupby(['cad', 'classification']).size().unstack(level=1)
d = d / d.count()
p = d.plot(kind='bar')
```



People who have coronary artery disease have very high chances of getting kidney disease. People who don't have coronary artery disease have less chance of getting kidney disease.

Appetite

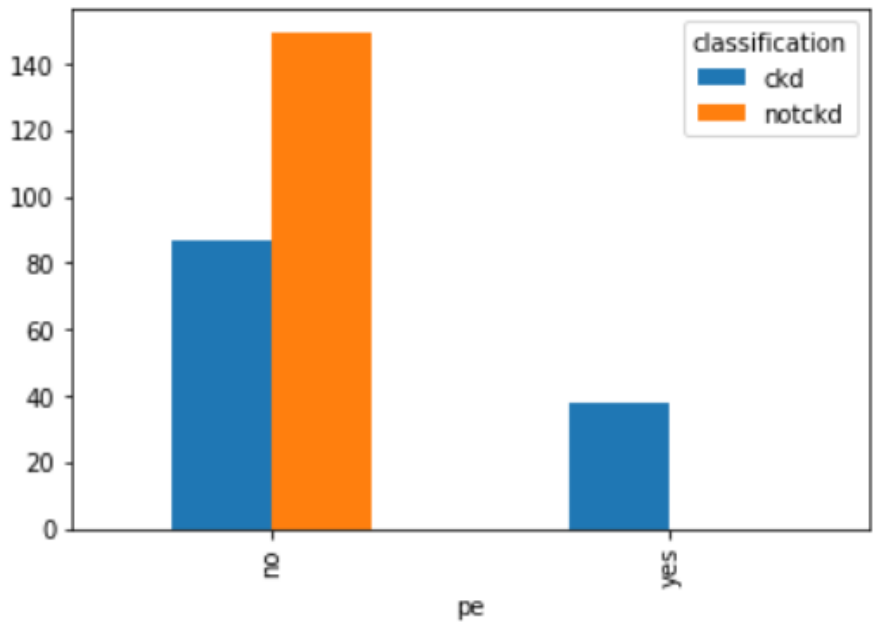
```
d = df1.groupby(['appet', 'classification']).size().unstack(level=1)
d = d / d.count()
p = d.plot(kind='bar')
```



People with poor appetite have more chances of getting kidney disease but from our dataset, we came to understand that people having good appetite can also have kidney disease as we have more entries of this type of data.

Pulmonary Embolism

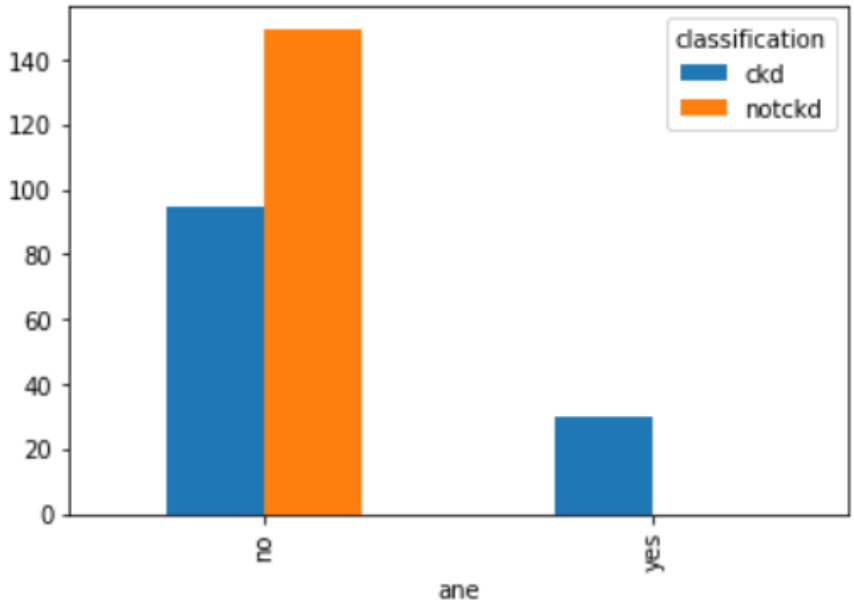
```
d = df1.groupby(['pe', 'classification']).size().unstack(level=1)
d = d / d.count()
p = d.plot(kind='bar')
```



People with pulmonary disease have very high chance of getting kidney disease but from our dataset, we understood that people without pulmonary embolism can also have kidney disease as we have higher entries of this type of data.

Audible Nasal Emission

```
d = df1.groupby(['ane', 'classification']).size().unstack(level=1)
d = d / d.count()
p = d.plot(kind='bar')
```

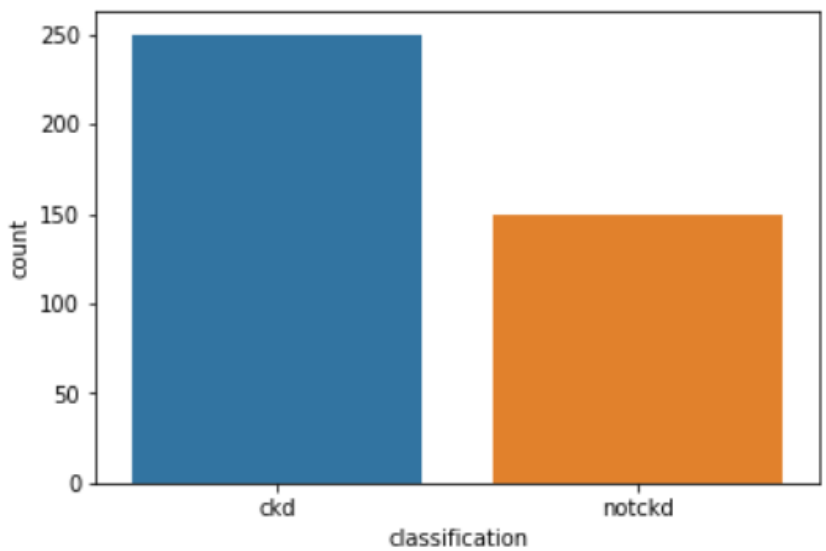


People with audible nasal emission have very high chance of getting kidney disease but from our dataset, we understood that people without audible nasal emission can also have kidney disease as we have higher entries of this type of data.

Disease detected or not

```
classi = pd.DataFrame(df1.groupby('classification').size()).idxmax()[0]
df1['classification'] = df1['classification'].fillna(classi)
sns.countplot(data=df1,x='classification')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x25ccf1db8d0>
```

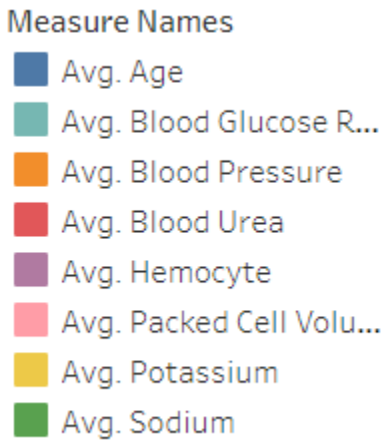


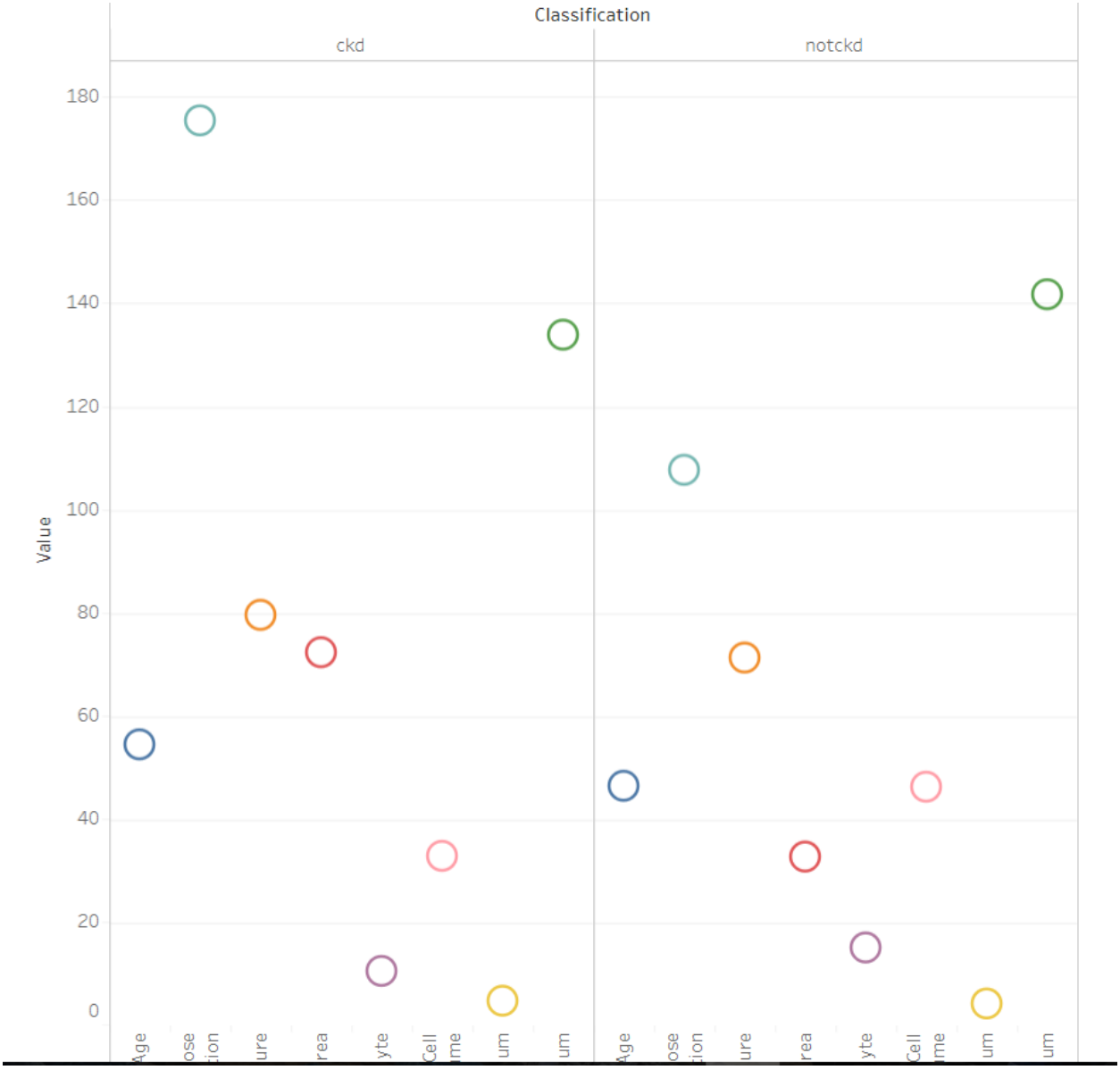

```
df1['classification'].value_counts()
```

```
ckd       250
notckd    150
Name: classification, dtype: int64
```

In our dataset, we have 250 entries with disease and 150 without disease. So, there is good balance in data in 2 outcomes. Thus, no need for oversample/undersample the data to train the model.

There are few numeric values like Blood pressure, Hemocyte, Potassium content, Sodium content etc. They are plotted after taking their average value.

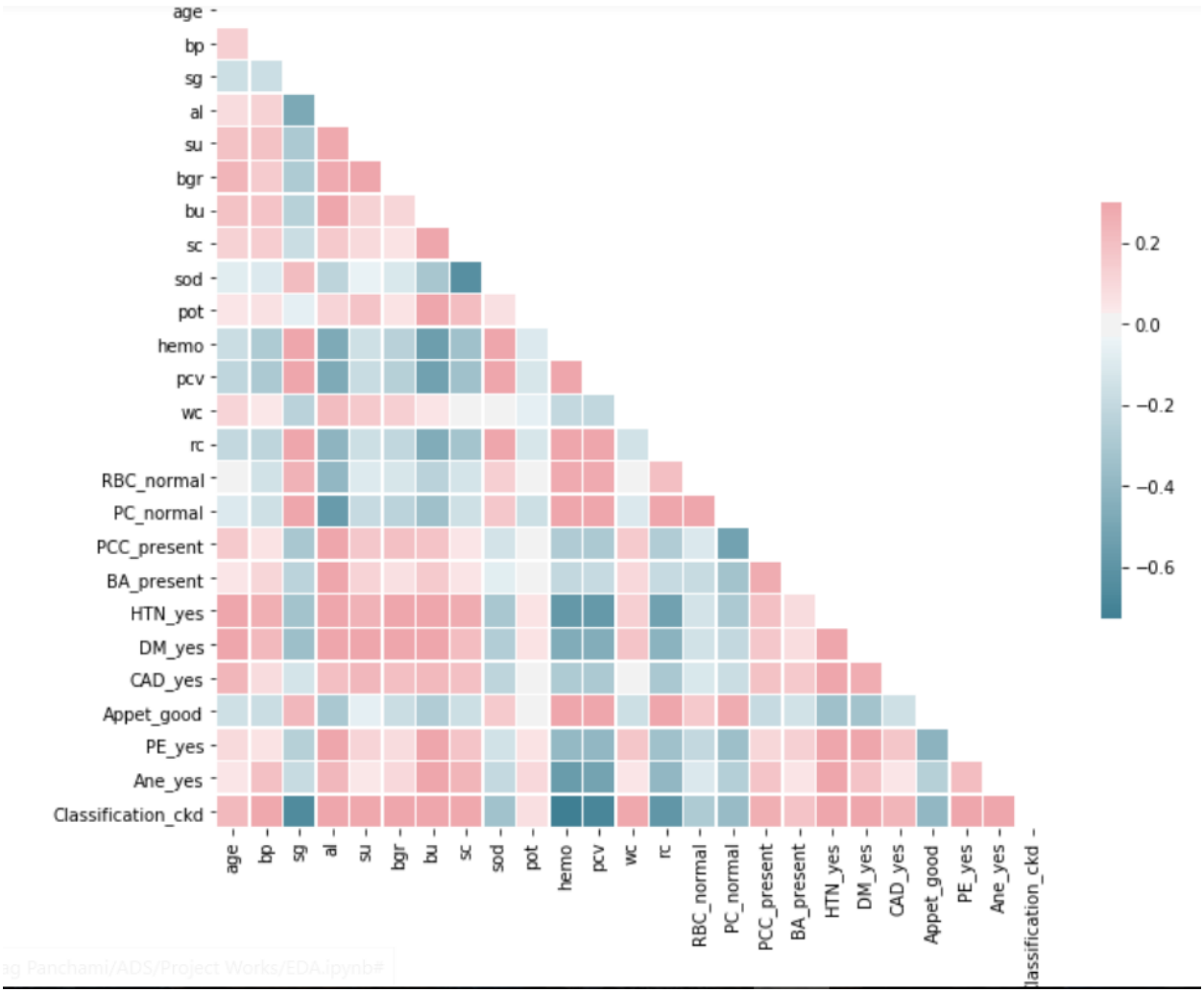




	ckd	notckd
Average Age	54.5	46.5
Average Blood Glucose Regulation	175.4	107.7
Average Blood Pressure	79.6	71.4
Average Blood Urea	72.4	32.8
Average Hemocyte	10.6	15.2
Average Packed Cell Volume	32.9	46.3
Average Potassium	4.9	4.3
Average Sodium	133.9	141.7

From the above table we can say that Average age, Blood Glucose Regulation, Blood Pressure, Blood Urea, Potassium of the people with ckd is higher than those without ckd. Whereas, Average Hemocyte, Packed Cell Volume, Sodium content of the people with ckd is lower than those without ckd.

Coorelation between different features in our dataset.



From the above graph we can see that ckd has higher correlation with age, bp, al, su, bgr, bu, sc, wc, pcc, ba, htn, dm, cad, pe, ane. Whereas ckd has lower correlation with sg,sod,rc,rcb,pc,appet.

All the columns are written in short form. They are all medical terms. It is important for us to understand the full form of these terms.

Short Form	Full Form
Bp	Blood pressure
Sg	Specific gravity
Al	Amyloidosis
Su	Strock unit
Rbc	Red blood cell
Pc	Packed cell
Pcc	Prothrombin complex concentrate
Ba	Barium
Bgr	Blood glucose regulation
Bu	Blood urea
Sod	Sodium
Pot	Potassium
Hemo	Hemocyte
Pcv	Packed cell volume
Wc	White cell
Rc	Red cell
Htn	Hypertension
Dm	Diabetes mellitus
Cad	Coronary artery disease
Pe	Pulmonary embolism
Ane	Audible nasal emission
appet	Appetite

Data Cleaning

Data cleaning is very necessary before running/ making models.

First, we will start with understand the null values in our dataset.

```
df.isnull().sum()
```

```
id                 0
age                9
bp                12
sg                47
al                46
su                49
rbc              152
pc                65
pcc                4
ba                 4
bgr               44
bu                19
sc                17
sod               87
pot               88
hemo              52
pcv               70
wc               105
rc               130
htn                2
dm                 2
cad                2
appet              1
pe                 1
ane                1
classification     0
dtype: int64
```

As we can see we have lots of missing values, it is important for us to handle these. But before that, we must understand the type of data in our dataset.

```
df.dtypes
```

```
id                  int64
age               float64
bp                float64
sg                float64
al                float64
su                float64
rbc                object
pc                 object
pcc                object
ba                 object
bgr               float64
bu                float64
sc                float64
sod               float64
pot               float64
hemo              float64
pcv                object
wc                 object
rc                 object
htn                object
dm                 object
cad                object
appet              object
pe                 object
ane                object
classification     object
dtype: object
```

We can see that not every column has numeric data. So, we should solve 2 problems.

Checking the object type columns

```
In [22]: df['rc'].unique()
```

```
Out[22]: array(['5.2', nan, '3.9', '4.6', '4.4', '5', '4', '3.7', '3.8', '3.4',
       '2.6', '2.8', '4.3', '3.2', '3.6', '4.1', '4.9', '2.5', '4.2',
       '4.5', '3.1', '4.7', '3.5', '6', '2.1', '5.6', '2.3', '2.9', '2.7',
       '8', '3.3', '3', '2.4', '4.8', '\t?', '5.4', '6.1', '6.2', '6.3',
       '5.1', '5.8', '5.5', '5.3', '6.4', '5.7', '5.9', '6.5'], dtype=object)
```

```
In [23]: df['wc'].unique()
```

```
Out[23]: array(['7800', '6000', '7500', '6700', '7300', nan, '6900', '9600',
       '12100', '4500', '12200', '11000', '3800', '11400', '5300', '9200',
       '6200', '8300', '8400', '10300', '9800', '9100', '7900', '6400',
       '8600', '18900', '21600', '4300', '8500', '11300', '7200', '7700',
       '14600', '6300', '\t6200', '7100', '11800', '9400', '5500', '5800',
       '13200', '12500', '5600', '7000', '11900', '10400', '10700',
       '12700', '6800', '6500', '13600', '10200', '9000', '14900', '8200',
       '15200', '5000', '16300', '12400', '\t8400', '10500', '4200',
       '4700', '10900', '8100', '9500', '2200', '12800', '11200', '19100',
       '\t?', '12300', '16700', '2600', '26400', '8800', '7400', '4900',
       '8000', '12000', '15700', '4100', '5700', '11500', '5400', '10800',
       '9900', '5200', '5900', '9300', '9700', '5100', '6600'], dtype=object)
```

```
In [24]: df['pcv'].unique()
```

```
Out[24]: array(['44', '38', '31', '32', '35', '39', '36', '33', '29', '28', nan,
       '16', '24', '37', '30', '34', '40', '45', '27', '48', '\t?', '52',
       '14', '22', '18', '42', '17', '46', '23', '19', '25', '41', '26',
       '15', '21', '43', '20', '\t43', '47', '9', '49', '50', '53', '51',
       '54'], dtype=object)
```

```
In [26]: df['ba'].unique()
```

```
Out[26]: array(['notpresent', 'present', nan], dtype=object)
```

This shows us that we need to handle '\t?' and also type cast few columns

We found that there are special characters in our dataset. This can hinder our analysis. So, replacing it with `null`

```
In [99]: df.replace(to_replace="\t?",value=np.nan,inplace=True)
```

Type casting the columns into numeric data type

```
In [102]: df['rc'] = df['rc'].astype('float64')
          df['pcv'] = df['pcv'].astype('float64')
          df['wc'] = df['wc'].astype('int64')
```

1. Filling missing values.
2. Changing every feature to numeric data.

```
df['age'] = df['age'].fillna(df['age'].mean(axis=0))
df['bgr'] = df['bgr'].fillna(df['bgr'].mean(axis=0))
df['sc'] = df['sc'].fillna(df['sc'].mean(axis=0))
df['bu'] = df['bu'].fillna(df['bu'].mean(axis=0))
df['sod'] = df['sod'].fillna(df['sod'].mean(axis=0))
df['pot'] = df['pot'].fillna(df['pot'].mean(axis=0))
df['pcv'] = df['pcv'].fillna(df['pcv'].mean(axis=0))
df['hemo'] = df['hemo'].fillna(df['hemo'].mean(axis=0))
df['rc'] = df['rc'].fillna(df['rc'].mean(axis=0))
```

Replacing missing values with mean. And then we have rounded off to it's nearest decimal value.

```
df.age = df.age.round()
df.bu = df.bu.round()
df.sod = df.sod.round()
df.pcv = df.pcv.round()
```

The columns which contains string data and has missing values, missing values most occurring value. And then created dummy variable for it.

```
ba = pd.DataFrame(df.groupby('ba').size()).idxmax()[0]
df['ba'] = df['ba'].fillna(ba)

bgr = pd.DataFrame(df.groupby('bgr').size()).idxmax()[0]
df['bgr'] = df['bgr'].fillna(bgr)

wc = pd.DataFrame(df.groupby('wc').size()).idxmax()[0]
df['wc'] = df['wc'].fillna(wc)

htn = pd.DataFrame(df.groupby('htn').size()).idxmax()[0]
df['htn'] = df['htn'].fillna(htn)

dm = pd.DataFrame(df.groupby('dm').size()).idxmax()[0]
df['dm'] = df['dm'].fillna(dm)

wc = pd.DataFrame(df.groupby('wc').size()).idxmax()[0]
df['wc'] = df['wc'].fillna(wc)

cad = pd.DataFrame(df.groupby('cad').size()).idxmax()[0]
df['cad'] = df['cad'].fillna(cad)

appet = pd.DataFrame(df.groupby('appet').size()).idxmax()[0]
df['appet'] = df['appet'].fillna(appet)

pe = pd.DataFrame(df.groupby('pe').size()).idxmax()[0]
df['pe'] = df['pe'].fillna(pe)

ane = pd.DataFrame(df.groupby('ane').size()).idxmax()[0]
df['ane'] = df['ane'].fillna(ane)

classification = pd.DataFrame(df.groupby('classification').size()).idxmax()[0]
df['classification'] = df['classification'].fillna(classification)
```

```
df=pd.get_dummies(df,prefix=['RBC','PC','PCC' , 'BA' , 'HTN' ,
                             'DM' , 'CAD' , 'Appet' , 'PE' , 'Ane' , 'Classification'],
                  columns=['rbc','pc','pcc' , 'ba' , 'htn' , 'dm' , 'cad' , 'appet' , 'pe' , 'ane' , 'classification'])
```

Removing all unwanted columns

```
In [111]: df.drop(['RBC_abnormal'],axis=1,inplace=True)
          df.drop(['PC_abnormal'],axis=1,inplace=True)
          df.drop(['PCC_notpresent'],axis=1,inplace=True)
          df.drop(['BA_notpresent'],axis=1,inplace=True)
          df.drop(['HTN_no'],axis=1,inplace=True)
          df.drop(['DM_no'],axis=1,inplace=True)
          df.drop(['CAD_no'],axis=1,inplace=True)
          df.drop(['Appet_poor'],axis=1,inplace=True)
          df.drop(['PE_no'],axis=1,inplace=True)
          df.drop(['Ane_no'],axis=1,inplace=True)
          df.drop(['Classification_notckd'],axis=1,inplace=True)
          df.drop(['id'],axis=1,inplace=True)
```

Saving the cleaned data in a new csv file.

```
In [114]: df.to_csv('ckd.csv' , index = False)
```

Conclusion: We have handled all the missing data from our dataset and we have also removed all the string values and converted it to numeric data which can make our data analysis easy.

Feature Engineering

From the below code, we have run to find the we are trying to find the key features in our random forest classifier.

```
feature_list = list(x_train.columns)
importances = list(rfc.feature_importances_)
feature_importances = [(x_train, round(importance, 2)) for x_train, importance in zip(feature_list, importances)]
feature_importances = sorted(feature_importances, key = lambda x: x[1], reverse = True)
[print('Variable: {:20} Importance: {}'.format(*pair)) for pair in feature_importances];
```

```
Variable: sc                   Importance: 0.18
Variable: hemo                 Importance: 0.16
Variable: pcv                  Importance: 0.14
Variable: sg                   Importance: 0.12
Variable: rc                   Importance: 0.11
Variable: al                   Importance: 0.06
Variable: DM_yes               Importance: 0.05
Variable: bu                   Importance: 0.04
Variable: sod                  Importance: 0.03
Variable: bgr                  Importance: 0.02
Variable: age                  Importance: 0.01
Variable: bp                   Importance: 0.01
Variable: su                   Importance: 0.01
Variable: pot                  Importance: 0.01
Variable: wc                   Importance: 0.01
Variable: PC_normal            Importance: 0.01
Variable: HTN_yes              Importance: 0.01
Variable: Appet_good           Importance: 0.01
Variable: PE_yes               Importance: 0.01
Variable: RBC_normal           Importance: 0.0
Variable: PCC_present          Importance: 0.0
Variable: BA_present           Importance: 0.0
Variable: CAD_yes              Importance: 0.0
Variable: Ane_yes              Importance: 0.0
```

We have found that there are certain columns don't have importance. So, we are building a new model where we are removing these columns.

```
drop_col_list=['RBC_normal','PCC_present','BA_present','CAD_yes','Ane_yes' , 'PC_normal' , 'Appet_good' ,
               'age' , 'pot' , 'wc' , 'PE_yes' , 'bp' , 'su' ]
x_train=df_train.iloc[:,:24]
x_train.drop(drop_col_list,axis=1,inplace=True)
y_train=df_train['Classification_ckd']
scaler.fit(x_train)
x_train_sc=scaler.transform(x_train)
x_test=df_test.iloc[:,:24]
x_test.drop(drop_col_list,axis=1,inplace=True)
y_test=df_test['Classification_ckd']
scaler.fit(x_test)
x_test_sc=scaler.transform(x_test)
```

```
feature_list = list(x_train.columns)
importances = list(rfc.feature_importances_)
feature_importances = [(x_train, round(importance, 2)) for x_train, importance in zip(feature_list, importances)]
feature_importances = sorted(feature_importances, key = lambda x: x[1], reverse = True)
[print('Variable: {:20} Importance: {}'.format(*pair)) for pair in feature_importances];
```

```
Variable: pcv                  Importance: 0.22
Variable: hemo                 Importance: 0.21
Variable: sg                   Importance: 0.13
Variable: sc                   Importance: 0.11
Variable: rc                   Importance: 0.09
Variable: al                   Importance: 0.06
Variable: DM_yes               Importance: 0.05
Variable: bgr                  Importance: 0.03
Variable: sod                  Importance: 0.03
Variable: HTN_yes              Importance: 0.03
Variable: bu                   Importance: 0.02
```

Conclusion : We have found after feature selection that these 11 columns have more importance and it's better to make the model with these.

Now this model will be used to run algorithms.

Algorithm

We are using the same model.

```
df_train,df_test = train_test_split(df,train_size=0.7,random_state=42)
drop_col_list=['RBC_normal','PCC_present','BA_present','CAD_yes','Ane_yes' , 'PC_normal' , 'Appet_good' ,
               'age' , 'pot' , 'wc' , 'PE_yes' , 'bp' , 'su' ]
x_train=df_train.iloc[:,:24]
x_train.drop(drop_col_list,axis=1,inplace=True)
y_train=df_train['Classification_ckd']
scaler.fit(x_train)
x_train_sc=scaler.transform(x_train)
x_test=df_test.iloc[:,:24]
x_test.drop(drop_col_list,axis=1,inplace=True)
y_test=df_test['Classification_ckd']
scaler.fit(x_test)
x_test_sc=scaler.transform(x_test)
```

There are few features like accuracy, f1 score, etc. which we are retrieving from the model after running algorithm. So, we are creating new lists.

```
accuracy =[]
model_name =[]
dataset=[]
f1score = []
precision = []
recall = []
true_positive =[]
false_positive =[]
true_negative =[]
false_negative =[]
```

Logistic Regression

```
logreg = LogisticRegression()
## fitiing the model
logreg.fit(x_train_sc, y_train)
filename = 'logReg_model.sav'
pickle.dump(logreg,open(filename,'wb'))
logreg
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

Results for training data.

Training

```
prediction = logreg.predict(x_train_sc)
f1 = f1_score(y_train, prediction)
p = precision_score(y_train, prediction)
r = recall_score(y_train, prediction)
a = accuracy_score(y_train, prediction)
cm = confusion_matrix(y_train, prediction)
tp = cm[0][0]
fp = cm[0][1]
fn = cm[1][0]
tn = cm[1][1]
dataset.append('Training')
model_name.append('Logistic Regression')
f1score.append(f1)
precision.append(p)
recall.append(r)
accuracy.append(a)
true_positive.append(tp)
false_positive.append(fp)
true_negative.append(tn)
false_negative.append(fn)
cm
```

```
array([[105,   1],
       [  3, 171]], dtype=int64)
```

Results for testing data.

Testing

```
prediction = logreg.predict(x_test_sc)
f1 = f1_score(y_test,  prediction)
p = precision_score(y_test,  prediction)
r = recall_score(y_test,  prediction)
a = accuracy_score(y_test,  prediction)
cm = confusion_matrix(y_test,  prediction)
tp = cm[0][0]
fp = cm[0][1]
fn = cm[1][0]
tn = cm[1][1]
model_name.append('Logistc Regression')
dataset.append('Testing')
f1score.append(f1)
precision.append(p)
recall.append(r)
accuracy.append(a)
true_positive.append(tp)
false_positive.append(fp)
true_negative.append(tn)
false_negative.append(fn)
cm
```

```
array([[44,  0],
       [ 1, 75]], dtype=int64)
```

Random Forest Classifier

```
rfc = RandomForestClassifier(n_estimators=50,random_state=0)
## fitiing the model
rfc.fit(x_train_sc, y_train)
filename = 'RFC_model.sav'
pickle.dump(rfc,open(filename,'wb'))
rfc
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=50, n_jobs=1,
            oob_score=False, random_state=0, verbose=0, warm_start=False)
```

Results for training data.

Training

```
prediction = rfc.predict(x_train_sc)
f1 = f1_score(y_train, prediction)
p = precision_score(y_train, prediction)
r = recall_score(y_train, prediction)
a = accuracy_score(y_train, prediction)
cm = confusion_matrix(y_train, prediction)
tp = cm[0][0]
fp = cm[0][1]
fn = cm[1][0]
tn = cm[1][1]
model_name.append('Random Forest Classifier')
dataset.append('Training')
f1score.append(f1)
precision.append(p)
recall.append(r)
accuracy.append(a)
true_positive.append(tp)
false_positive.append(fp)
true_negative.append(tn)
false_negative.append(fn)
cm
```

```
array([[106,   0],
       [  0, 174]], dtype=int64)
```

Results for testing data.

Testing

```
prediction = rfc.predict(x_test_sc)
f1 = f1_score(y_test,  prediction)
p = precision_score(y_test,  prediction)
r = recall_score(y_test,  prediction)
a = accuracy_score(y_test,  prediction)
cm = confusion_matrix(y_test,  prediction)
tp = cm[0][0]
fp = cm[0][1]
fn = cm[1][0]
tn = cm[1][1]
model_name.append('Random Forest Classifier')
dataset.append('Testing')
f1score.append(f1)
precision.append(p)
recall.append(r)
accuracy.append(a)
true_positive.append(tp)
false_positive.append(fp)
true_negative.append(tn)
false_negative.append(fn)
cm
```

```
array([[44,  0],
       [ 0, 76]], dtype=int64)
```

Gradient Naïve Bayes

```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(x_train_sc, y_train)
filename = 'GNB_model.sav'
pickle.dump(gnb,open(filename,'wb'))
gnb
```

```
GaussianNB(priors=None)
```

Results for training data.

Training

```
prediction = gnb.predict(x_train_sc)
f1 = f1_score(y_train, prediction)
p = precision_score(y_train, prediction)
r = recall_score(y_train, prediction)
a = accuracy_score(y_train, prediction)
cm = confusion_matrix(y_train, prediction)
tp = cm[0][0]
fp = cm[0][1]
fn = cm[1][0]
tn = cm[1][1]
model_name.append('Gradient Naive Bayes')
dataset.append('Training')
f1score.append(f1)
precision.append(p)
recall.append(r)
accuracy.append(a)
true_positive.append(tp)
false_positive.append(fp)
true_negative.append(tn)
false_negative.append(fn)
cm
```

```
array([[106,   0],
       [ 18, 156]], dtype=int64)
```

Results for testing data.

Testing

```
prediction = gnb.predict(x_test_sc)
f1 = f1_score(y_test,  prediction)
p = precision_score(y_test,  prediction)
r = recall_score(y_test,  prediction)
a = accuracy_score(y_test,  prediction)
cm = confusion_matrix(y_test,  prediction)
tp = cm[0][0]
fp = cm[0][1]
fn = cm[1][0]
tn = cm[1][1]
model_name.append('Gradient Naive Bayes')
dataset.append('Testing')
f1score.append(f1)
precision.append(p)
recall.append(r)
accuracy.append(a)
true_positive.append(tp)
false_positive.append(fp)
true_negative.append(tn)
false_negative.append(fn)
cm
```

```
array([[ 0, 44],
       [ 0, 76]], dtype=int64)
```

Bernoulli Naïve Bayes

```
NB = BernoulliNB()
## fitiing the model
NB.fit(x_train_sc, y_train)
filename = 'BernoulliNB_model.sav'
pickle.dump(NB,open(filename,'wb'))
NB
```

```
BernoulliNB(alpha=1.0, binarize=0.0, class_prior=None, fit_prior=True)
```

Results for training data.

Training

```
prediction = NB.predict(x_train_sc)
f1 = f1_score(y_train, prediction)
p = precision_score(y_train, prediction)
r = recall_score(y_train, prediction)
a = accuracy_score(y_train, prediction)
cm = confusion_matrix(y_train, prediction)
tp = cm[0][0]
fp = cm[0][1]
fn = cm[1][0]
tn = cm[1][1]
model_name.append('Bernoulli Naive Bayes')
dataset.append('Training')
f1score.append(f1)
precision.append(p)
recall.append(r)
accuracy.append(a)
true_positive.append(tp)
false_positive.append(fp)
true_negative.append(tn)
false_negative.append(fn)
cm
```

```
array([[102,   4],
       [  3, 171]], dtype=int64)
```

Results for testing data.

Testing

```
prediction = NB.predict(x_test_sc)
f1 = f1_score(y_test,  prediction)
p = precision_score(y_test,  prediction)
r = recall_score(y_test,  prediction)
a = accuracy_score(y_test,  prediction)
cm = confusion_matrix(y_test,  prediction)
tp = cm[0][0]
fp = cm[0][1]
fn = cm[1][0]
tn = cm[1][1]
model_name.append('Bernoulli Naive Bayes')
dataset.append('Testing')
f1score.append(f1)
precision.append(p)
recall.append(r)
accuracy.append(a)
true_positive.append(tp)
false_positive.append(fp)
true_negative.append(tn)
false_negative.append(fn)
cm
```

```
array([[44,  0],
       [ 2, 74]], dtype=int64)
```

Support Vector Classifier

```
from sklearn.svm import SVC
svc = SVC (kernel = 'linear' , C = 0.025 , random_state = 42)
svc.fit(x_train_sc, y_train)
filename = 'SVC_model.sav'
pickle.dump(svc,open(filename,'wb'))
svc
```

```
SVC(C=0.025, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
  max_iter=-1, probability=False, random_state=42, shrinking=True,
  tol=0.001, verbose=False)
```

Results for training data.

Training

```
prediction = svc.predict(x_train_sc)
f1 = f1_score(y_train, prediction)
p = precision_score(y_train, prediction)
r = recall_score(y_train, prediction)
a = accuracy_score(y_train, prediction)
cm = confusion_matrix(y_train, prediction)
tp = cm[0][0]
fp = cm[0][1]
fn = cm[1][0]
tn = cm[1][1]
model_name.append('Support Vector Classifier')
dataset.append('Training')
f1score.append(f1)
precision.append(p)
recall.append(r)
accuracy.append(a)
true_positive.append(tp)
false_positive.append(fp)
true_negative.append(tn)
false_negative.append(fn)
cm
```

```
array([[105,   1],
       [  4, 170]], dtype=int64)
```

Results for testing data.

Testing

```
prediction = svc.predict(x_test_sc)
f1 = f1_score(y_test,  prediction)
p = precision_score(y_test,  prediction)
r = recall_score(y_test,  prediction)
a = accuracy_score(y_test,  prediction)
cm = confusion_matrix(y_test,  prediction)
tp = cm[0][0]
fp = cm[0][1]
fn = cm[1][0]
tn = cm[1][1]
model_name.append('Support Vector Classifier')
dataset.append('Testing')
f1score.append(f1)
precision.append(p)
recall.append(r)
accuracy.append(a)
true_positive.append(tp)
false_positive.append(fp)
true_negative.append(tn)
false_negative.append(fn)
cm
```

```
array([[44,  0],
       [ 1, 75]], dtype=int64)
```

Stochastic Gradient Decent

```
from sklearn.linear_model import SGDClassifier
sgd = SGDClassifier (loss = 'modified_huber' , shuffle = True , random_state = 42)
sgd.fit(x_train_sc, y_train)
filename = 'SGD_model.sav'
pickle.dump(sgd,open(filename,'wb'))
sgd
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gradient.p
ameters have been added in <class 'sklearn.linear_model.stochastic_gradient.SGDClassi
ey default to max_iter=5 and tol=None. If tol is not None, max_iter defaults to max_i
be 1000, and default tol will be 1e-3.
  "and default tol will be 1e-3." % type(self), FutureWarning)
```

```
SGDClassifier(alpha=0.0001, average=False, class_weight=None, epsilon=0.1,
       eta0=0.0, fit_intercept=True, l1_ratio=0.15,
       learning_rate='optimal', loss='modified_huber', max_iter=None,
       n_iter=None, n_jobs=1, penalty='l2', power_t=0.5, random_state=42,
       shuffle=True, tol=None, verbose=0, warm_start=False)
```

Results for training data.

Training

```
prediction = sgd.predict(x_train_sc)
f1 = f1_score(y_train, prediction)
p = precision_score(y_train, prediction)
r = recall_score(y_train, prediction)
a = accuracy_score(y_train, prediction)
cm = confusion_matrix(y_train, prediction)
tp = cm[0][0]
fp = cm[0][1]
fn = cm[1][0]
tn = cm[1][1]
model_name.append('Stochastic Gradient Decent')
dataset.append('Training')
f1score.append(f1)
precision.append(p)
recall.append(r)
accuracy.append(a)
true_positive.append(tp)
false_positive.append(fp)
true_negative.append(tn)
false_negative.append(fn)
cm
```

```
array([[103,   3],
       [  1, 173]], dtype=int64)
```

Results for testing data.

Testing

```
prediction = sgd.predict(x_test_sc)
f1 = f1_score(y_test,  prediction)
p = precision_score(y_test,  prediction)
r = recall_score(y_test,  prediction)
a = accuracy_score(y_test,  prediction)
cm = confusion_matrix(y_test,  prediction)
tp = cm[0][0]
fp = cm[0][1]
fn = cm[1][0]
tn = cm[1][1]
model_name.append('Stochastic Gradient Decent')
dataset.append('Testing')
f1score.append(f1)
precision.append(p)
recall.append(r)
accuracy.append(a)
true_positive.append(tp)
false_positive.append(fp)
true_negative.append(tn)
false_negative.append(fn)
cm
```

```
array([[44,  0],
       [ 1, 75]], dtype=int64)
```

K Nearest Neighbor

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 2)
knn.fit(x_train_sc, y_train)
filename = 'KNN_model.sav'
pickle.dump(knn,open(filename,'wb'))
knn
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=2, p=2,
           weights='uniform')
```

Results for training data.

Training

```
prediction = knn.predict(x_train_sc)
f1 = f1_score(y_train, prediction)
p = precision_score(y_train, prediction)
r = recall_score(y_train, prediction)
a = accuracy_score(y_train, prediction)
cm = confusion_matrix(y_train, prediction)
tp = cm[0][0]
fp = cm[0][1]
fn = cm[1][0]
tn = cm[1][1]
model_name.append('k nearest Neighbour')
dataset.append('Training')
f1score.append(f1)
precision.append(p)
recall.append(r)
accuracy.append(a)
true_positive.append(tp)
false_positive.append(fp)
true_negative.append(tn)
false_negative.append(fn)
cm
```

```
array([[106,   0],
       [  4, 170]], dtype=int64)
```

Results for testing data.

Testing

```
prediction = knn.predict(x_test_sc)
f1 = f1_score(y_test,  prediction)
p = precision_score(y_test,  prediction)
r = recall_score(y_test,  prediction)
a = accuracy_score(y_test,  prediction)
cm = confusion_matrix(y_test,  prediction)
tp = cm[0][0]
fp = cm[0][1]
fn = cm[1][0]
tn = cm[1][1]
model_name.append('k nearest Neighbour')
dataset.append('Testing')
f1score.append(f1)
precision.append(p)
recall.append(r)
accuracy.append(a)
true_positive.append(tp)
false_positive.append(fp)
true_negative.append(tn)
false_negative.append(fn)
cm
```

```
array([[44,  0],
       [ 1, 75]], dtype=int64)
```

Feature Selection

Boruta

Initially we are building model with our dataset.

```
df_train,df_test = train_test_split(df,train_size=0.7,random_state=42)
x_train=df_train.iloc[:,:24]
y_train=df_train['Classification_ckd']
scaler.fit(x_train)
x_train_sc=scaler.transform(x_train)
x_test=df_test.iloc[:,:24]
y_test=df_test['Classification_ckd']
scaler.fit(x_test)
x_test_sc=scaler.transform(x_test)
```

Now when the model is done, we are trying to execute Boruta.

```
import pandas as pd
#from sklearn.ensemble import RandomForestClassifier
from boruta import BorutaPy

# load X and y
# NOTE BorutaPy accepts numpy arrays only, hence the .values attribute
X = x_train_sc
y = y_train

# define random forest classifier, with utilising all cores and
# sampling in proportion to y labels

rfc = RandomForestClassifier(n_estimators=50,random_state=0)

# define Boruta feature selection method
feat_selector_rf = BorutaPy(rfc, n_estimators='auto', verbose=2)

# find all relevant features
feat_selector_rf.fit(X, y)
```

We can see the output with below screenshot.

```
Out[43]: BorutaPy(alpha=0.05,
     estimator=RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=58, n_jobs=1,
            oob_score=False,
            random_state=<mtrand.RandomState object at 0x0000023A204B8E58>,
            verbose=0, warm_start=False),
     max_iter=100, n_estimators='auto', perc=100,
     random_state=<mtrand.RandomState object at 0x0000023A204B8E58>,
     two_step=True, verbose=2)
```

With the help of Boruta, we are retrieving key features.

```
x_train.columns[feat_selector_rf.support_]
```

```
Index(['age', 'bp', 'sg', 'al', 'bgr', 'bu', 'sc', 'sod', 'hemo', 'pcv', 'wc',
       'rc', 'HTN_yes', 'DM_yes', 'Appet_good'],
      dtype='object')
```

RFE

Initially we are building model with features which we got after Boruta.

```
df_train,df_test = train_test_split(df,train_size=0.7,random_state=42)
drop_col_list=['RBC_normal','PCC_present','BA_present','CAD_yes','Ane_yes' , 'PC_normal' , 'Appet_good' ,
               'age' , 'pot' , 'wc' , 'PE_yes' , 'bp' , 'su' ]
x_train=df_train.iloc[:,:24]
x_train.drop(drop_col_list,axis=1,inplace=True)
y_train=df_train['Classification_ckd']
scaler.fit(x_train)
x_train_sc=scaler.transform(x_train)
x_test=df_test.iloc[:,:24]
x_test.drop(drop_col_list,axis=1,inplace=True)
y_test=df_test['Classification_ckd']
scaler.fit(x_test)
x_test_sc=scaler.transform(x_test)
```

After this, we execute RFE.

```
X = x_train_sc
y = y_train
estimator = SVC(kernel="linear")
selector = RFE(estimator, 10, step=1)
selector = selector.fit(X, y)
selector.support_
```

```
array([False, False,  True,  True, False,  True,  True,  True, False,
       False,  True,  True, False, False, False, False, False, False,
       False,  True, False,  True,  True, False], dtype=bool)
```

Once it’s done, we are retrieving features.

```
x_train.columns[selector.support_]
```

```
Index(['sg', 'al', 'bgr', 'bu', 'sc', 'hemo', 'pcv', 'DM_yes', 'Appet_good',
       'PE_yes'],
      dtype='object')
```

Then we are building new model.

Building model with newly retrived features.

```
column_list = x_train.columns[selector.support_]
x_train=df_train.iloc[:,1:24]
x_train= x_train[column_list]
y_train=df_train['Classification_ckd']
scaler.fit(x_train)
x_train_sc=scaler.transform(x_train)
x_test=df_test.iloc[:,1:24]
x_test = x_test[column_list]
y_test=df_test['Classification_ckd']
x_test_sc=scaler.transform(x_test)
```

And we are running SVC on it.

```
svc = SVC(C=0.025, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
  max_iter=-1, probability=False, random_state=42, shrinking=True,
  tol=0.001, verbose=False)
svc.fit(x_train_sc, y_train)
```

```
SVC(C=0.025, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
  max_iter=-1, probability=False, random_state=42, shrinking=True,
  tol=0.001, verbose=False)
```

On Training Data.

```
prediction = svc.predict(x_train_sc)
f1 = f1_score(y_train, prediction)
p = precision_score(y_train, prediction)
r = recall_score(y_train, prediction)
a = accuracy_score(y_train, prediction)
cm = confusion_matrix(y_train, prediction)
tp = cm[0][0]
fp = cm[0][1]
fn = cm[1][0]
tn = cm[1][1]
model_name.append('Support Vector Classifier')
dataset.append('Training')
f1score.append(f1)
precision.append(p)
recall.append(r)
accuracy.append(a)
true_positive.append(tp)
false_positive.append(fp)
true_negative.append(tn)
false_negative.append(fn)
cm
```

```
array([[106,   0],
       [  3, 171]], dtype=int64)
```

On Testing data.

```
prediction = svc.predict(x_test_sc)
f1 = f1_score(y_test,  prediction)
p = precision_score(y_test,  prediction)
r = recall_score(y_test,  prediction)
a = accuracy_score(y_test,  prediction)
cm = confusion_matrix(y_test,  prediction)
tp = cm[0][0]
fp = cm[0][1]
fn = cm[1][0]
tn = cm[1][1]
model_name.append('Support Vector Classifier')
dataset.append('Testing')
f1score.append(f1)
precision.append(p)
recall.append(r)
accuracy.append(a)
true_positive.append(tp)
false_positive.append(fp)
true_negative.append(tn)
false_negative.append(fn)
cm
```

```
array([[44,  0],
       [ 1, 75]], dtype=int64)
```

Summary Matrix

Initially we are writing all list into one dataframe.

```
Summary = model_name,dataset,f1score,precision,recall,accuracy,true_positive,false_positive,true_negative,false_negative
#Summary
```

```
## Making a dataframe of the accuracy and error metrics
describe1 = pd.DataFrame(Summary[0],columns = {"Model_Name              "})
describe2 = pd.DataFrame(Summary[1],columns = {"Dataset"})
describe3 = pd.DataFrame(Summary[2],columns = {"F1_score"})
describe4 = pd.DataFrame(Summary[3],columns = {"Precision_score"})
describe5 = pd.DataFrame(Summary[4],columns = {"Recall_score"})
describe6 = pd.DataFrame(Summary[5], columns ={"Accuracy_score"})
describe7 = pd.DataFrame(Summary[6], columns ={"True_Positive"})
describe8 = pd.DataFrame(Summary[7], columns ={"False_Positive"})
describe9 = pd.DataFrame(Summary[8], columns ={"True_Negative"})
describe10 = pd.DataFrame(Summary[9], columns ={"False_Negative"})
des = describe1.merge(describe2, left_index=True, right_index=True, how='inner')
des = des.merge(describe3,left_index=True, right_index=True, how='inner')
des = des.merge(describe4,left_index=True, right_index=True, how='inner')
des = des.merge(describe5,left_index=True, right_index=True, how='inner')
des = des.merge(describe6,left_index=True, right_index=True, how='inner')
des = des.merge(describe7,left_index=True, right_index=True, how='inner')
des = des.merge(describe8,left_index=True, right_index=True, how='inner')
des = des.merge(describe9,left_index=True, right_index=True, how='inner')
des = des.merge(describe10,left_index=True, right_index=True, how='inner')
#des = des.merge(describe9,left_index=True, right_index=True, how='inner')
Summary_csv = des.sort_values(ascending=True,by="False_Negative").reset_index(drop = True)
Summary_csv
```

The results are printed in a summary matrix. Screenshot of the same is mentioned below.

Model_Name	Dataset	F1_score	Precision_score	Recall_score	Accuracy_score	True_Positive	False_Positive	True_Negative	False_Negative
Random Forest Classifier	Training	1.000000	1.000000	1.000000	1.000000	106	0	174	0
Random Forest Classifier	Testing	1.000000	1.000000	1.000000	1.000000	44	0	76	0
Stochastic Gradient Decent	Training	0.988571	0.982955	0.994253	0.985714	103	3	173	1
Stochastic Gradient Decent	Testing	0.993377	1.000000	0.986842	0.991667	44	0	75	1
k nearest Neighbour	Training	0.988372	1.000000	0.977011	0.985714	106	0	170	4
k nearest Neighbour	Testing	0.993377	1.000000	0.986842	0.991667	44	0	75	1
Logistic Regression	Training	0.988439	0.994186	0.982759	0.985714	105	1	171	3
Logistc Regression	Testing	0.993377	1.000000	0.986842	0.991667	44	0	75	1
Support Vector Classifier	Training	0.985507	0.994152	0.977011	0.982143	105	1	170	4
Support Vector Classifier	Testing	0.993377	1.000000	0.986842	0.991667	44	0	75	1
Bernoulli Naive Bayes	Training	0.979943	0.977143	0.982759	0.975000	102	4	171	3
Bernoulli Naive Bayes	Testing	0.986667	1.000000	0.973684	0.983333	44	0	74	2
Gradient Naive Bayes	Training	0.945455	1.000000	0.896552	0.935714	106	0	156	18
Gradient Naive Bayes	Testing	0.775510	0.633333	1.000000	0.633333	0	44	76	0

Conclusion: We are concentrating to reduce ‘False_negative’ as much as possible. From the above summary metrics, we can see that ‘Random Forest Classifier’ is giving the best result. So, we will prefer the outcome of this model.

Liver Disease

Dataset

You can find the dataset in below link.

https://www.kaggle.com/jeevannagaraj/indian-liver-patient-dataset

Exploratory Data Analysis

We will start EDA by importing necessary libraries.

```
import pandas as pd
import numpy as np
import sklearn
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
import datetime
import calendar
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
from matplotlib.pylab import rcParams
import seaborn as sns
```

And importing the csv file.

Importing csv file.

```
df = pd.read_csv('liver_final.csv')
```

Now we must find out how much data is there in dataset.

```
df.shape
```

```
(378, 12)
```

We have 12 features/columns and 378 rows of data.

Let’s find out how many null values are there in our dataset.

```
df.isnull().sum()
```

```
id                  0
age                 0
gender              0
tot_bilirubin       0
direct_bilirubin    0
tot_proteins        0
albumin             0
ag_ratio            0
sgpt                0
sgot                0
alkphos             4
is_patient          0
dtype: int64
```

There is only one column with null values. So, the dataset is very much full.

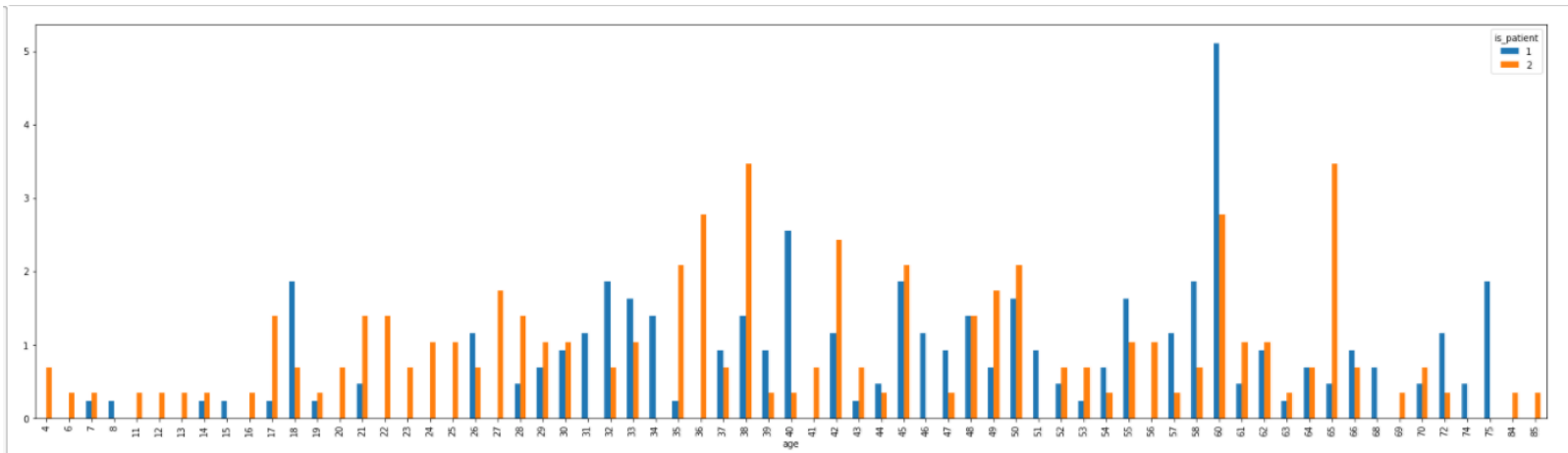
Let’s find out what are datatypes of the data in our dataset.

```
df.dtypes
```

```
id                    int64
age                   int64
gender               object
tot_bilirubin       float64
direct_bilirubin    float64
tot_proteins          int64
albumin               int64
ag_ratio              int64
sgpt                float64
sgot                float64
alkphos             float64
is_patient            int64
dtype: object
```

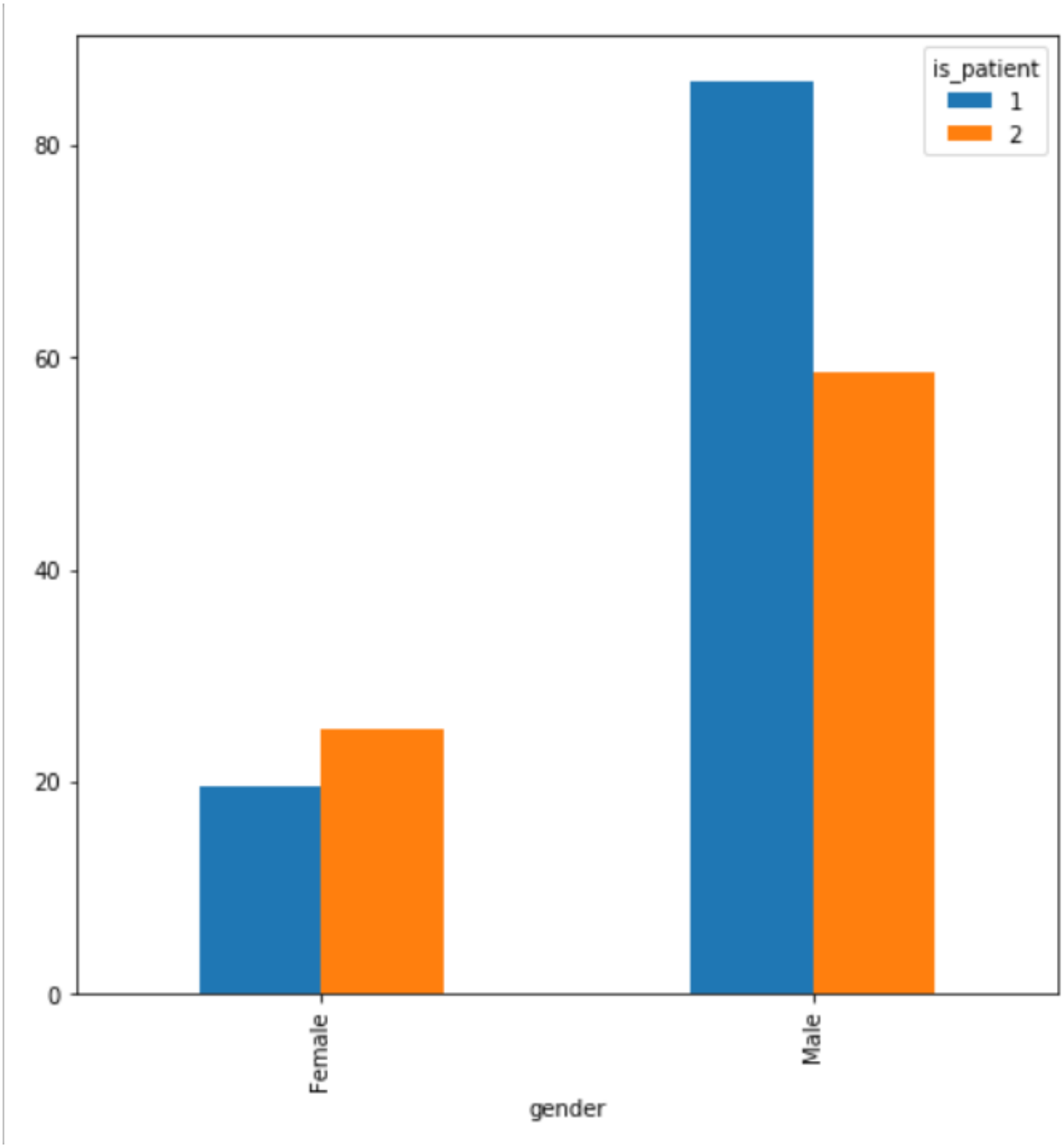
Except for one column, all other columns have numeric data.

Let’s find out how patients are distributed over age.



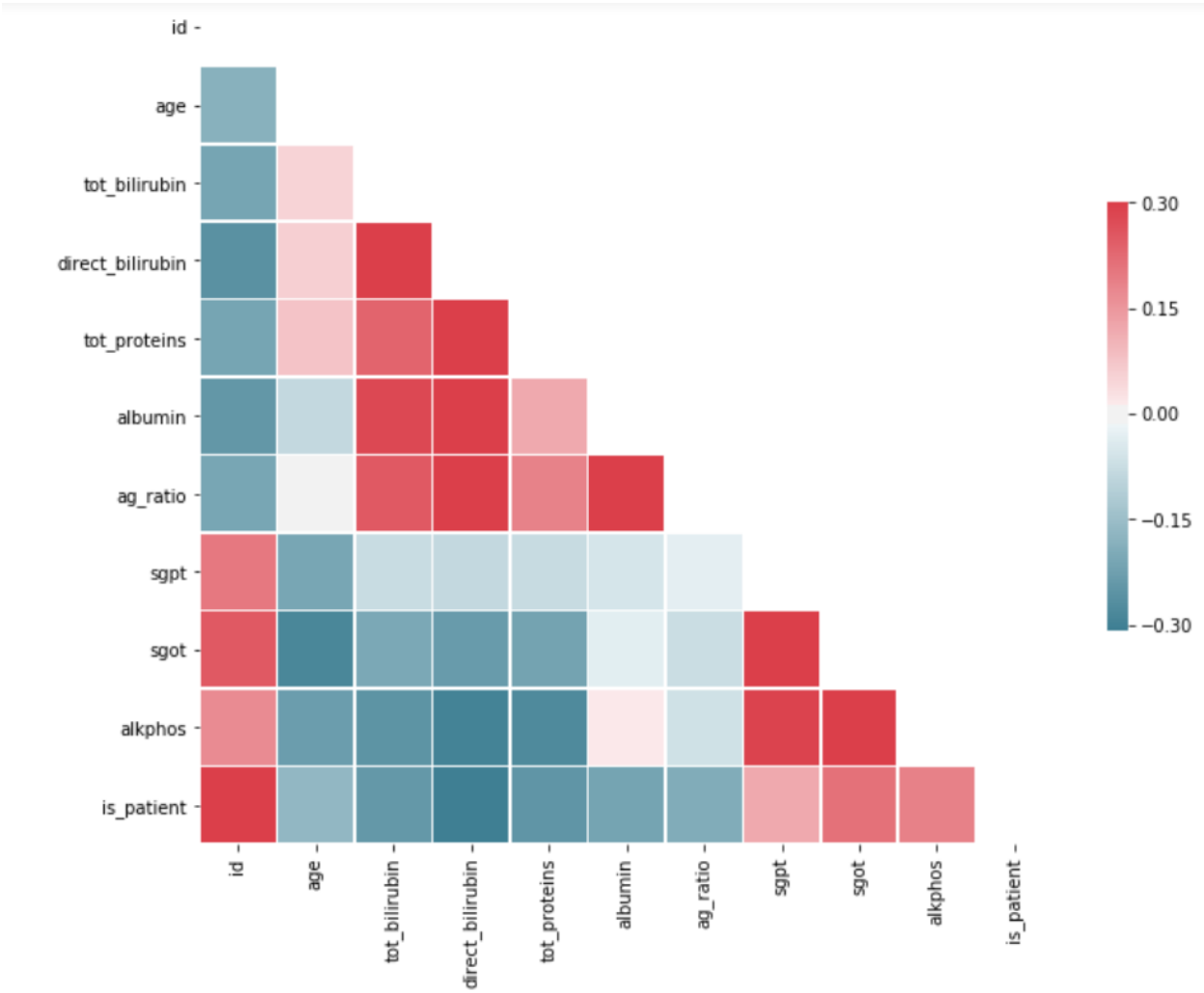
In our dataset, data is distributed evenly over all ages. Highest number of patients are recorder at age 38 and 65. Highest age of not patient is at age = 60.

Let's see how patients are distributed over gender



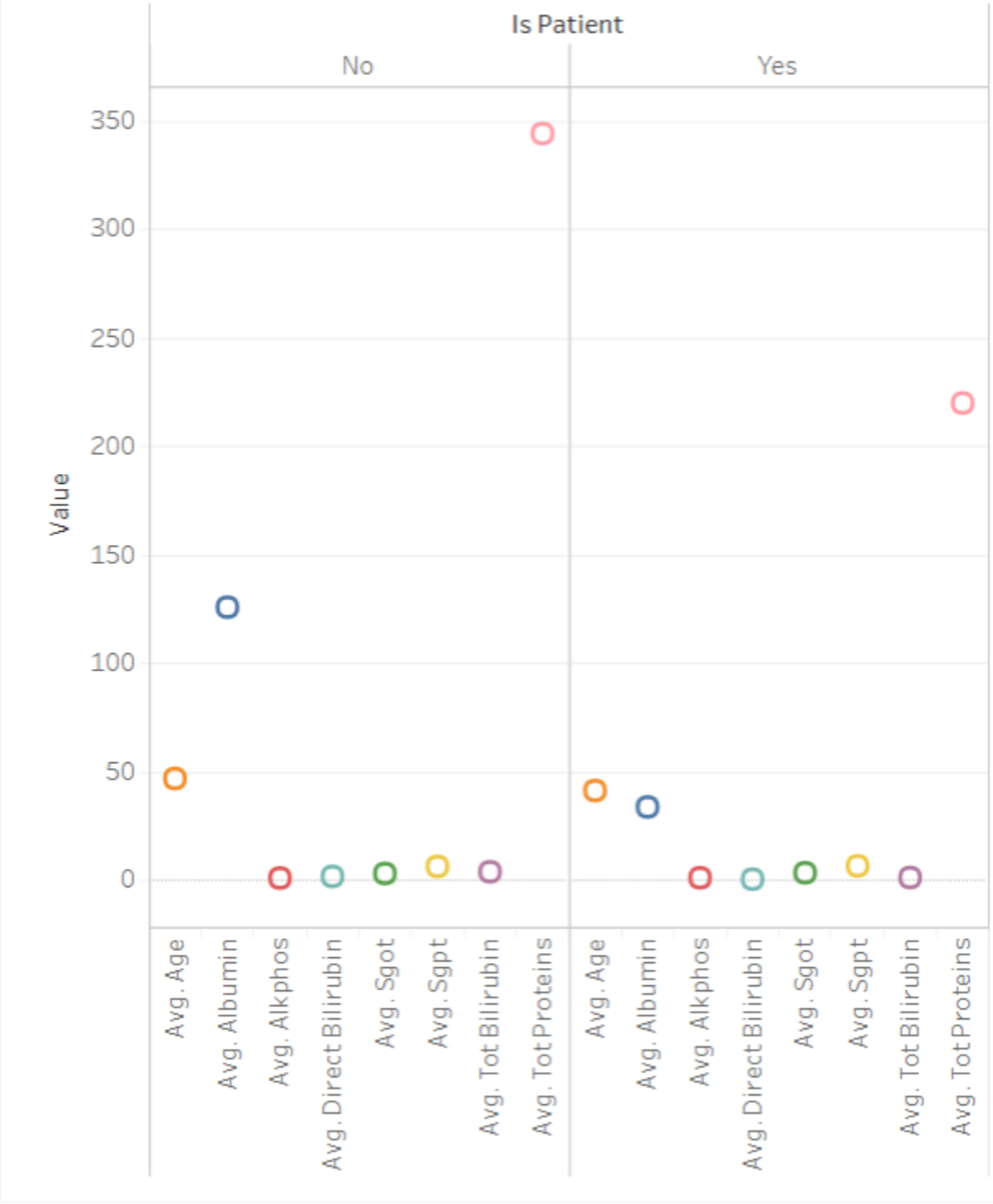
In our dataset, number of male with disease are more as compared to completely well male. Number of female with disease is less than those who are well.

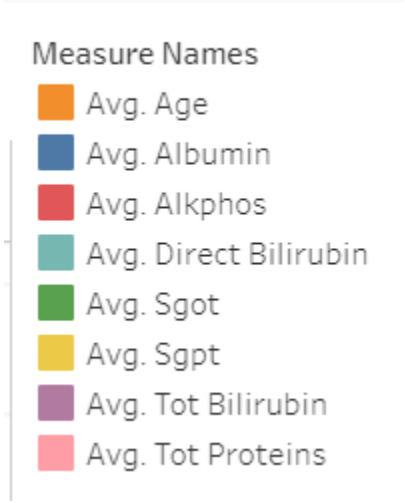
Let's find the correlation between all the columns in our dataset.



From the heat map, we can understand that 'age' have less coorealtion with every column. 'ag_ratio' and 'albumin' have higher correlation between 'tot_bilirubin' and 'direct_bilirubin'. 'alkphos' have higher correlation with 'sgpt' and 'sgot'.

Let’s find out how numeric data is distributed over the dataset.





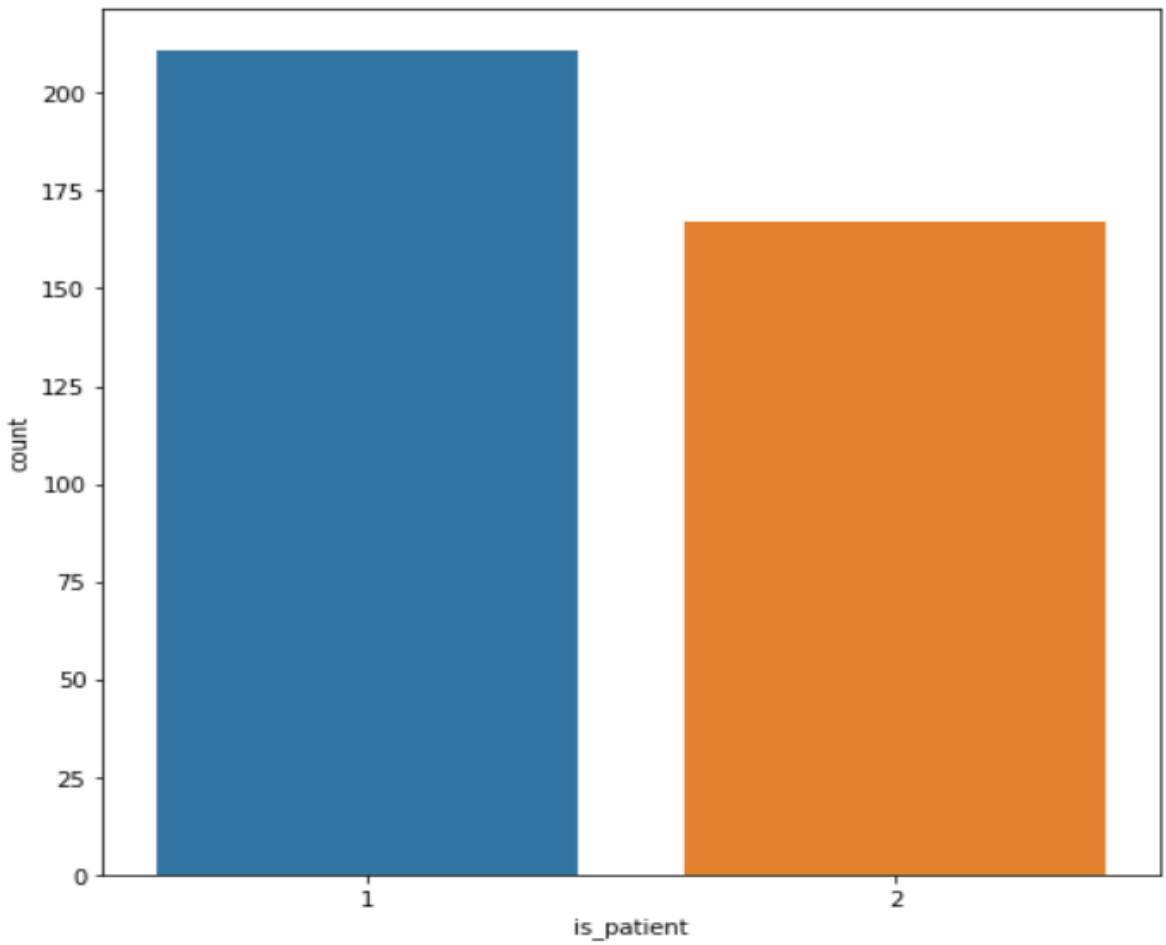
	Is_Patient = No	Is_Patient = Yes
Average Age	46.7	41.2
Average Albumin	125.7	33.7
Average Alkphos	0.9	1.0
Average Direct Bilirubin	1.7	0.4
Average sgot	3.0	3.3
Average sgpt	6.3	6.5
Average Total Bilirubin	3.8	1.1
Average Total Protein	343.9	219.8

This graph is plotted in tableau.

From the above table, we can understand that average age, average albumin, average direct bilirubin, average total bilirubin and average total protein is more for people who are not suffering from liver disease. Whereas, average alkphos, average sgot and average sgpt is more for people who are suffering from liver disease.

Short Form	**Full Form**
Sgot	Serum glutamic oxaloacetic transaminase
sgpt	Serum glutamic pyruvic transaminase

Let's find out how our 'y-axis' is distributed.




```
df['is_patient'].value_counts()
```

```
1    211
2    167
Name: is_patient, dtype: int64
```

There are 211 entries of people without liver disease and 167 entrie of people with liver disease. So, there is good balance of both liver patients and not liver patients in the dataset and we need not oversample or under sample the dataset for training the models.

Data Cleaning

We will start data cleaning by checking the data inside our dataset.

```
df.head()
```

	id	age	gender	tot_bilirubin	direct_bilirubin	tot_proteins	albumin	ag_ratio	sgpt	sgot	alkphos	is_patient
0	1	65	Female	0.7	0.1	187	16	18	6.8	3.3	0.90	1
1	2	62	Male	10.9	5.5	699	64	100	7.5	3.2	0.74	1
2	3	62	Male	7.3	4.1	490	60	68	7.0	3.3	0.89	1
3	4	58	Male	1.0	0.4	182	14	20	6.8	3.4	1.00	1
4	5	72	Male	3.9	2.0	195	27	59	7.3	2.4	0.40	1

Let’s check whether we have any missing data in our file or not.

```
: df.isnull().sum()

: id                 0
  age                0
  gender             0
  tot_bilirubin      0
  direct_bilirubin   0
  tot_proteins       0
  albumin            0
  ag_ratio           0
  sgpt               0
  sgot               0
  alkphos            4
  is_patient         0
  dtype: int64
```

We have one column with few missing entries. Rest all columns are stacked with data.

Let’s find out the data type of data in our dataset.

```
df.dtypes
```

```
id                    int64
age                   int64
gender               object
tot_bilirubin       float64
direct_bilirubin    float64
tot_proteins          int64
albumin               int64
ag_ratio              int64
sgpt                float64
sgot                float64
alkphos             float64
is_patient            int64
dtype: object
```

Except for one column, all other column has numeric data.

Let’s remove missing data with mean value of that column.

```
df['alkphos'] = round(df['alkphos'].fillna(df['alkphos'].mean(axis=0)),2)
```

```
df.isnull().sum()
```

```
id                  0
age                 0
gender              0
tot_bilirubin       0
direct_bilirubin    0
tot_proteins        0
albumin             0
ag_ratio            0
sgpt                0
sgot                0
alkphos             0
is_patient          0
dtype: int64
```

Now we have seen that we have removed the missing values.

Now we will make dummy columns for columns which has string data.

```
df=pd.get_dummies(df,prefix=['gender','patient'],columns=['gender','is_patient'])
df.rename(columns={'patient_2':'is_patient'}, inplace=True)
```

```
df.dtypes
```

```
id                    int64
age                   int64
tot_bilirubin       float64
direct_bilirubin    float64
tot_proteins          int64
albumin               int64
ag_ratio              int64
sgpt                float64
sgot                float64
alkphos             float64
gender_Female         uint8
gender_Male           uint8
patient_1             uint8
is_patient            uint8
dtype: object
```

Now we can see that we don’t have any string data and all the data in our dataset are in numeric form. Now we can proceed with feature selection and running algorithms.

Feature Engineering

From the below code, we have run to find the we are trying to find the key features in our random forest classifier.

```
feature_list = list(x_train.columns)
importances = list(rfc.feature_importances_)
feature_importances = [(x_train, round(importance, 2)) for x_train, importance in zip(feature_list, importances)]
feature_importances = sorted(feature_importances, key = lambda x: x[1], reverse = True)
[print('Variable: {:20} Importance: {}'.format(*pair)) for pair in feature_importances];
```

```
Variable: ag_ratio             Importance: 0.18
Variable: direct_bilirubin     Importance: 0.14
Variable: albumin              Importance: 0.14
Variable: tot_proteins         Importance: 0.13
Variable: age                  Importance: 0.12
Variable: tot_bilirubin        Importance: 0.11
Variable: sgpt                 Importance: 0.09
Variable: alkphos              Importance: 0.08
```

We have found that there are certain columns don't have importance. So, we are building a new model where we are removing these columns.

```
col_list=['age','tot_bilirubin','direct_bilirubin','tot_proteins','albumin','ag_ratio','sgpt'
          ,
          #,'sgot','gender_Male']
         ]
df_train,df_test = train_test_split(df,train_size=0.7,random_state=42)
x_train=df_train[col_list]
y_train=df_train['is_patient']
scaler.fit(x_train)
x_train_sc=scaler.transform(x_train)
x_test=df_test[col_list]
y_test=df_test['is_patient']
scaler.fit(x_test)
x_test_sc=scaler.transform(x_test)
print(x_train.shape,y_train.shape)
```

```
(338, 7) (338,)
```

```
feature_list = list(x_train.columns)
importances = list(rfc.feature_importances_)
feature_importances = [(x_train, round(importance, 2)) for x_train, importance in zip(feature_list, importances)]
feature_importances = sorted(feature_importances, key = lambda x: x[1], reverse = True)
[print('Variable: {:20} Importance: {}'.format(*pair)) for pair in feature_importances];
```

```
Variable: tot_proteins         Importance: 0.19
Variable: albumin              Importance: 0.15
Variable: ag_ratio             Importance: 0.15
Variable: tot_bilirubin        Importance: 0.14
Variable: direct_bilirubin     Importance: 0.13
Variable: age                  Importance: 0.12
Variable: sgpt                 Importance: 0.11
```

Now this model will be used to run algorithms.

Algorithm

We are using the same model.

```
col_list=['age','tot_bilirubin','direct_bilirubin','tot_proteins','albumin','ag_ratio','sgpt'
          ,'alkphos'
          #,'sgot','gender_Male']
         ]
df_train,df_test = train_test_split(df,train_size=0.7,random_state=42)
x_train=df_train[col_list]
y_train=df_train['is_patient']
scaler.fit(x_train)
x_train_sc=scaler.transform(x_train)
x_test=df_test[col_list]
y_test=df_test['is_patient']
scaler.fit(x_test)
x_test_sc=scaler.transform(x_test)
print(x_train.shape,y_train.shape)
```

There are few features like accuracy, f1 score, etc. which we are retrieving from the model after running algorithm. So, we are creating new lists.

```
accuracy =[]
model_name =[]
dataset=[]
f1score = []
precision = []
recall = []
true_positive =[]
false_positive =[]
true_negative =[]
false_negative =[]
```

Logistic Regression

```
logreg = LogisticRegression()
## fitiing the model
logreg.fit(x_train_sc, y_train)
filename = 'logReg_model.sav'
pickle.dump(logreg,open(filename,'wb'))
logreg
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

Results for training data.

Training

```
prediction = logreg.predict(x_train_sc)
f1 = f1_score(y_train, prediction)
p = precision_score(y_train, prediction)
r = recall_score(y_train, prediction)
a = accuracy_score(y_train, prediction)
cm = confusion_matrix(y_train, prediction)
tp = cm[0][0]
fp = cm[0][1]
fn = cm[1][0]
tn = cm[1][1]
dataset.append('Training')
model_name.append('Logistic Regression')
f1score.append(f1)
precision.append(p)
recall.append(r)
accuracy.append(a)
true_positive.append(tp)
false_positive.append(fp)
true_negative.append(tn)
false_negative.append(fn)
cm
```

```
array([[ 42,  55],
       [  6, 235]], dtype=int64)
```

Results for testing data.

Testing

```
prediction = logreg.predict(x_test_sc)
f1 = f1_score(y_test,  prediction)
p = precision_score(y_test,  prediction)
r = recall_score(y_test,  prediction)
a = accuracy_score(y_test,  prediction)
cm = confusion_matrix(y_test,  prediction)
tp = cm[0][0]
fp = cm[0][1]
fn = cm[1][0]
tn = cm[1][1]
model_name.append('Logistc Regression')
dataset.append('Testing')
f1score.append(f1)
precision.append(p)
recall.append(r)
accuracy.append(a)
true_positive.append(tp)
false_positive.append(fp)
true_negative.append(tn)
false_negative.append(fn)
cm
```

```
array([[21, 32],
       [ 0, 93]], dtype=int64)
```

Random Forest Classifier

```
rfc = RandomForestClassifier(n_estimators=50,random_state=0)
## fitiing the model
rfc.fit(x_train_sc, y_train)
filename = 'RFC_model.sav'
pickle.dump(rfc,open(filename,'wb'))
rfc
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=50, n_jobs=1,
            oob_score=False, random_state=0, verbose=0, warm_start=False)
```

Results for training data.

Training

```
prediction = rfc.predict(x_train_sc)
f1 = f1_score(y_train, prediction)
p = precision_score(y_train, prediction)
r = recall_score(y_train, prediction)
a = accuracy_score(y_train, prediction)
cm = confusion_matrix(y_train, prediction)
tp = cm[0][0]
fp = cm[0][1]
fn = cm[1][0]
tn = cm[1][1]
model_name.append('Random Forest Classifier')
dataset.append('Training')
f1score.append(f1)
precision.append(p)
recall.append(r)
accuracy.append(a)
true_positive.append(tp)
false_positive.append(fp)
true_negative.append(tn)
false_negative.append(fn)
cm
```

```
array([[ 97,   0],
       [  0, 241]], dtype=int64)
```

Results for testing data.

Testing

```
prediction = rfc.predict(x_test_sc)
f1 = f1_score(y_test,  prediction)
p = precision_score(y_test,  prediction)
r = recall_score(y_test,  prediction)
a = accuracy_score(y_test,  prediction)
cm = confusion_matrix(y_test,  prediction)
tp = cm[0][0]
fp = cm[0][1]
fn = cm[1][0]
tn = cm[1][1]
model_name.append('Random Forest Classifier')
dataset.append('Testing')
f1score.append(f1)
precision.append(p)
recall.append(r)
accuracy.append(a)
true_positive.append(tp)
false_positive.append(fp)
true_negative.append(tn)
false_negative.append(fn)
cm
```

```
array([[30, 23],
       [ 4, 89]], dtype=int64)
```

Gradient Naïve Bayes

```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(x_train_sc, y_train)
filename = 'GNB_model.sav'
pickle.dump(gnb,open(filename,'wb'))
gnb
```

```
GaussianNB(priors=None)
```

Results for training data.

Training

```
prediction = gnb.predict(x_train_sc)
f1 = f1_score(y_train, prediction)
p = precision_score(y_train, prediction)
r = recall_score(y_train, prediction)
a = accuracy_score(y_train, prediction)
cm = confusion_matrix(y_train, prediction)
tp = cm[0][0]
fp = cm[0][1]
fn = cm[1][0]
tn = cm[1][1]
model_name.append('Gradient Naive Bayes')
dataset.append('Training')
f1score.append(f1)
precision.append(p)
recall.append(r)
accuracy.append(a)
true_positive.append(tp)
false_positive.append(fp)
true_negative.append(tn)
false_negative.append(fn)
cm
```

```
array([[ 40,  57],
       [ 11, 230]], dtype=int64)
```

Results for testing data.

Testing

```
prediction = gnb.predict(x_test_sc)
f1 = f1_score(y_test,  prediction)
p = precision_score(y_test,  prediction)
r = recall_score(y_test,  prediction)
a = accuracy_score(y_test,  prediction)
cm = confusion_matrix(y_test,  prediction)
tp = cm[0][0]
fp = cm[0][1]
fn = cm[1][0]
tn = cm[1][1]
model_name.append('Gradient Naive Bayes')
dataset.append('Testing')
f1score.append(f1)
precision.append(p)
recall.append(r)
accuracy.append(a)
true_positive.append(tp)
false_positive.append(fp)
true_negative.append(tn)
false_negative.append(fn)
cm
```

```
array([[20, 33],
       [ 2, 91]], dtype=int64)
```

Bernoulli Naïve Bayes

```
NB = BernoulliNB()
## fitiing the model
NB.fit(x_train_sc, y_train)
filename = 'BernoulliNB_model.sav'
pickle.dump(NB,open(filename,'wb'))
NB
```

```
BernoulliNB(alpha=1.0, binarize=0.0, class_prior=None, fit_prior=True)
```

Results for training data.

Training

```
prediction = NB.predict(x_train_sc)
f1 = f1_score(y_train, prediction)
p = precision_score(y_train, prediction)
r = recall_score(y_train, prediction)
a = accuracy_score(y_train, prediction)
cm = confusion_matrix(y_train, prediction)
tp = cm[0][0]
fp = cm[0][1]
fn = cm[1][0]
tn = cm[1][1]
model_name.append('Bernoulli Naive Bayes')
dataset.append('Training')
f1score.append(f1)
precision.append(p)
recall.append(r)
accuracy.append(a)
true_positive.append(tp)
false_positive.append(fp)
true_negative.append(tn)
false_negative.append(fn)
cm
```

```
array([[ 57,  40],
       [ 21, 220]], dtype=int64)
```

Results for testing data.

Testing

```
prediction = NB.predict(x_test_sc)
f1 = f1_score(y_test,  prediction)
p = precision_score(y_test,  prediction)
r = recall_score(y_test,  prediction)
a = accuracy_score(y_test,  prediction)
cm = confusion_matrix(y_test,  prediction)
tp = cm[0][0]
fp = cm[0][1]
fn = cm[1][0]
tn = cm[1][1]
model_name.append('Bernoulli Naive Bayes')
dataset.append('Testing')
f1score.append(f1)
precision.append(p)
recall.append(r)
accuracy.append(a)
true_positive.append(tp)
false_positive.append(fp)
true_negative.append(tn)
false_negative.append(fn)
cm
```

```
array([[28, 25],
       [ 5, 88]], dtype=int64)
```

Support Vector Classifier

```
from sklearn.svm import SVC
svc = SVC (kernel = 'linear' , C = 0.025 , random_state = 42)
svc.fit(x_train_sc, y_train)
filename = 'SVC_model.sav'
pickle.dump(svc,open(filename,'wb'))
svc
```

```
SVC(C=0.025, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
  max_iter=-1, probability=False, random_state=42, shrinking=True,
  tol=0.001, verbose=False)
```

Results for training data.

Training

```
prediction = svc.predict(x_train_sc)
f1 = f1_score(y_train, prediction)
p = precision_score(y_train, prediction)
r = recall_score(y_train, prediction)
a = accuracy_score(y_train, prediction)
cm = confusion_matrix(y_train, prediction)
tp = cm[0][0]
fp = cm[0][1]
fn = cm[1][0]
tn = cm[1][1]
model_name.append('Support Vector Classifier')
dataset.append('Training')
f1score.append(f1)
precision.append(p)
recall.append(r)
accuracy.append(a)
true_positive.append(tp)
false_positive.append(fp)
true_negative.append(tn)
false_negative.append(fn)
cm
```

```
array([[ 30,  67],
       [  6, 235]], dtype=int64)
```

Results for testing data.

Testing

```
prediction = svc.predict(x_test_sc)
f1 = f1_score(y_test,  prediction)
p = precision_score(y_test,  prediction)
r = recall_score(y_test,  prediction)
a = accuracy_score(y_test,  prediction)
cm = confusion_matrix(y_test,  prediction)
tp = cm[0][0]
fp = cm[0][1]
fn = cm[1][0]
tn = cm[1][1]
model_name.append('Support Vector Classifier')
dataset.append('Testing')
f1score.append(f1)
precision.append(p)
recall.append(r)
accuracy.append(a)
true_positive.append(tp)
false_positive.append(fp)
true_negative.append(tn)
false_negative.append(fn)
cm
```

```
array([[16, 37],
       [ 0, 93]], dtype=int64)
```

Stochastic Gradient Decent

```
from sklearn.linear_model import SGDClassifier
sgd = SGDClassifier (loss = 'modified_huber' , shuffle = True , random_state = 42)
sgd.fit(x_train_sc, y_train)
filename = 'SGD_model.sav'
pickle.dump(sgd,open(filename,'wb'))
sgd
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gradient.p
ameters have been added in <class 'sklearn.linear_model.stochastic_gradient.SGDClassi
ey default to max_iter=5 and tol=None. If tol is not None, max_iter defaults to max_i
be 1000, and default tol will be 1e-3.
  "and default tol will be 1e-3." % type(self), FutureWarning)
```

```
SGDClassifier(alpha=0.0001, average=False, class_weight=None, epsilon=0.1,
       eta0=0.0, fit_intercept=True, l1_ratio=0.15,
       learning_rate='optimal', loss='modified_huber', max_iter=None,
       n_iter=None, n_jobs=1, penalty='l2', power_t=0.5, random_state=42,
       shuffle=True, tol=None, verbose=0, warm_start=False)
```

Results for training data.

Training

```
prediction = sgd.predict(x_train_sc)
f1 = f1_score(y_train, prediction)
p = precision_score(y_train, prediction)
r = recall_score(y_train, prediction)
a = accuracy_score(y_train, prediction)
cm = confusion_matrix(y_train, prediction)
tp = cm[0][0]
fp = cm[0][1]
fn = cm[1][0]
tn = cm[1][1]
model_name.append('Stochastic Gradient Decent')
dataset.append('Training')
f1score.append(f1)
precision.append(p)
recall.append(r)
accuracy.append(a)
true_positive.append(tp)
false_positive.append(fp)
true_negative.append(tn)
false_negative.append(fn)
cm
```

```
array([[ 51,  46],
       [ 49, 192]], dtype=int64)
```

Results for testing data.

Testing

```
prediction = sgd.predict(x_test_sc)
f1 = f1_score(y_test,  prediction)
p = precision_score(y_test,  prediction)
r = recall_score(y_test,  prediction)
a = accuracy_score(y_test,  prediction)
cm = confusion_matrix(y_test,  prediction)
tp = cm[0][0]
fp = cm[0][1]
fn = cm[1][0]
tn = cm[1][1]
model_name.append('Stochastic Gradient Decent')
dataset.append('Testing')
f1score.append(f1)
precision.append(p)
recall.append(r)
accuracy.append(a)
true_positive.append(tp)
false_positive.append(fp)
true_negative.append(tn)
false_negative.append(fn)
cm
```

```
array([[26, 27],
       [17, 76]], dtype=int64)
```

K Nearest Neighbor

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 2)
knn.fit(x_train_sc, y_train)
filename = 'KNN_model.sav'
pickle.dump(knn,open(filename,'wb'))
knn
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=2, p=2,
           weights='uniform')
```

Results for training data.

Training

```
prediction = knn.predict(x_train_sc)
f1 = f1_score(y_train, prediction)
p = precision_score(y_train, prediction)
r = recall_score(y_train, prediction)
a = accuracy_score(y_train, prediction)
cm = confusion_matrix(y_train, prediction)
tp = cm[0][0]
fp = cm[0][1]
fn = cm[1][0]
tn = cm[1][1]
model_name.append('k nearest Neighbour')
dataset.append('Training')
f1score.append(f1)
precision.append(p)
recall.append(r)
accuracy.append(a)
true_positive.append(tp)
false_positive.append(fp)
true_negative.append(tn)
false_negative.append(fn)
cm
```

```
array([[ 97,   0],
       [ 15, 226]], dtype=int64)
```

Results for testing data.

Testing

```
prediction = knn.predict(x_test_sc)
f1 = f1_score(y_test,  prediction)
p = precision_score(y_test,  prediction)
r = recall_score(y_test,  prediction)
a = accuracy_score(y_test,  prediction)
cm = confusion_matrix(y_test,  prediction)
tp = cm[0][0]
fp = cm[0][1]
fn = cm[1][0]
tn = cm[1][1]
model_name.append('k nearest Neighbour')
dataset.append('Testing')
f1score.append(f1)
precision.append(p)
recall.append(r)
accuracy.append(a)
true_positive.append(tp)
false_positive.append(fp)
true_negative.append(tn)
false_negative.append(fn)
cm
```

```
array([[33, 20],
       [18, 75]], dtype=int64)
```

Summary Matrix

Initially we are writing all list into one dataframe.

```
Summary = model_name,dataset,f1score,precision,recall,accuracy,true_positive,false_positive,true_negative,false_negative
#Summary
```

```
## Making a dataframe of the accuracy and error metrics
describe1 = pd.DataFrame(Summary[0],columns = {"Model_Name              "})
describe2 = pd.DataFrame(Summary[1],columns = {"Dataset"})
describe3 = pd.DataFrame(Summary[2],columns = {"F1_score"})
describe4 = pd.DataFrame(Summary[3],columns = {"Precision_score"})
describe5 = pd.DataFrame(Summary[4],columns = {"Recall_score"})
describe6 = pd.DataFrame(Summary[5], columns ={"Accuracy_score"})
describe7 = pd.DataFrame(Summary[6], columns ={"True_Positive"})
describe8 = pd.DataFrame(Summary[7], columns ={"False_Positive"})
describe9 = pd.DataFrame(Summary[8], columns ={"True_Negative"})
describe10 = pd.DataFrame(Summary[9], columns ={"False_Negative"})
des = describe1.merge(describe2, left_index=True, right_index=True, how='inner')
des = des.merge(describe3,left_index=True, right_index=True, how='inner')
des = des.merge(describe4,left_index=True, right_index=True, how='inner')
des = des.merge(describe5,left_index=True, right_index=True, how='inner')
des = des.merge(describe6,left_index=True, right_index=True, how='inner')
des = des.merge(describe7,left_index=True, right_index=True, how='inner')
des = des.merge(describe8,left_index=True, right_index=True, how='inner')
des = des.merge(describe9,left_index=True, right_index=True, how='inner')
des = des.merge(describe10,left_index=True, right_index=True, how='inner')
#des = des.merge(describe9,left_index=True, right_index=True, how='inner')
Summary_csv = des.sort_values(ascending=True,by="False_Negative").reset_index(drop = True)
Summary_csv
```

The results are printed in a summary matrix. Screenshot of the same is mentioned below.

Model_Name	Dataset	F1_score	Precision_score	Recall_score	Accuracy_score	True_Positive	False_Positive	True_Negative	False_Negative
Random Forest Classifier	Training	1.000000	1.000000	1.000000	1.000000	145	0	119	0
Random Forest Classifier	Testing	0.540000	0.519231	0.562500	0.596491	41	25	27	21
Stochastic Gradient Decent	Training	0.529101	0.714286	0.420168	0.662879	125	20	50	69
Stochastic Gradient Decent	Testing	0.416667	0.625000	0.312500	0.631579	57	9	15	33
k nearest Neighbour	Training	0.839024	1.000000	0.722689	0.875000	145	0	86	33
k nearest Neighbour	Testing	0.404762	0.472222	0.354167	0.561404	47	19	17	31
Logistic Regression	Training	0.754717	0.684932	0.840336	0.753788	99	46	100	19
Logistc Regression	Testing	0.576923	0.535714	0.625000	0.614035	40	26	30	18
Support Vector Classifier	Training	0.737931	0.625731	0.899160	0.712121	81	64	107	12
Support Vector Classifier	Testing	0.672269	0.563380	0.833333	0.657895	35	31	40	8
Bernoulli Naive Bayes	Training	0.755245	0.646707	0.907563	0.734848	86	59	108	11
Bernoulli Naive Bayes	Testing	0.600000	0.500000	0.750000	0.578947	30	36	36	12
Gradient Naive Bayes	Training	0.748387	0.607330	0.974790	0.704545	70	75	116	3
Gradient Naive Bayes	Testing	0.619048	0.500000	0.812500	0.578947	27	39	39	9

Conclusion: We are concentrating to reduce ‘False_negative’ as much as possible. From the above summary metrics, we can see that ‘Random Forest Classifier’ is giving the best result. So, we will prefer the outcome of this model.

Diabetes

Dataset

The dataset which is used can be found in below mentioned link.

https://www.kaggle.com/uciml/pima-indians-diabetes-database

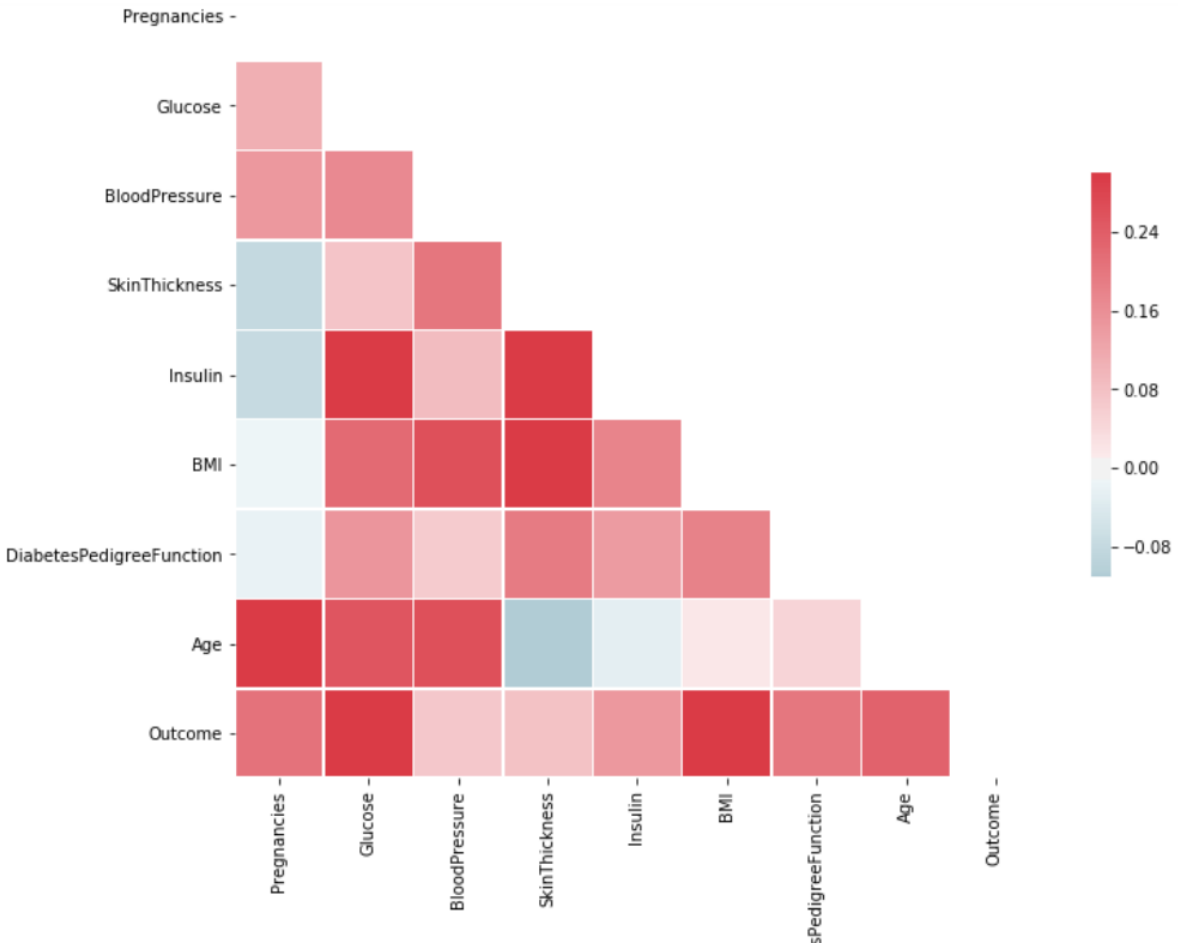
Exploratory Data Analysis

Will start with Importing the libraries and csv file.

```
import pandas as pd
import numpy as np
import sklearn
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
import datetime
import calendar
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
from matplotlib.pylab import rcParams
import seaborn as sns
```

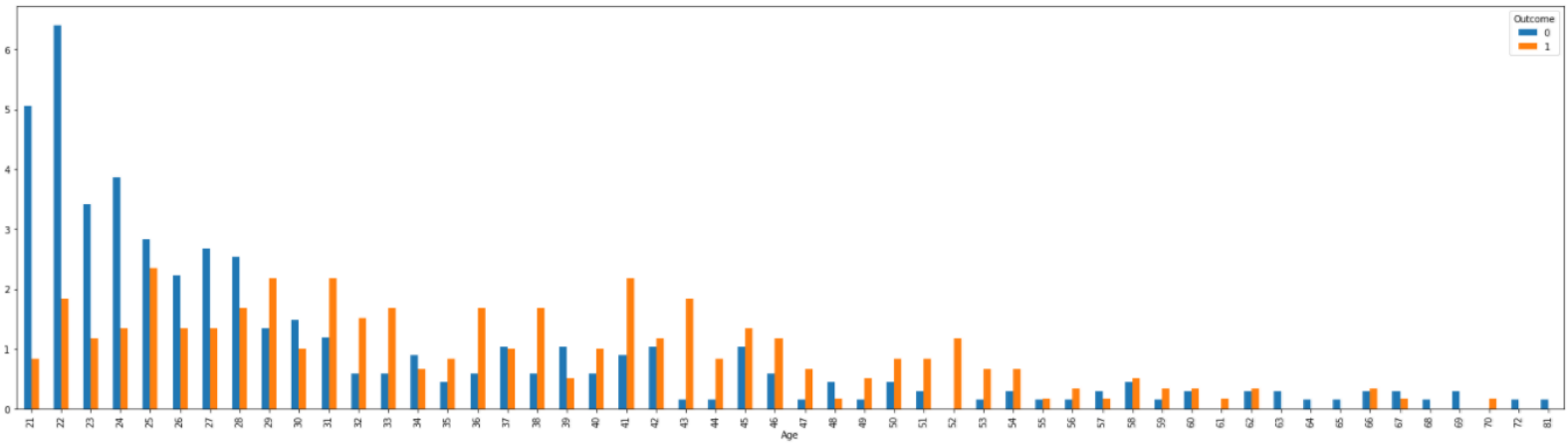
```
df = pd.read_csv('diabetes1.csv')
```

Now we find the correlation between different columns in our dataset.



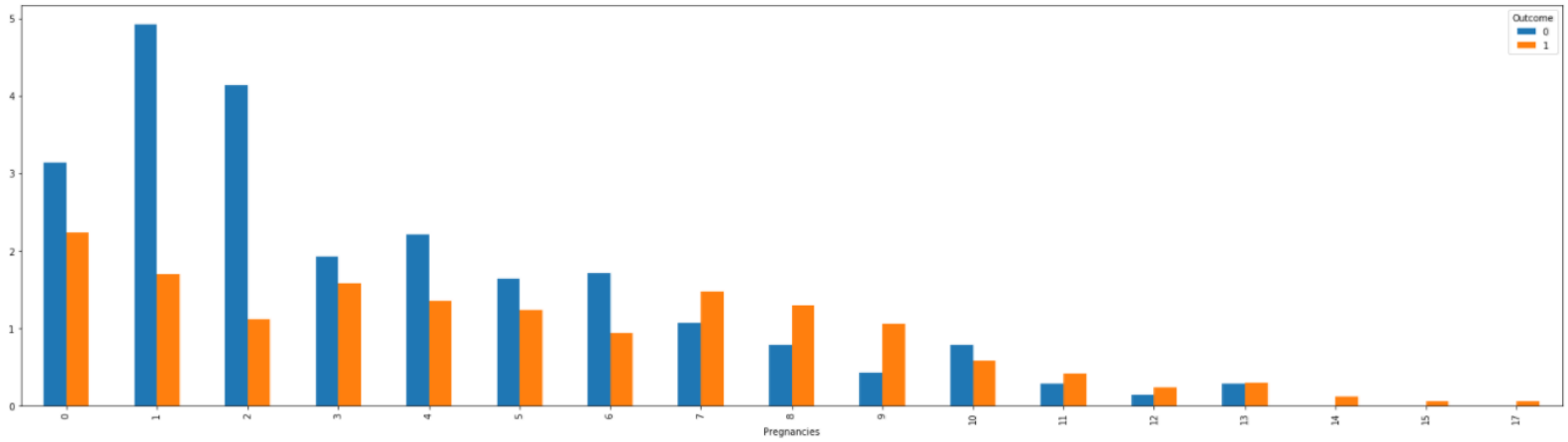
We can see that ‘glucose’ have high correlation between ‘age’ , ‘insulin’ and ‘bmi’. ‘age’ have high correlation with ‘pregnancies’ and ‘BloodPressure’. ‘skinThickness’ have high correlation with ‘DiabetesPedigreeFunction’ , ‘bmi’ and ‘insulin’.

Now let's find out who are patient or not with respect to age



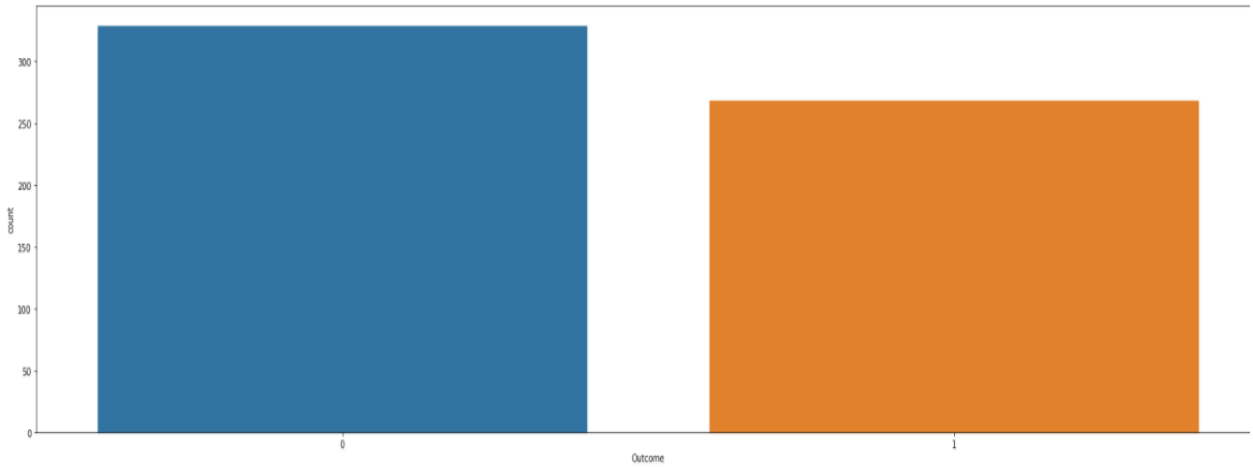
We can see that in our dataset, frequency of data of younger generation is high as compared to older age data.

Now let's find out how pregnancy affects diabetes outcome.



We have 100% data of patients having diabetes with 14 or more pregnancies.

Let's see how our 'y-axis' is distributed.

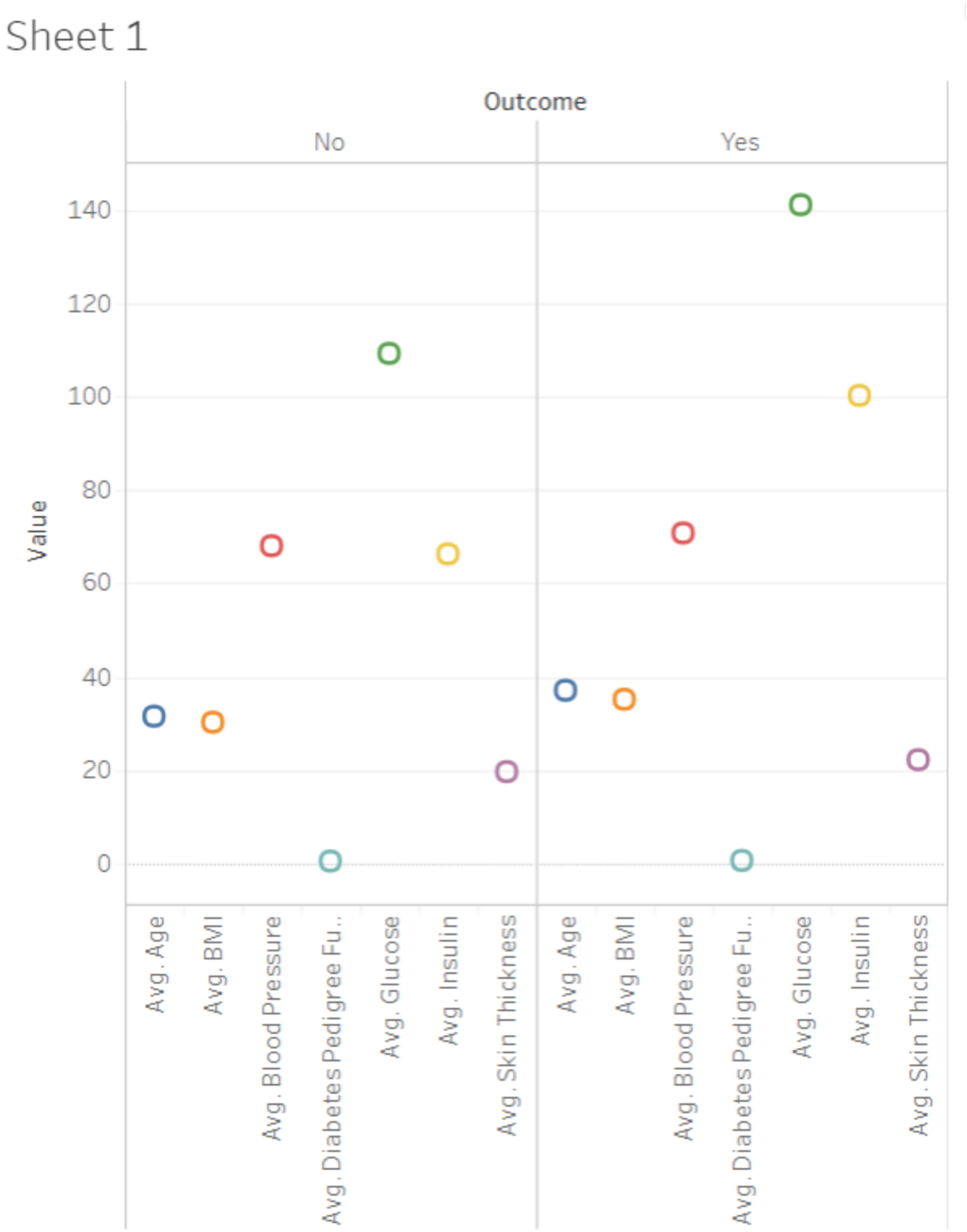



```
df['Outcome'].value_counts()
```

```
0    329
1    268
Name: Outcome, dtype: int64
```

There are 329 entries with no diabetes and 268 entries with diabetes in our dataset.

Now let’s see how numeric data is distributed.


Sheet 1
Outcome
No
Yes
Value
140
120
100
80
60
40
20
0
Avg. Age
Avg. BMI
Avg. Blood Pressure
Avg. Diabetes Pedigree Fu..
Avg. Glucose
Avg. Insulin
Avg. Skin Thickness
Avg. Age
Avg. BMI
Avg. Blood Pressure
Avg. Diabetes Pedigree Fu..
Avg. Glucose
Avg. Insulin
Avg. Skin Thickness

	Outcome = No	Outcome = Yes
Average Age	31.5	37.5
Average BMI	30.2	35.1
Average Blood Pressure	68.1	70.8
Average Diabetes Pedigree	0.4	0.6
Average Glucose	109.4	141.3
Average Insulin	66.3	100.3
Average Skin Thickness	196	22.2

From the above table, we can understand that average skin thickness is more for non-diabetes patients whereas average age, average bmi, average blood pressure, average diabetes pedigree, average glucose and average insulin is more for diabetes patients.

The above graph is drawn in tableau.

Data Cleaning

Let’s start with importing the libraries and dataset.

```
import pandas as pd
import numpy as np
import datetime
import sklearn
from sklearn.cross_validation import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import RandomizedSearchCV
from sklearn.grid_search import GridSearchCV
from sklearn import linear_model
from sklearn.metrics import *
from sklearn.neural_network import MLPRegressor
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

```
df = pd.read_csv('diabetes.csv')
```

Let's find out whether our data contains any missing data or not.

```
df.isnull().sum()
```

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

There are no missing data in our dataset.

Let's find out what are different types of data in our dataset.

```
df.dtypes
```

```
Pregnancies                   int64
Glucose                       int64
BloodPressure                 int64
SkinThickness                 int64
Insulin                       int64
BMI                         float64
DiabetesPedigreeFunction    float64
Age                           int64
Outcome                       int64
dtype: object
```

All data are in numeric form.

There is no need for data cleaning as the dataset is in perfect condition. It does not have any missing data and all the columns have numeric data.

Feature Engineering

From the below code, we have run to find the we are trying to find the key features in our random forest classifier.

```
feature_list = list(x_train.columns)
importances = list(rfc.feature_importances_)
feature_importances = [(x_train, round(importance, 2)) for x_train, importance in zip(feature_list, importances)]
feature_importances = sorted(feature_importances, key = lambda x: x[1], reverse = True)
[print('Variable: {:20} Importance: {}'.format(*pair)) for pair in feature_importances];
```

```
Variable: Glucose              Importance: 0.23
Variable: BMI                  Importance: 0.19
Variable: Age                  Importance: 0.14
Variable: DiabetesPedigreeFunction Importance: 0.12
Variable: Pregnancies          Importance: 0.08
Variable: BloodPressure        Importance: 0.08
Variable: SkinThickness        Importance: 0.08
Variable: Insulin              Importance: 0.08
```

We are removing the last placed column and making a new model with it.

```
col_list=['Pregnancies','Glucose','BloodPressure','SkinThickness','BMI','DiabetesPedigreeFunction','Age']
df_train,df_test = train_test_split(df,train_size=0.7,random_state=42)
x_train=df_train[col_list]
y_train=df_train['Outcome']
scaler.fit(x_train)
x_train_sc=scaler.transform(x_train)
x_test=df_test[col_list]
y_test=df_test['Outcome']
scaler.fit(x_test)
x_test_sc=scaler.transform(x_test)
print(x_train.shape,y_train.shape)
```

```
(417, 7) (417,)
```

```
feature_list = list(x_train.columns)
importances = list(rfc.feature_importances_)
feature_importances = [(x_train, round(importance, 2)) for x_train, importance in zip(feature_list, importances)]
feature_importances = sorted(feature_importances, key = lambda x: x[1], reverse = True)
[print('Variable: {:20} Importance: {}'.format(*pair)) for pair in feature_importances];
```

```
Variable: Glucose              Importance: 0.28
Variable: BMI                  Importance: 0.19
Variable: Age                  Importance: 0.15
Variable: DiabetesPedigreeFunction Importance: 0.12
Variable: BloodPressure        Importance: 0.09
Variable: SkinThickness        Importance: 0.09
Variable: Pregnancies          Importance: 0.08
```

Now this model will be used to run algorithms.

Algorithm

We are using the same model.

```
col_list=['Pregnancies','Glucose','BloodPressure','SkinThickness','BMI','DiabetesPedigreeFunction','Age']
df_train,df_test = train_test_split(df,train_size=0.7,random_state=42)
x_train=df_train[col_list]
y_train=df_train['Outcome']
scaler.fit(x_train)
x_train_sc=scaler.transform(x_train)
x_test=df_test[col_list]
y_test=df_test['Outcome']
scaler.fit(x_test)
x_test_sc=scaler.transform(x_test)
print(x_train.shape,y_train.shape)
```

```
(417, 7) (417,)
```

There are few features like accuracy, f1 score, etc. which we are retrieving from the model after running algorithm. So, we are creating new lists.

```
accuracy =[]
model_name =[]
dataset=[]
f1score = []
precision = []
recall = []
true_positive =[]
false_positive =[]
true_negative =[]
false_negative =[]
```

Logistic Regression

```
logreg = LogisticRegression()
## fitiing the model
logreg.fit(x_train_sc, y_train)
filename = 'logReg_model.sav'
pickle.dump(logreg,open(filename,'wb'))
logreg
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

Results for training data.

Training

```
prediction = logreg.predict(x_train_sc)
f1 = f1_score(y_train, prediction)
p = precision_score(y_train, prediction)
r = recall_score(y_train, prediction)
a = accuracy_score(y_train, prediction)
cm = confusion_matrix(y_train, prediction)
tp = cm[0][0]
fp = cm[0][1]
fn = cm[1][0]
tn = cm[1][1]
dataset.append('Training')
model_name.append('Logistic Regression')
f1score.append(f1)
precision.append(p)
recall.append(r)
accuracy.append(a)
true_positive.append(tp)
false_positive.append(fp)
true_negative.append(tn)
false_negative.append(fn)
cm
```

```
array([[192,  39],
       [ 62, 124]], dtype=int64)
```

Results for testing data.

Testing

```
prediction = logreg.predict(x_test_sc)
f1 = f1_score(y_test,  prediction)
p = precision_score(y_test,  prediction)
r = recall_score(y_test,  prediction)
a = accuracy_score(y_test,  prediction)
cm = confusion_matrix(y_test,  prediction)
tp = cm[0][0]
fp = cm[0][1]
fn = cm[1][0]
tn = cm[1][1]
model_name.append('Logistc Regression')
dataset.append('Testing')
f1score.append(f1)
precision.append(p)
recall.append(r)
accuracy.append(a)
true_positive.append(tp)
false_positive.append(fp)
true_negative.append(tn)
false_negative.append(fn)
cm
```

```
array([[83, 15],
       [26, 56]], dtype=int64)
```

Random Forest Classifier

```
rfc = RandomForestClassifier(n_estimators=50,random_state=0)
## fitiing the model
rfc.fit(x_train_sc, y_train)
filename = 'RFC_model.sav'
pickle.dump(rfc,open(filename,'wb'))
rfc
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=50, n_jobs=1,
            oob_score=False, random_state=0, verbose=0, warm_start=False)
```

Results for training data.

Training

```
prediction = rfc.predict(x_train_sc)
f1 = f1_score(y_train, prediction)
p = precision_score(y_train, prediction)
r = recall_score(y_train, prediction)
a = accuracy_score(y_train, prediction)
cm = confusion_matrix(y_train, prediction)
tp = cm[0][0]
fp = cm[0][1]
fn = cm[1][0]
tn = cm[1][1]
model_name.append('Random Forest Classifier')
dataset.append('Training')
f1score.append(f1)
precision.append(p)
recall.append(r)
accuracy.append(a)
true_positive.append(tp)
false_positive.append(fp)
true_negative.append(tn)
false_negative.append(fn)
cm
```

```
array([[231,   0],
       [  0, 186]], dtype=int64)
```

Results for testing data.

Testing

```
prediction = rfc.predict(x_test_sc)
f1 = f1_score(y_test,  prediction)
p = precision_score(y_test,  prediction)
r = recall_score(y_test,  prediction)
a = accuracy_score(y_test,  prediction)
cm = confusion_matrix(y_test,  prediction)
tp = cm[0][0]
fp = cm[0][1]
fn = cm[1][0]
tn = cm[1][1]
model_name.append('Random Forest Classifier')
dataset.append('Testing')
f1score.append(f1)
precision.append(p)
recall.append(r)
accuracy.append(a)
true_positive.append(tp)
false_positive.append(fp)
true_negative.append(tn)
false_negative.append(fn)
cm
```

```
array([[76, 22],
       [19, 63]], dtype=int64)
```

Gradient Naïve Bayes

```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(x_train_sc, y_train)
filename = 'GNB_model.sav'
pickle.dump(gnb,open(filename,'wb'))
gnb
```

```
GaussianNB(priors=None)
```

Results for training data.

Training

```
prediction = gnb.predict(x_train_sc)
f1 = f1_score(y_train, prediction)
p = precision_score(y_train, prediction)
r = recall_score(y_train, prediction)
a = accuracy_score(y_train, prediction)
cm = confusion_matrix(y_train, prediction)
tp = cm[0][0]
fp = cm[0][1]
fn = cm[1][0]
tn = cm[1][1]
model_name.append('Gradient Naive Bayes')
dataset.append('Training')
f1score.append(f1)
precision.append(p)
recall.append(r)
accuracy.append(a)
true_positive.append(tp)
false_positive.append(fp)
true_negative.append(tn)
false_negative.append(fn)
cm
```

```
array([[187,  44],
       [ 61, 125]], dtype=int64)
```

Results for testing data.

Testing

```
prediction = gnb.predict(x_test_sc)
f1 = f1_score(y_test,  prediction)
p = precision_score(y_test,  prediction)
r = recall_score(y_test,  prediction)
a = accuracy_score(y_test,  prediction)
cm = confusion_matrix(y_test,  prediction)
tp = cm[0][0]
fp = cm[0][1]
fn = cm[1][0]
tn = cm[1][1]
model_name.append('Gradient Naive Bayes')
dataset.append('Testing')
f1score.append(f1)
precision.append(p)
recall.append(r)
accuracy.append(a)
true_positive.append(tp)
false_positive.append(fp)
true_negative.append(tn)
false_negative.append(fn)
cm
```

```
array([[81, 17],
       [29, 53]], dtype=int64)
```

Bernoulli Naïve Bayes

```
NB = BernoulliNB()
## fitiing the model
NB.fit(x_train_sc, y_train)
filename = 'BernoulliNB_model.sav'
pickle.dump(NB,open(filename,'wb'))
NB
```

```
BernoulliNB(alpha=1.0, binarize=0.0, class_prior=None, fit_prior=True)
```

Results for training data.

Training

```
prediction = NB.predict(x_train_sc)
f1 = f1_score(y_train, prediction)
p = precision_score(y_train, prediction)
r = recall_score(y_train, prediction)
a = accuracy_score(y_train, prediction)
cm = confusion_matrix(y_train, prediction)
tp = cm[0][0]
fp = cm[0][1]
fn = cm[1][0]
tn = cm[1][1]
model_name.append('Bernoulli Naive Bayes')
dataset.append('Training')
f1score.append(f1)
precision.append(p)
recall.append(r)
accuracy.append(a)
true_positive.append(tp)
false_positive.append(fp)
true_negative.append(tn)
false_negative.append(fn)
cm
```

```
array([[164,  67],
       [ 62, 124]], dtype=int64)
```

Results for testing data.

Testing

```
prediction = NB.predict(x_test_sc)
f1 = f1_score(y_test,  prediction)
p = precision_score(y_test,  prediction)
r = recall_score(y_test,  prediction)
a = accuracy_score(y_test,  prediction)
cm = confusion_matrix(y_test,  prediction)
tp = cm[0][0]
fp = cm[0][1]
fn = cm[1][0]
tn = cm[1][1]
model_name.append('Bernoulli Naive Bayes')
dataset.append('Testing')
f1score.append(f1)
precision.append(p)
recall.append(r)
accuracy.append(a)
true_positive.append(tp)
false_positive.append(fp)
true_negative.append(tn)
false_negative.append(fn)
cm
```

```
array([[75, 23],
       [27, 55]], dtype=int64)
```

Support Vector Classifier

```
from sklearn.svm import SVC
svc = SVC (kernel = 'linear' , C = 0.025 , random_state = 42)
svc.fit(x_train_sc, y_train)
filename = 'SVC_model.sav'
pickle.dump(svc,open(filename,'wb'))
svc
```

```
SVC(C=0.025, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
  max_iter=-1, probability=False, random_state=42, shrinking=True,
  tol=0.001, verbose=False)
```

Results for training data.

Training

```
prediction = svc.predict(x_train_sc)
f1 = f1_score(y_train, prediction)
p = precision_score(y_train, prediction)
r = recall_score(y_train, prediction)
a = accuracy_score(y_train, prediction)
cm = confusion_matrix(y_train, prediction)
tp = cm[0][0]
fp = cm[0][1]
fn = cm[1][0]
tn = cm[1][1]
model_name.append('Support Vector Classifier')
dataset.append('Training')
f1score.append(f1)
precision.append(p)
recall.append(r)
accuracy.append(a)
true_positive.append(tp)
false_positive.append(fp)
true_negative.append(tn)
false_negative.append(fn)
cm
```

```
array([[191,  40],
       [ 63, 123]], dtype=int64)
```

Results for testing data.

Testing

```
prediction = svc.predict(x_test_sc)
f1 = f1_score(y_test,  prediction)
p = precision_score(y_test,  prediction)
r = recall_score(y_test,  prediction)
a = accuracy_score(y_test,  prediction)
cm = confusion_matrix(y_test,  prediction)
tp = cm[0][0]
fp = cm[0][1]
fn = cm[1][0]
tn = cm[1][1]
model_name.append('Support Vector Classifier')
dataset.append('Testing')
f1score.append(f1)
precision.append(p)
recall.append(r)
accuracy.append(a)
true_positive.append(tp)
false_positive.append(fp)
true_negative.append(tn)
false_negative.append(fn)
cm
```

```
array([[83, 15],
       [27, 55]], dtype=int64)
```

Stochastic Gradient Decent

```
from sklearn.linear_model import SGDClassifier
sgd = SGDClassifier (loss = 'modified_huber' , shuffle = True , random_state = 42)
sgd.fit(x_train_sc, y_train)
filename = 'SGD_model.sav'
pickle.dump(sgd,open(filename,'wb'))
sgd
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gradient.p
ameters have been added in <class 'sklearn.linear_model.stochastic_gradient.SGDClassi
ey default to max_iter=5 and tol=None. If tol is not None, max_iter defaults to max_i
be 1000, and default tol will be 1e-3.
  "and default tol will be 1e-3." % type(self), FutureWarning)
```

```
SGDClassifier(alpha=0.0001, average=False, class_weight=None, epsilon=0.1,
       eta0=0.0, fit_intercept=True, l1_ratio=0.15,
       learning_rate='optimal', loss='modified_huber', max_iter=None,
       n_iter=None, n_jobs=1, penalty='l2', power_t=0.5, random_state=42,
       shuffle=True, tol=None, verbose=0, warm_start=False)
```

Results for training data.

Training

```
prediction = sgd.predict(x_train_sc)
f1 = f1_score(y_train, prediction)
p = precision_score(y_train, prediction)
r = recall_score(y_train, prediction)
a = accuracy_score(y_train, prediction)
cm = confusion_matrix(y_train, prediction)
tp = cm[0][0]
fp = cm[0][1]
fn = cm[1][0]
tn = cm[1][1]
model_name.append('Stochastic Gradient Decent')
dataset.append('Training')
f1score.append(f1)
precision.append(p)
recall.append(r)
accuracy.append(a)
true_positive.append(tp)
false_positive.append(fp)
true_negative.append(tn)
false_negative.append(fn)
cm
```

```
array([[157,  74],
       [ 44, 142]], dtype=int64)
```

Results for testing data.

Testing

```
prediction = sgd.predict(x_test_sc)
f1 = f1_score(y_test,  prediction)
p = precision_score(y_test,  prediction)
r = recall_score(y_test,  prediction)
a = accuracy_score(y_test,  prediction)
cm = confusion_matrix(y_test,  prediction)
tp = cm[0][0]
fp = cm[0][1]
fn = cm[1][0]
tn = cm[1][1]
model_name.append('Stochastic Gradient Decent')
dataset.append('Testing')
f1score.append(f1)
precision.append(p)
recall.append(r)
accuracy.append(a)
true_positive.append(tp)
false_positive.append(fp)
true_negative.append(tn)
false_negative.append(fn)
cm
```

```
array([[68, 30],
       [21, 61]], dtype=int64)
```

K Nearest Neighbor

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 2)
knn.fit(x_train_sc, y_train)
filename = 'KNN_model.sav'
pickle.dump(knn,open(filename,'wb'))
knn
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=2, p=2,
           weights='uniform')
```

Results for training data.

Training

```
prediction = knn.predict(x_train_sc)
f1 = f1_score(y_train, prediction)
p = precision_score(y_train, prediction)
r = recall_score(y_train, prediction)
a = accuracy_score(y_train, prediction)
cm = confusion_matrix(y_train, prediction)
tp = cm[0][0]
fp = cm[0][1]
fn = cm[1][0]
tn = cm[1][1]
model_name.append('k nearest Neighbour')
dataset.append('Training')
f1score.append(f1)
precision.append(p)
recall.append(r)
accuracy.append(a)
true_positive.append(tp)
false_positive.append(fp)
true_negative.append(tn)
false_negative.append(fn)
cm
```

```
array([[231,   0],
       [ 55, 131]], dtype=int64)
```

Results for testing data.

Testing

```
prediction = knn.predict(x_test_sc)
f1 = f1_score(y_test,  prediction)
p = precision_score(y_test,  prediction)
r = recall_score(y_test,  prediction)
a = accuracy_score(y_test,  prediction)
cm = confusion_matrix(y_test,  prediction)
tp = cm[0][0]
fp = cm[0][1]
fn = cm[1][0]
tn = cm[1][1]
model_name.append('k nearest Neighbour')
dataset.append('Testing')
f1score.append(f1)
precision.append(p)
recall.append(r)
accuracy.append(a)
true_positive.append(tp)
false_positive.append(fp)
true_negative.append(tn)
false_negative.append(fn)
cm
```

```
array([[82, 16],
       [55, 27]], dtype=int64)
```

Summary Matrix

Initially we are writing all list into one dataframe.

```
Summary = model_name,dataset,f1score,precision,recall,accuracy,true_positive,false_positive,true_negative,false_negative
#Summary
```

```
## Making a dataframe of the accuracy and error metrics
describe1 = pd.DataFrame(Summary[0],columns = {"Model_Name                "})
describe2 = pd.DataFrame(Summary[1],columns = {"Dataset"})
describe3 = pd.DataFrame(Summary[2],columns = {"F1_score"})
describe4 = pd.DataFrame(Summary[3],columns = {"Precision_score"})
describe5 = pd.DataFrame(Summary[4],columns = {"Recall_score"})
describe6 = pd.DataFrame(Summary[5], columns ={"Accuracy_score"})
describe7 = pd.DataFrame(Summary[6], columns ={"True_Positive"})
describe8 = pd.DataFrame(Summary[7], columns ={"False_Positive"})
describe9 = pd.DataFrame(Summary[8], columns ={"True_Negative"})
describe10 = pd.DataFrame(Summary[9], columns ={"False_Negative"})
des = describe1.merge(describe2, left_index=True, right_index=True, how='inner')
des = des.merge(describe3,left_index=True, right_index=True, how='inner')
des = des.merge(describe4,left_index=True, right_index=True, how='inner')
des = des.merge(describe5,left_index=True, right_index=True, how='inner')
des = des.merge(describe6,left_index=True, right_index=True, how='inner')
des = des.merge(describe7,left_index=True, right_index=True, how='inner')
des = des.merge(describe8,left_index=True, right_index=True, how='inner')
des = des.merge(describe9,left_index=True, right_index=True, how='inner')
des = des.merge(describe10,left_index=True, right_index=True, how='inner')
#des = des.merge(describe9,left_index=True, right_index=True, how='inner')
Summary_csv = des.sort_values(ascending=True,by="False_Negative").reset_index(drop = True)
Summary_csv
```

The results are printed in a summary matrix. Screenshot of the same is mentioned below.

Model_Name	Dataset	F1_score	Precision_score	Recall_score	Accuracy_score	True_Positive	False_Positive	True_Negative	False_Negative
Random Forest Classifier	Training	1.000000	1.000000	1.000000	1.000000	231	0	186	0
Random Forest Classifier	Testing	0.766667	0.750000	0.732558	0.768293	75	23	63	19
Stochastic Gradient Decent	Training	0.702638	0.635294	0.701299	0.580645	185	46	108	78
Stochastic Gradient Decent	Testing	0.694444	0.640523	0.690141	0.597561	76	22	49	33
k nearest Neighbour	Training	0.851319	0.834225	0.829787	0.838710	199	32	156	30
k nearest Neighbour	Testing	0.738889	0.700637	0.733333	0.670732	78	20	55	27
Logistic Regression	Training	0.760192	0.712644	0.765432	0.666667	193	38	124	62
Logistic Regression	Testing	0.772222	0.732026	0.788732	0.682927	83	15	56	26
Support Vector Classifier	Training	0.750600	0.699422	0.756250	0.650538	192	39	121	65
Support Vector Classifier	Testing	0.766667	0.723684	0.785714	0.670732	83	15	55	27
Bernoulli Naive Bayes	Training	0.700240	0.668435	0.659686	0.677419	166	65	126	60
Bernoulli Naive Bayes	Testing	0.711111	0.679012	0.687500	0.670732	73	25	55	27
Gradient Naive Bayes	Training	0.733813	0.676385	0.738854	0.623656	190	41	116	70
Gradient Naive Bayes	Testing	0.766667	0.727273	0.777778	0.682927	82	16	56	26

Conclusion: We are concentrating to reduce ‘False_negative’ as much as possible. From the above summary metrics, we can see that ‘Random Forest Classifier’ is giving the best result. So, we will prefer the outcome of this model.

Confusion Matrix

In the field of machine learning and specifically the problem of statistical classification, a confusion matrix, also known as an error matrix,[4] is a specific table layout that allows visualization of the performance of an algorithm, typically a supervised learning one (in unsupervised learning it is usually called a matching matrix). Each row of the matrix represents the instances in a predicted class while each column represents the instances in an actual class (or vice versa).[2] The name stems from the fact that it makes it easy to see if the system is confusing two classes (i.e. commonly mislabeling one as another).

It is a special kind of contingency table, with two dimensions ("actual" and "predicted"), and identical sets of "classes" in both dimensions (each combination of dimension and class is a variable in the contingency table).

		True condition				
	Total population	Condition positive	Condition negative	Prevalence $= \frac{\Sigma \text{ Condition positive}}{\Sigma \text{ Total population}}$	Accuracy (ACC) $= \frac{\Sigma \text{ True positive} + \Sigma \text{ True negative}}{\Sigma \text{ Total population}}$	
Predicted condition	Predicted condition positive	**True positive**, Power	**False positive**, Type I error	Positive predictive value (PPV), Precision $= \frac{\Sigma \text{ True positive}}{\Sigma \text{ Predicted condition positive}}$	False discovery rate (FDR) $= \frac{\Sigma \text{ False positive}}{\Sigma \text{ Predicted condition positive}}$	
	Predicted condition negative	**False negative**, Type II error	**True negative**	False omission rate (FOR) $= \frac{\Sigma \text{ False negative}}{\Sigma \text{ Predicted condition negative}}$	Negative predictive value (NPV) $= \frac{\Sigma \text{ True negative}}{\Sigma \text{ Predicted condition negative}}$	
		True positive rate (TPR), Recall, Sensitivity, probability of detection $= \frac{\Sigma \text{ True positive}}{\Sigma \text{ Condition positive}}$	False positive rate (FPR), Fall-out, probability of false alarm $= \frac{\Sigma \text{ False positive}}{\Sigma \text{ Condition negative}}$	Positive likelihood ratio (LR+) $= \frac{\text{TPR}}{\text{FPR}}$	Diagnostic odds ratio (DOR) $= \frac{\text{LR+}}{\text{LR−}}$	F_1 score $= \frac{2}{\frac{1}{\text{Recall}} + \frac{1}{\text{Precision}}}$
		False negative rate (FNR), Miss rate $= \frac{\Sigma \text{ False negative}}{\Sigma \text{ Condition positive}}$	True negative rate (TNR), Specificity (SPC) $= \frac{\Sigma \text{ True negative}}{\Sigma \text{ Condition negative}}$	Negative likelihood ratio (LR−) $= \frac{\text{FNR}}{\text{TNR}}$		

F1 Score

In statistical analysis of binary classification, the F1 score (also F-score or F-measure) is a measure of a test's accuracy. It considers both the precision p and the recall r of the test to compute the score: p is the number of correct positive results divided by the number of all positive results returned by the classifier, and r is the number of correct positive results divided by the number of all relevant samples (all samples that should have been identified as positive). The F1 score is the harmonic average of the precision and recall, where an F1 score reaches its best value at 1 (perfect precision and recall) and worst at 0.

Precision Score

In pattern recognition, information retrieval and binary classification, precision (also called positive predictive value) is the fraction of relevant instances among the retrieved instances, while recall (also known as sensitivity) is the fraction of relevant instances that have been retrieved over the total amount of relevant instances. Both precision and recall are therefore based on an understanding and measure of relevance.

Recall Score

Sensitivity and specificity are statistical measures of the performance of a binary classification test, also known in statistics as classification function:

- Sensitivity (also called the true positive rate, the recall, or probability of detection in some fields) measures the proportion of positives that are correctly identified as such (e.g. the percentage of sick people who are correctly identified as having the condition).
- Specificity (also called the true negative rate) measures the proportion of negatives that are correctly identified as such (e.g. the percentage of healthy people who are correctly identified as not having the condition).

Accuracy

Precision is a description of random errors, a measure of statistical variability.

Accuracy has two definitions:

1. More commonly, it is a description of systematic errors, a measure of statistical bias; as these cause a difference between a result and a "true" value, ISO calls this trueness.
2. Alternatively, ISO defines accuracy as describing a combination of both types of observational error above (random and systematic), so high accuracy requires both high precision and high trueness.

In simplest terms, given a set of data points from repeated measurements of the same quantity, the set can be said to be precise if the values are close to each other, while the set can be said to be accurate if their average is close to the true value of the quantity being measured. The two concepts are independent of each other, so a set of data can be said to be either accurate, or precise, or both, or neither.

Docker

Command to build docker image:

docker build -t <username>/adsprojecthealthcare .

After build the docker image use following command to run the image

docker run -e ak='accesskey' -e sak='secretaccesskey' -p 5000:80 <username>/adsprojecthealthcare

To push the image image run following command

docker login

(enter your credentials)

docker push <username>/adsprojecthealthcare

Note :

The username we have used is akashj.

AWS

Create an EC2 instance by referring the following link.

https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EC2_GetStarted.html

Once the EC2 instance is created install docker on that instance using following link.

https://docs.aws.amazon.com/AmazonECS/latest/developerguide/docker-basics.html

Then pull the docker image using following command

Docker pull akashj/adsprojecthealthcare

Run the following command

docker run -e ak='accesskey' -e sak='secretaccesskey' -p 5000:80 akashj/adsprojecthealthcare

DNS for our EC2 instance is ec2-52-27-35-131.us-west-2.compute.amazonaws.com

and we have mapped port number 80 to 5000.

Flask

Link to our application:

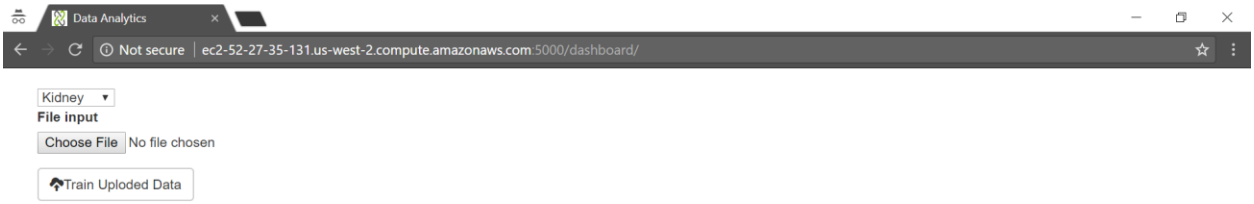
http://ec2-52-27-35-131.us-west-2.compute.amazonaws.com:5000/

Loging in with Admin credentials:

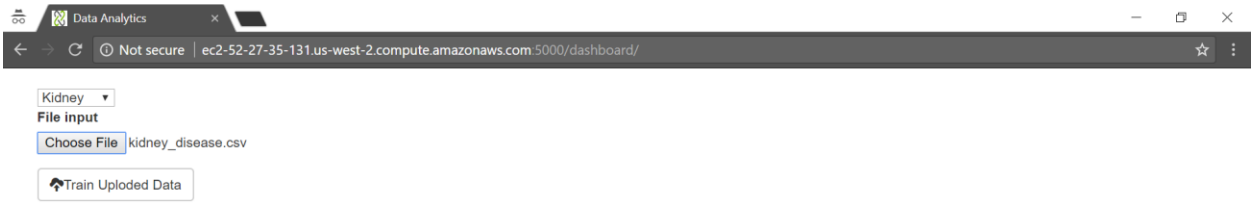
Admin is allowed to train the models for kidney, Liver, Diabetes

Username: admin2

Password: password



Training the models for kidney :



Error metrics after training the models

Kidney

Error Metrics

	Model_Name	Dataset	F1_score	Precision_score	Recall_score	Accuracy_score	True_Positive	False_Positive	True_Negative	False_Negative
0	Random Forest Classifier	Training	1.000000	1.000000	1.000000	1.000000	106	0	174	0
1	Random Forest Classifier	Testing	1.000000	1.000000	1.000000	1.000000	44	0	76	0
2	Stochastic Gradient Decent	Training	0.985714	0.988571	0.982955	0.994253	103	3	173	1
3	Stochastic Gradient Decent	Testing	0.991667	0.993377	1.000000	0.986842	44	0	75	1
4	k nearest Neighbour	Training	0.985714	0.988571	0.982955	0.994253	103	3	173	1
5	k nearest Neighbour	Testing	0.991667	0.993377	1.000000	0.986842	44	0	75	1
6	Logistic Regression	Training	0.985714	0.988439	0.994186	0.982759	105	1	171	3
7	Logistic Regression	Testing	0.991667	0.993377	1.000000	0.986842	44	0	75	1
8	Support Vector Classifier	Training	0.982143	0.985507	0.994152	0.977011	105	1	170	4
9	Support Vector Classifier	Testing	0.991667	0.993377	1.000000	0.986842	44	0	75	1
10	Bernoulli Naive Bayes	Training	0.975000	0.979943	0.977143	0.982759	102	4	171	3
11	Bernoulli Naive Bayes	Testing	0.983333	0.986667	1.000000	0.973684	44	0	74	2
12	Gradient Naive Bayes	Training	0.935714	0.945455	1.000000	0.896552	106	0	156	18
13	Gradient Naive Bayes	Testing	0.633333	0.775510	0.633333	1.000000	0	44	76	0

Training models for liver

Kidney

Error Metrics

	Model_Name	Dataset	F1_score	Precision_score	Recall_score	Accuracy_score	True_Positive	False_Positive	True_Negative	False_Negative
0	Random Forest Classifier	Training	1.000000	1.000000	1.000000	1.000000	106	0	174	0
1	Random Forest Classifier	Testing	1.000000	1.000000	1.000000	1.000000	44	0	76	0
2	Stochastic Gradient Decent	Training	0.985714	0.988571	0.982955	0.994253	103	3	173	1
3	Stochastic Gradient Decent	Testing	0.991667	0.993377	1.000000	0.986842	44	0	75	1
4	k nearest Neighbour	Training	0.985714	0.988571	0.982955	0.994253	103	3	173	1
5	k nearest Neighbour	Testing	0.991667	0.993377	1.000000	0.986842	44	0	75	1
6	Logistic Regression	Training	0.985714	0.988439	0.994186	0.982759	105	1	171	3
7	Logistic Regression	Testing	0.991667	0.993377	1.000000	0.986842	44	0	75	1
8	Support Vector Classifier	Training	0.982143	0.985507	0.994152	0.977011	105	1	170	4
9	Support Vector Classifier	Testing	0.991667	0.993377	1.000000	0.986842	44	0	75	1
10	Bernoulli Naive Bayes	Training	0.975000	0.979943	0.977143	0.982759	102	4	171	3
11	Bernoulli Naive Bayes	Testing	0.983333	0.986667	1.000000	0.973684	44	0	74	2
12	Gradient Naive Bayes	Training	0.935714	0.945455	1.000000	0.896552	106	0	156	18
13	Gradient Naive Bayes	Testing	0.633333	0.775510	0.633333	1.000000	0	44	76	0

Error Metrics after training the models for liver

Liver

Error Metrics

	Model_Name	Dataset	F1_score	Precision_score	Recall_score	Accuracy_score	True_Positive	False_Positive	True_Negative	False_Negative
0	Random Forest Classifier	Training	1.000000	1.000000	1.000000	1.000000	145	0	119	0
1	Random Forest Classifier	Testing	0.596491	0.540000	0.519231	0.562500	41	25	27	21
2	Stochastic Gradient Decent	Training	0.700758	0.590674	0.770270	0.478992	128	17	57	62
3	Stochastic Gradient Decent	Testing	0.631579	0.400000	0.636364	0.291667	58	8	14	34
4	k nearest Neighbour	Training	0.829545	0.817814	0.789062	0.848739	118	27	101	18
5	k nearest Neighbour	Testing	0.561404	0.528302	0.482759	0.583333	36	30	28	20
6	Logistic Regression	Training	0.753788	0.754717	0.684932	0.840336	99	46	100	19
7	Logistic Regression	Testing	0.614035	0.576923	0.535714	0.625000	40	26	30	18
8	Support Vector Classifier	Training	0.712121	0.737931	0.625731	0.899160	81	64	107	12
9	Support Vector Classifier	Testing	0.657895	0.672269	0.563380	0.833333	35	31	40	8
10	Bernoulli Naive Bayes	Training	0.734848	0.755245	0.646707	0.907563	86	59	108	11
11	Bernoulli Naive Bayes	Testing	0.578947	0.600000	0.500000	0.750000	30	36	36	12
12	Gradient Naive Bayes	Training	0.704545	0.748387	0.607330	0.974790	70	75	116	3
13	Gradient Naive Bayes	Testing	0.578947	0.619048	0.500000	0.812500	27	39	39	9

Training models for diabetes

Liver

Error Metrics

	Model_Name	Dataset	F1_score	Precision_score	Recall_score	Accuracy_score	True_Positive	False_Positive	True_Negative	False_Negative
0	Random Forest Classifier	Training	1.000000	1.000000	1.000000	1.000000	145	0	119	0
1	Random Forest Classifier	Testing	0.596491	0.540000	0.519231	0.562500	41	25	27	21
2	Stochastic Gradient Decent	Training	0.700758	0.590674	0.770270	0.478992	128	17	57	62
3	Stochastic Gradient Decent	Testing	0.631579	0.400000	0.636364	0.291667	58	8	14	34
4	k nearest Neighbour	Training	0.829545	0.817814	0.789062	0.848739	118	27	101	18
5	k nearest Neighbour	Testing	0.561404	0.528302	0.482759	0.583333	36	30	28	20
6	Logistic Regression	Training	0.753788	0.754717	0.684932	0.840336	99	46	100	19
7	Logistic Regression	Testing	0.614035	0.576923	0.535714	0.625000	40	26	30	18
8	Support Vector Classifier	Training	0.712121	0.737931	0.625731	0.899160	81	64	107	12
9	Support Vector Classifier	Testing	0.657895	0.672269	0.563380	0.833333	35	31	40	8
10	Bernoulli Naive Bayes	Training	0.734848	0.755245	0.646707	0.907563	86	59	108	11
11	Bernoulli Naive Bayes	Testing	0.578947	0.600000	0.500000	0.750000	30	36	36	12
12	Gradient Naive Bayes	Training	0.704545	0.748387	0.607330	0.974790	70	75	116	3
13	Gradient Naive Bayes	Testing	0.578947	0.619048	0.500000	0.812500	27	39	39	9

Error metrics after training models for diabetes

Diab

Error Metrics

	Model_Name	Dataset	F1_score	Precision_score	Recall_score	Accuracy_score	True_Positive	False_Positive	True_Negative	False_Negative
0	Random Forest Classifier	Training	1.000000	1.000000	1.000000	1.000000	231	0	186	0
1	Random Forest Classifier	Testing	0.766667	0.750000	0.732558	0.768293	75	23	63	19
2	Stochastic Gradient Decent	Training	0.702638	0.635294	0.701299	0.580645	185	46	108	78
3	Stochastic Gradient Decent	Testing	0.694444	0.640523	0.690141	0.597561	76	22	49	33
4	k nearest Neighbour	Training	0.851319	0.834225	0.829787	0.838710	199	32	156	30
5	k nearest Neighbour	Testing	0.738889	0.700637	0.733333	0.670732	78	20	55	27
6	Logistic Regression	Training	0.760192	0.712644	0.765432	0.666667	193	38	124	62
7	Logistic Regression	Testing	0.772222	0.732026	0.788732	0.682927	83	15	56	26
8	Support Vector Classifier	Training	0.750600	0.699422	0.756250	0.650538	192	39	121	65
9	Support Vector Classifier	Testing	0.766667	0.723684	0.785714	0.670732	83	15	55	27
10	Bernoulli Naive Bayes	Training	0.700240	0.668435	0.659686	0.677419	166	65	126	60
11	Bernoulli Naive Bayes	Testing	0.711111	0.679012	0.687500	0.670732	73	25	55	27
12	Gradient Naive Bayes	Training	0.733813	0.676385	0.738854	0.623656	190	41	116	70
13	Gradient Naive Bayes	Testing	0.766667	0.727273	0.777778	0.682927	82	16	56	26

Loging-in as a user.

Username: user2

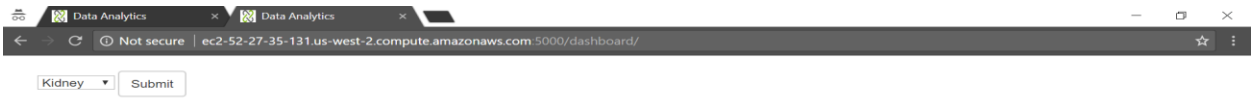
Password: password

User is allowed to upload the data and get the classified output of that data.

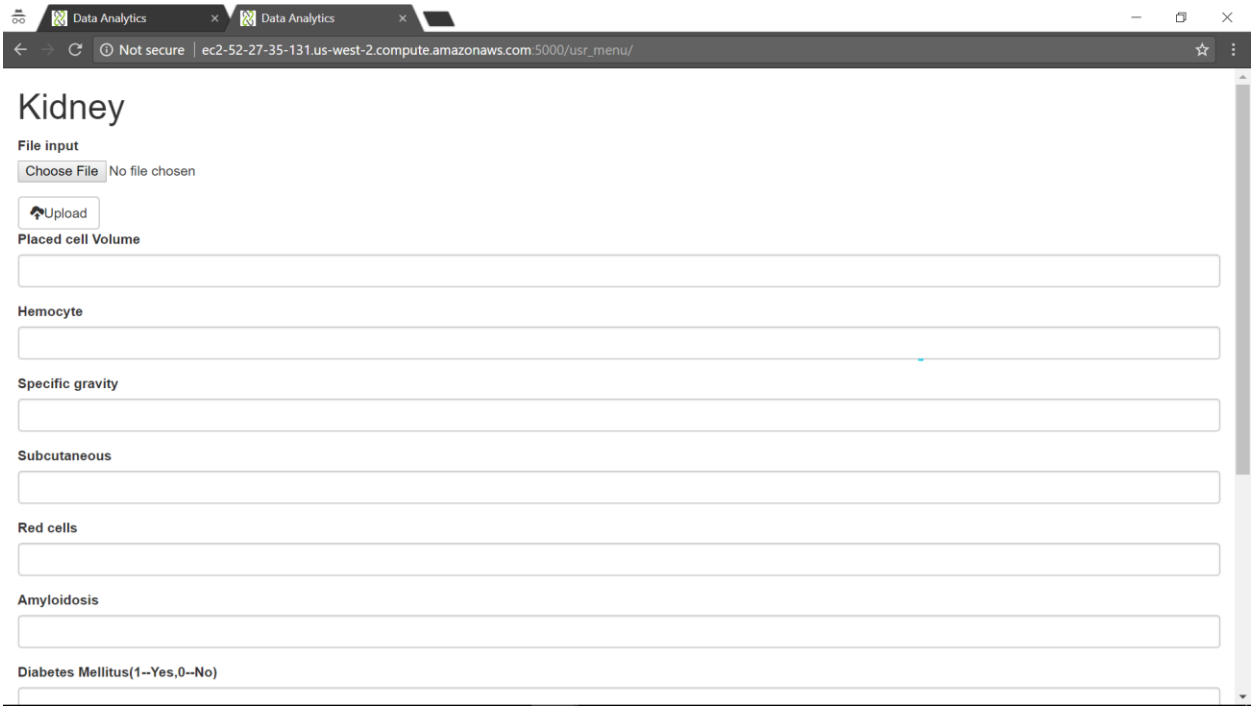
For single entry data can be entered in the form to get the classified output of that data

For bulk data upload a csv file to get the predicted output of that data.

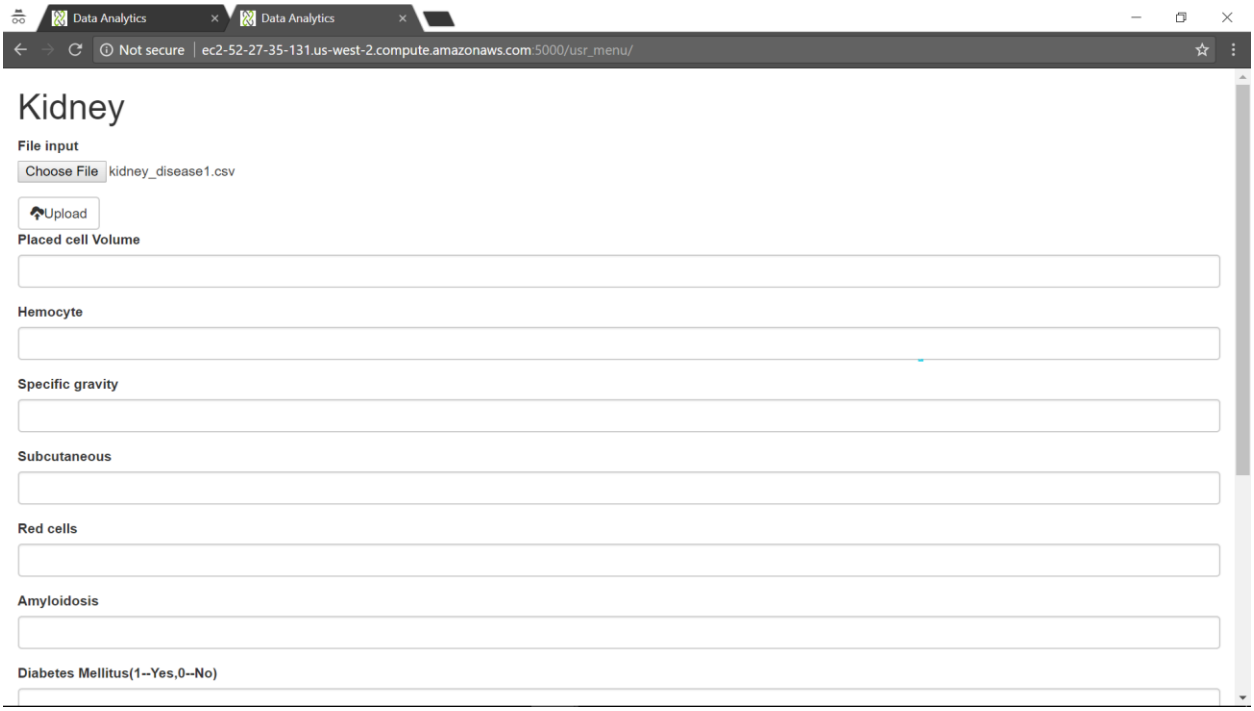
From the drop down select the type on which classification needs to be performed



For Kidney:



Uploading bulk data through a csv file



Predicted O/p with error metrics of the models that were trained by admin

Data Analytics | Data Analytics

Not secure | ec2-52-27-35-131.us-west-2.compute.amazonaws.com:5000/upload/

Kidney

Download-File

Error Metrics

	Unnamed: 0	Model_Name	Dataset	F1_score	Precision_score	Recall_score	Accuracy_score	True_Positive	False_Positive	True_Negative	False_Negative
0	0	Random Forest Classifier	Training	1.000000	1.000000	1.000000	1.000000	106	0	174	0
1	1	Random Forest Classifier	Testing	1.000000	1.000000	1.000000	1.000000	44	0	76	0
2	2	Stochastic Gradient Decent	Training	0.985714	0.988571	0.982955	0.994253	103	3	173	1
3	3	Stochastic Gradient Decent	Testing	0.991667	0.993377	1.000000	0.986842	44	0	75	1
4	4	k nearest Neighbour	Training	0.985714	0.988571	0.982955	0.994253	103	3	173	1
5	5	k nearest Neighbour	Testing	0.991667	0.993377	1.000000	0.986842	44	0	75	1
6	6	Logistic Regression	Training	0.985714	0.988439	0.994186	0.982759	105	1	171	3
7	7	Logistic Regression	Testing	0.991667	0.993377	1.000000	0.986842	44	0	75	1
8	8	Support Vector Classifier	Training	0.982143	0.985507	0.994152	0.977011	105	1	170	4
9	9	Support Vector Classifier	Testing	0.991667	0.993377	1.000000	0.986842	44	0	75	1
10	10	Bernoulli Naive Bayes	Training	0.975000	0.979943	0.977143	0.982759	102	4	171	3
11	11	Bernoulli Naive Bayes	Testing	0.983333	0.986667	1.000000	0.973684	44	0	74	2
12	12	Gradient Naive Bayes	Training	0.935714	0.945455	1.000000	0.896552	106	0	156	18
13	13	Gradient Naive Bayes	Testing	0.633333	0.775510	0.633333	1.000000	0	44	76	0

Predicted O/P

	id	age	bp	sg	al	su	bgr	bu	sc	sod	pot	hemo	pcv	wc	rc	RBC_normal	PC_normal	PCC_present	BA_present	HTN_yes	DM_yes	CAD_yes	Appet_good	PE_yes	A
0	0	48.0	80.0	1.020	1.0	0.0	121.0	36.0	1.200000	137.0	4.26988	15.400000	44.0	7800	5.200000	1	1	0	0	1	1	0	1	0	0
1	1	7.0	50.0	1.020	4.0	0.0	99.0	18.0	0.800000	137.0	4.26988	11.300000	38.0	6000	4.795294	1	1	0	0	0	0	0	1	0	0

Data Analytics | Data Analytics

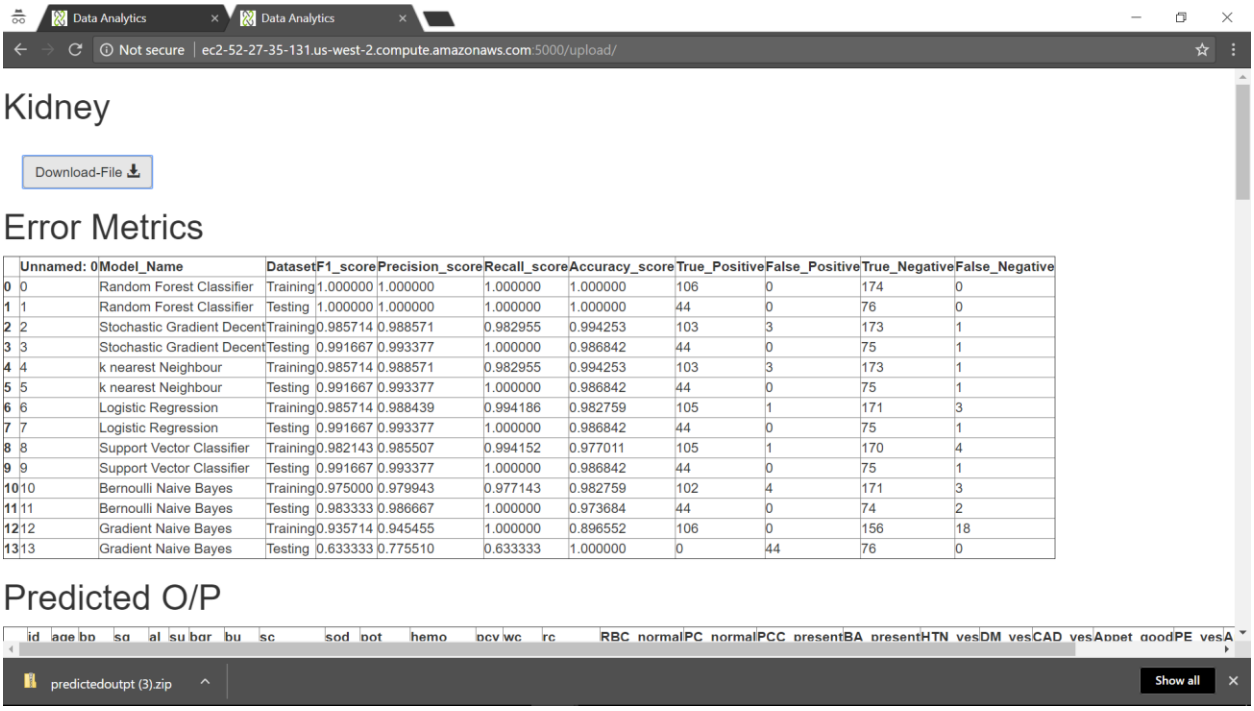
Not secure | ec2-52-27-35-131.us-west-2.compute.amazonaws.com:5000/upload/

ort Vector Classifier	Training	0.982143	0.985507	0.994152	0.977011	105	1	170	4
ort Vector Classifier	Testing	0.991667	0.993377	1.000000	0.986842	44	0	75	1
ulli Naive Bayes	Training	0.975000	0.979943	0.977143	0.982759	102	4	171	3
ulli Naive Bayes	Testing	0.983333	0.986667	1.000000	0.973684	44	0	74	2
ent Naive Bayes	Training	0.935714	0.945455	1.000000	0.896552	106	0	156	18
ent Naive Bayes	Testing	0.633333	0.775510	0.633333	1.000000	0	44	76	0

d O/P

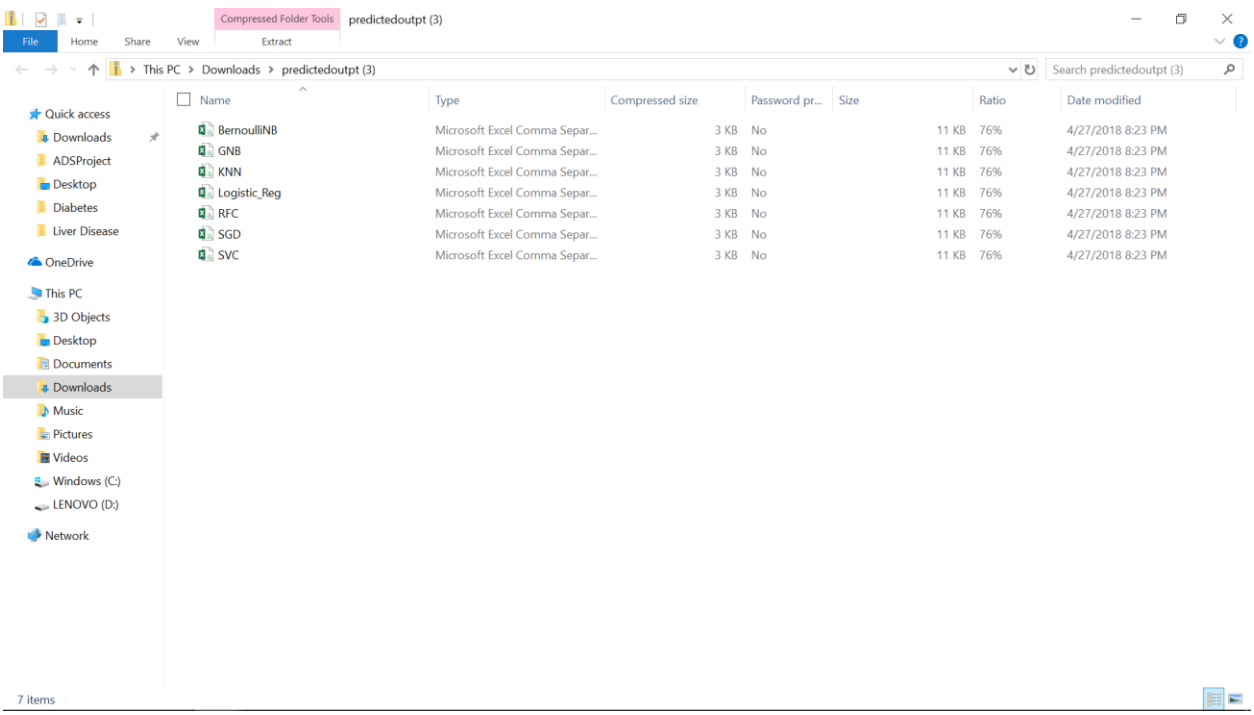
	al	su	bgr	bu	sc	sod	pot	hemo	pcv	wc	rc	RBC_normal	PC_normal	PCC_present	BA_present	HTN_yes	DM_yes	CAD_yes	Appet_good	PE_yes	Ane_yes	Classificatio
20	1.0	0.0	121.0	36.0	1.200000	137.0	4.26988	15.400000	44.0	7800	5.200000	1	1	0	0	1	1	0	1	0	0	ckd
20	4.0	0.0	99.0	18.0	0.800000	137.0	4.26988	11.300000	38.0	6000	4.795294	1	1	0	0	0	0	0	1	0	0	ckd
10	2.0	3.0	423.0	53.0	1.800000	137.0	4.26988	9.600000	31.0	7500	4.795294	1	1	0	0	0	1	0	0	0	1	ckd
05	4.0	0.0	117.0	56.0	3.800000	111.0	2.50000	11.200000	32.0	6700	3.900000	1	0	1	0	1	0	0	0	1	1	ckd
10	2.0	0.0	106.0	26.0	1.400000	137.0	4.26988	11.600000	35.0	7300	4.600000	1	1	0	0	0	0	0	1	0	0	ckd
15	3.0	0.0	74.0	25.0	1.100000	142.0	3.20000	12.200000	39.0	7800	4.400000	1	1	0	0	1	1	0	1	1	0	ckd
10	0.0	0.0	100.0	54.0	24.000000	104.0	4.00000	12.400000	36.0	5800	4.795294	1	1	0	0	0	0	0	1	0	0	ckd
15	2.0	4.0	410.0	31.0	1.100000	137.0	4.26988	12.400000	44.0	6900	5.000000	1	0	0	0	0	1	0	1	1	0	ckd
15	3.0	0.0	138.0	60.0	1.900000	137.0	4.26988	10.800000	33.0	9600	4.000000	1	0	1	0	1	1	0	1	0	1	ckd
20	2.0	0.0	70.0	107.0	7.200000	114.0	3.70000	9.500000	29.0	12100	3.700000	0	0	1	0	1	1	0	0	0	1	ckd
10	2.0	4.0	490.0	55.0	4.000000	137.0	4.26988	9.400000	28.0	5800	4.795294	1	0	1	0	1	1	0	1	0	1	ckd
10	3.0	0.0	380.0	60.0	2.700000	131.0	4.20000	10.800000	32.0	4500	3.800000	0	0	1	0	1	1	0	0	1	0	ckd
15	3.0	1.0	208.0	72.0	2.100000	138.0	5.80000	9.700000	28.0	12200	3.400000	1	1	1	0	1	1	1	0	1	0	ckd
25	0.0	0.0	98.0	86.0	4.600000	135.0	3.40000	9.800000	41.0	5800	4.795294	1	1	0	0	1	1	1	0	1	0	ckd
10	3.0	2.0	157.0	90.0	4.100000	130.0	6.40000	5.600000	16.0	11000	2.600000	1	0	1	1	1	1	1	0	1	0	ckd
15	3.0	0.0	76.0	162.0	9.600000	141.0	4.90000	7.600000	24.0	3800	2.800000	1	1	0	0	1	0	0	1	0	1	ckd
15	2.0	0.0	99.0	46.0	2.200000	138.0	4.10000	12.600000	41.0	5800	4.795294	1	1	0	0	0	0	0	1	0	0	ckd
25	0.0	0.0	114.0	87.0	5.200000	139.0	3.70000	12.100000	41.0	5800	4.795294	1	1	0	0	1	0	0	0	0	0	ckd
25	0.0	3.0	263.0	27.0	1.300000	135.0	4.30000	12.700000	37.0	11400	4.300000	1	1	0	0	1	1	1	1	0	0	ckd
15	1.0	0.0	100.0	31.0	1.600000	137.0	4.26988	10.300000	30.0	5300	3.700000	1	0	1	0	1	0	1	1	0	0	ckd
15	2.0	0.0	173.0	148.0	3.900000	135.0	5.20000	7.700000	24.0	9200	3.200000	0	0	0	0	1	1	1	0	1	1	ckd

Downloading the output files

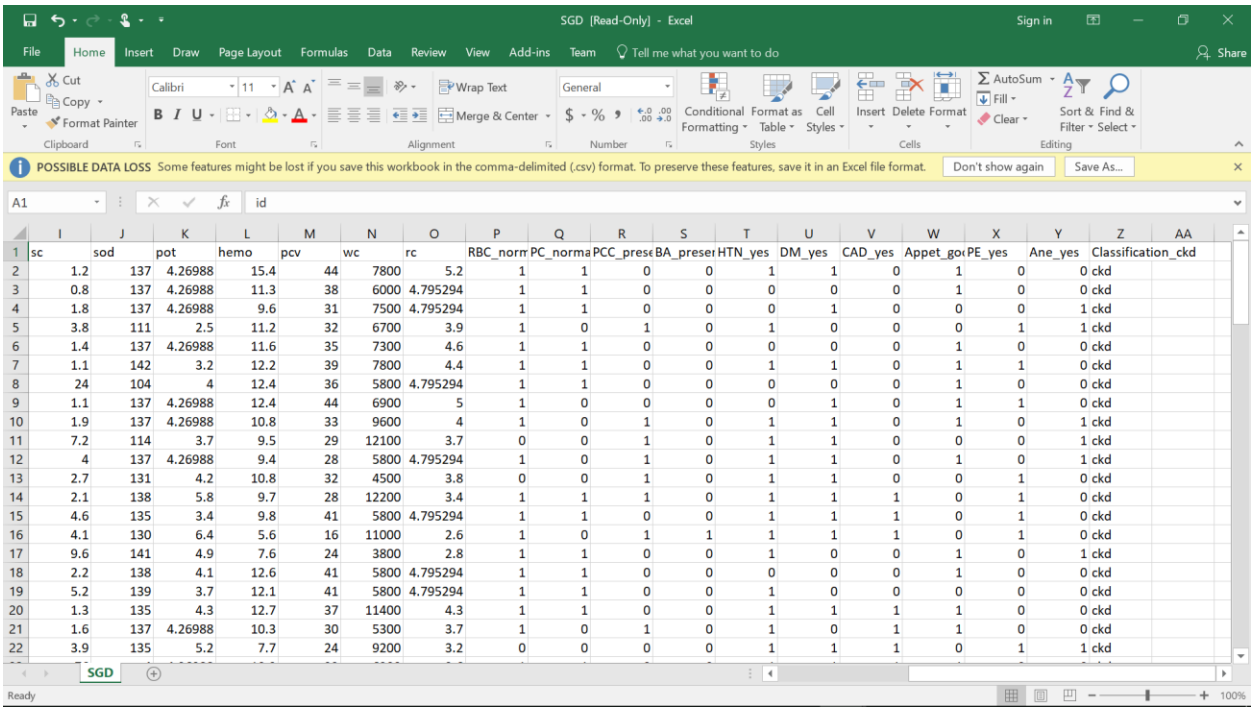


	Unnamed: 0	Model_Name	Dataset	F1_score	Precision_score	Recall_score	Accuracy_score	True_Positive	False_Positive	True_Negative	False_Negative
0	0	Random Forest Classifier	Training	1.000000	1.000000	1.000000	1.000000	106	0	174	0
1	1	Random Forest Classifier	Testing	1.000000	1.000000	1.000000	1.000000	44	0	76	0
2	2	Stochastic Gradient Decent	Training	0.985714	0.988571	0.982955	0.994253	103	3	173	1
3	3	Stochastic Gradient Decent	Testing	0.991667	0.993377	1.000000	0.986842	44	0	75	1
4	4	k nearest Neighbour	Training	0.985714	0.988571	0.982955	0.994253	103	3	173	1
5	5	k nearest Neighbour	Testing	0.991667	0.993377	1.000000	0.986842	44	0	75	1
6	6	Logistic Regression	Training	0.985714	0.988439	0.994186	0.982759	105	1	171	3
7	7	Logistic Regression	Testing	0.991667	0.993377	1.000000	0.986842	44	0	75	1
8	8	Support Vector Classifier	Training	0.982143	0.985507	0.994152	0.977011	105	1	170	4
9	9	Support Vector Classifier	Testing	0.991667	0.993377	1.000000	0.986842	44	0	75	1
10	10	Bernoulli Naive Bayes	Training	0.975000	0.979943	0.977143	0.982759	102	4	171	3
11	11	Bernoulli Naive Bayes	Testing	0.983333	0.986667	1.000000	0.973684	44	0	74	2
12	12	Gradient Naive Bayes	Training	0.935714	0.945455	1.000000	0.896552	106	0	156	18
13	13	Gradient Naive Bayes	Testing	0.633333	0.775510	0.633333	1.000000	0	44	76	0

Output files generated by each model

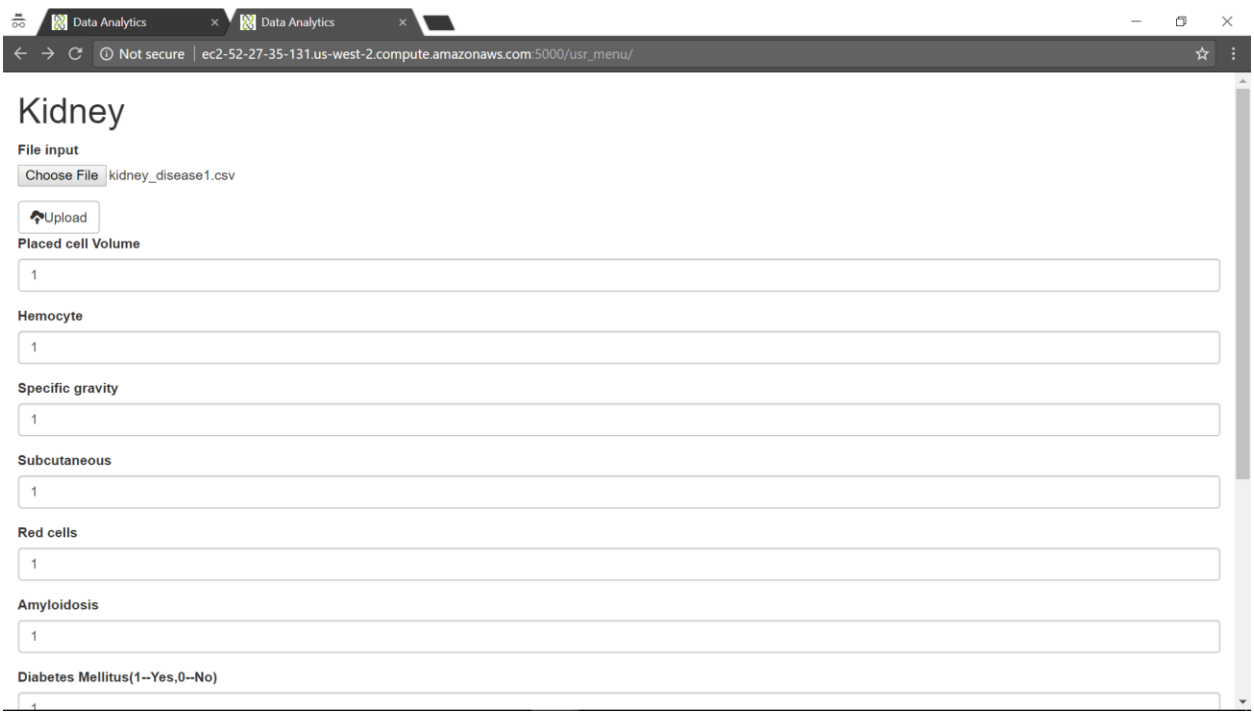


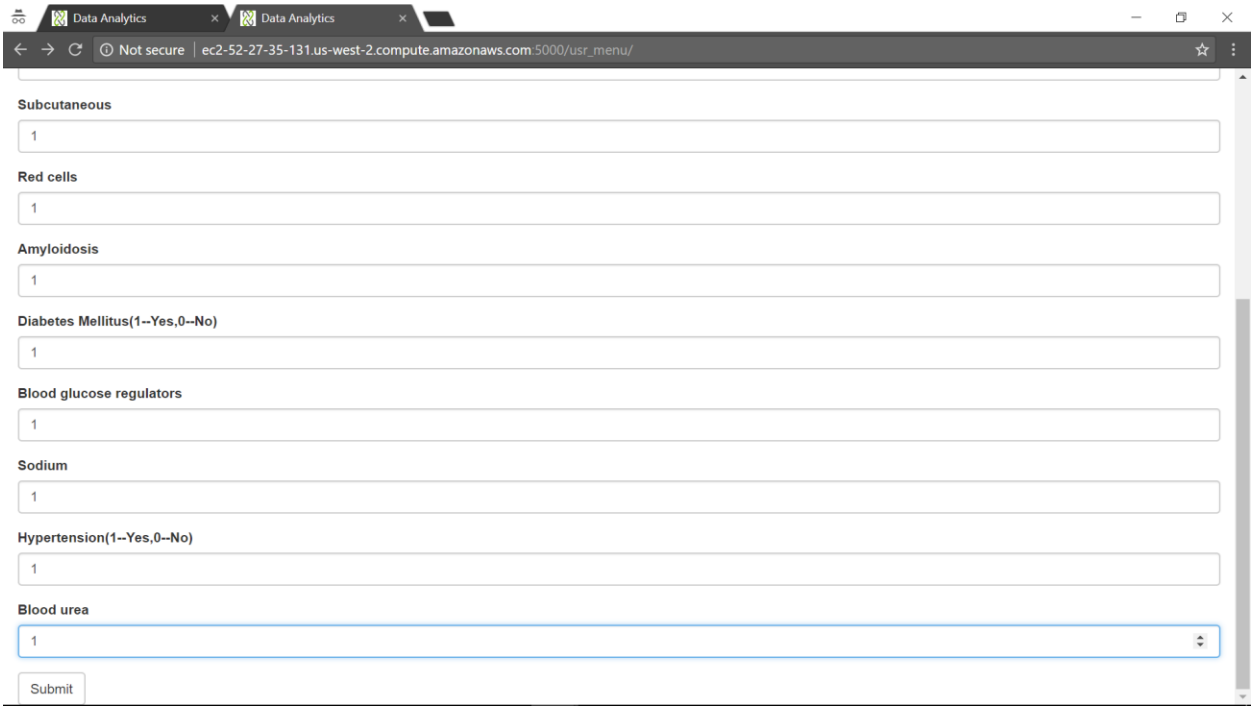
Output file generated by SDG



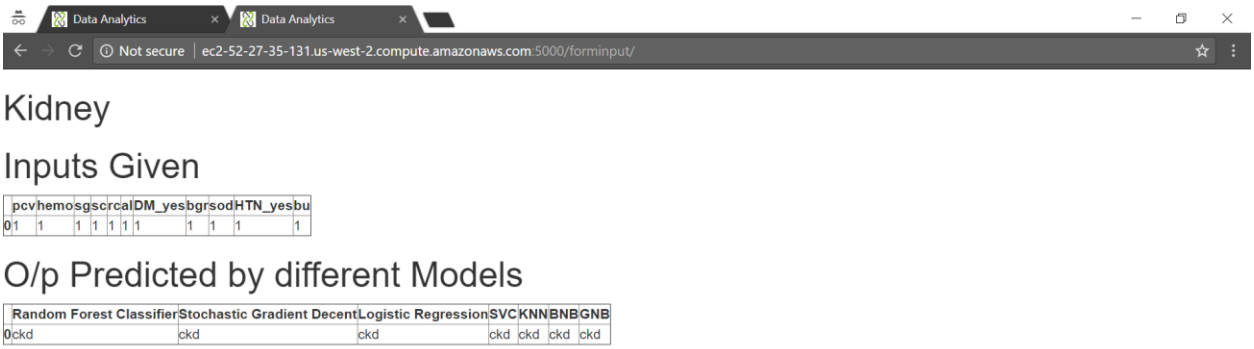
sc	sod	pot	hemo	pcv	wc	rc	RBC_norm	PC_norma	PCC_prese	BA_preser	HTN_yes	DM_yes	CAD_yes	Appet_go	PE_yes	Ane_yes	Classification_ckd
1.2	137	4.26988	15.4	44	7800	5.2	1	1	0	0	1	1	0	1	0	0	ckd
0.8	137	4.26988	11.3	38	6000	4.795294	1	1	0	0	0	0	0	1	0	0	ckd
1.8	137	4.26988	9.6	31	7500	4.795294	1	1	0	0	0	1	0	0	0	1	ckd
3.8	111	2.5	11.2	32	6700	3.9	1	0	1	0	1	0	0	0	1	1	ckd
1.4	137	4.26988	11.6	35	7300	4.6	1	1	0	0	0	0	0	1	0	0	ckd
1.1	142	3.2	12.2	39	7800	4.4	1	1	0	0	1	1	0	1	1	0	ckd
24	104	4	12.4	36	5800	4.795294	1	1	0	0	0	0	0	1	0	0	ckd
1.1	137	4.26988	12.4	44	6900	5	1	0	0	0	0	1	0	1	1	0	ckd
1.9	137	4.26988	10.8	33	9600	4	1	0	1	0	1	1	0	1	0	1	ckd
7.2	114	3.7	9.5	29	12100	3.7	0	0	1	0	1	1	0	0	0	1	ckd
4	137	4.26988	9.4	28	5800	4.795294	1	0	1	0	1	1	0	1	0	1	ckd
2.7	131	4.2	10.8	32	4500	3.8	0	0	1	0	1	1	0	0	1	0	ckd
2.1	138	5.8	9.7	28	12200	3.4	1	1	1	0	1	1	1	0	1	0	ckd
4.6	135	3.4	9.8	41	5800	4.795294	1	1	0	0	1	1	1	0	1	0	ckd
4.1	130	6.4	5.6	16	11000	2.6	1	0	1	1	1	1	1	0	1	0	ckd
9.6	141	4.9	7.6	24	3800	2.8	1	1	0	0	1	0	0	1	0	1	ckd
2.2	138	4.1	12.6	41	5800	4.795294	1	1	0	0	0	0	0	1	0	0	ckd
5.2	139	3.7	12.1	41	5800	4.795294	1	1	0	0	1	0	0	0	0	0	ckd
1.3	135	4.3	12.7	37	11400	4.3	1	1	0	0	1	1	1	1	0	0	ckd
1.6	137	4.26988	10.3	30	5300	3.7	1	0	1	0	1	0	1	1	0	0	ckd
3.9	135	5.2	7.7	24	9200	3.2	0	0	0	0	1	1	1	0	1	1	ckd

Form Data entry





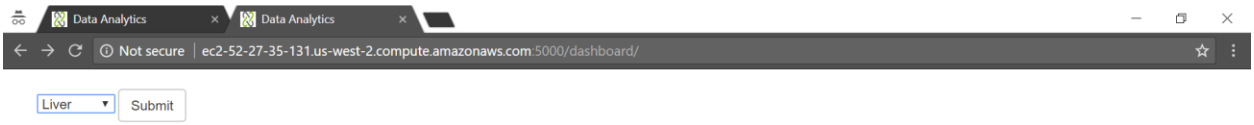
Predicted output of the form data

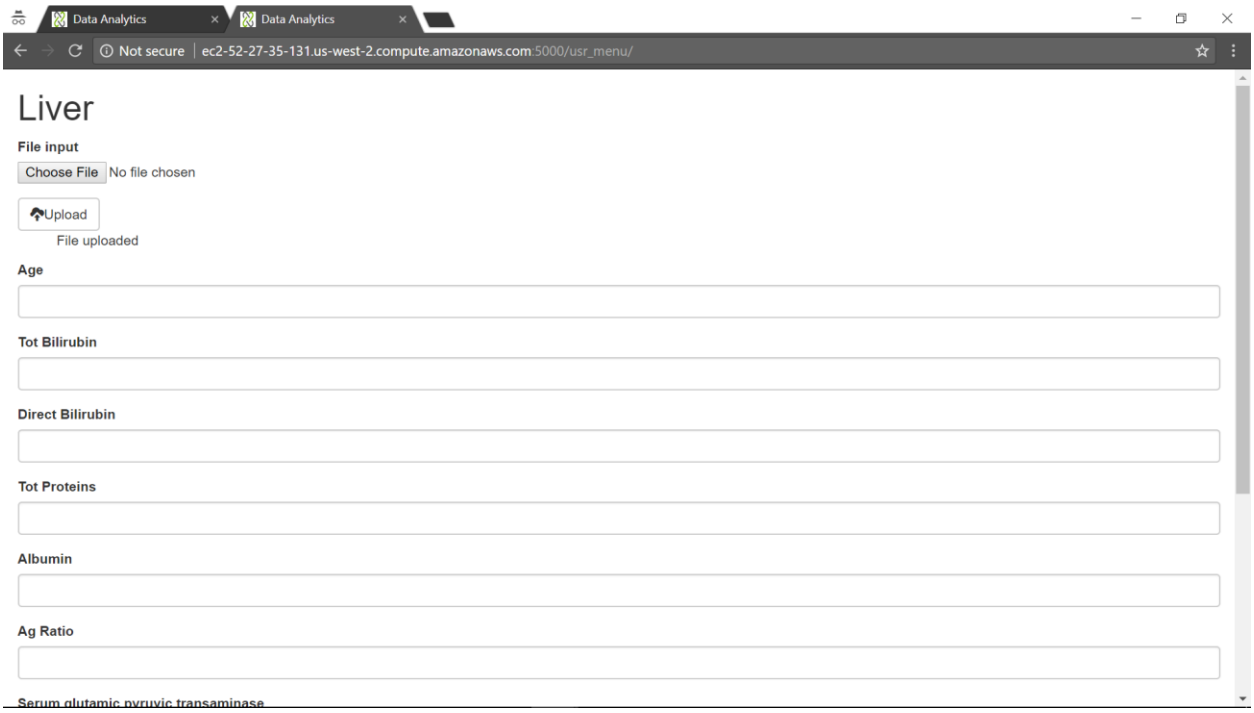


	pcv	hemo	sg	sc	rc	al	DM_yes	bgr	sod	HTN_yes	bu
0	1	1	1	1	1	1	1	1	1	1	1

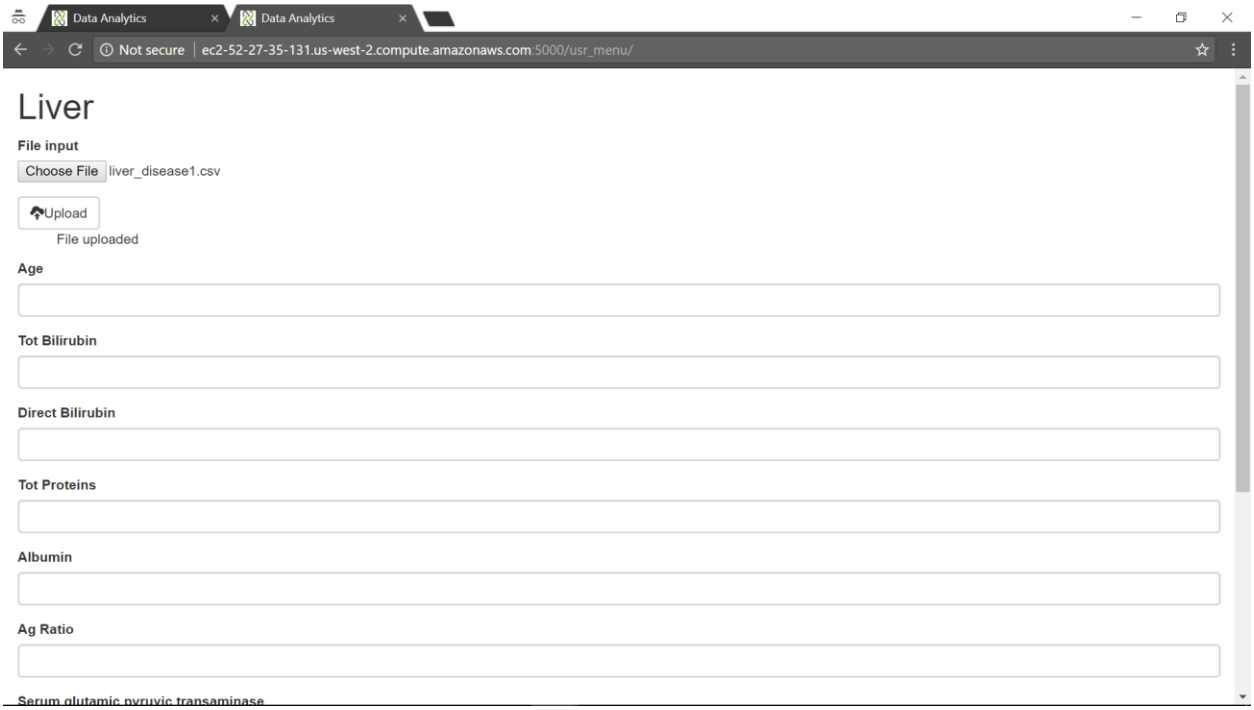
	Random Forest Classifier	Stochastic Gradient Decent	Logistic Regression	SVC	KNN	BNB	GNB
0	ckd	ckd	ckd	ckd	ckd	ckd	ckd

or Liver:

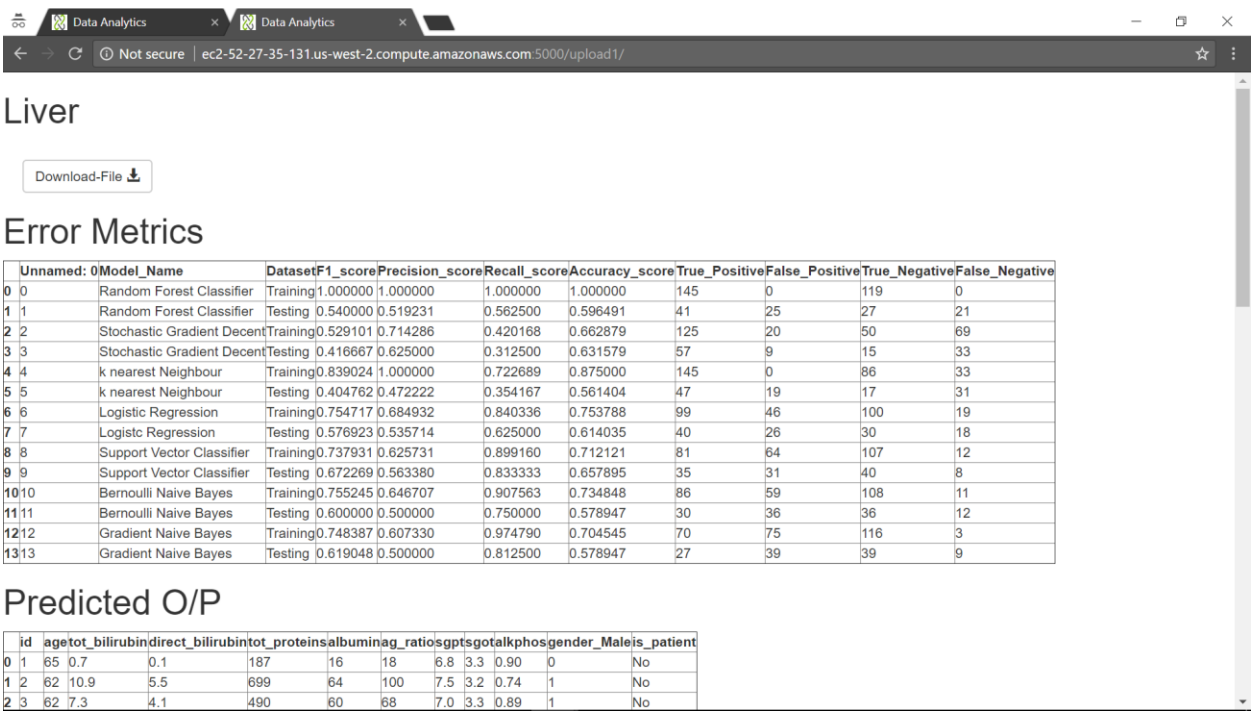




Uploading bulk data via csv file for prediction



Predicted O/p with error metrics of the models that were trained by admin



	Unnamed: 0	Model_Name	Dataset	F1_score	Precision_score	Recall_score	Accuracy_score	True_Positive	False_Positive	True_Negative	False_Negative
0	0	Random Forest Classifier	Training	1.000000	1.000000	1.000000	1.000000	145	0	119	0
1	1	Random Forest Classifier	Testing	0.540000	0.519231	0.562500	0.596491	41	25	27	21
2	2	Stochastic Gradient Decent	Training	0.529101	0.714286	0.420168	0.662879	125	20	50	69
3	3	Stochastic Gradient Decent	Testing	0.416667	0.625000	0.312500	0.631579	57	9	15	33
4	4	k nearest Neighbour	Training	0.839024	1.000000	0.722689	0.875000	145	0	86	33
5	5	k nearest Neighbour	Testing	0.404762	0.472222	0.354167	0.561404	47	19	17	31
6	6	Logistic Regression	Training	0.754717	0.684932	0.840336	0.753788	99	46	100	19
7	7	Logistc Regression	Testing	0.576923	0.535714	0.625000	0.614035	40	26	30	18
8	8	Support Vector Classifier	Training	0.737931	0.625731	0.899160	0.712121	81	64	107	12
9	9	Support Vector Classifier	Testing	0.672269	0.563380	0.833333	0.657895	35	31	40	8
10	10	Bernoulli Naive Bayes	Training	0.755245	0.646707	0.907563	0.734848	86	59	108	11
11	11	Bernoulli Naive Bayes	Testing	0.600000	0.500000	0.750000	0.578947	30	36	36	12
12	12	Gradient Naive Bayes	Training	0.748387	0.607330	0.974790	0.704545	70	75	116	3
13	13	Gradient Naive Bayes	Testing	0.619048	0.500000	0.812500	0.578947	27	39	39	9

	id	age	tot_bilirubin	direct_bilirubin	tot_proteins	albumin	ag_ratio	sgpt	sgot	alkphos	gender_Male	is_patient
0	1	65	0.7	0.1	187	16	18	6.8	3.3	0.90	0	No
1	2	62	10.9	5.5	699	64	100	7.5	3.2	0.74	1	No
2	3	62	7.3	4.1	490	60	68	7.0	3.3	0.89	1	No

Data Analytics | Data Analytics

Not secure | ec2-52-27-35-131.us-west-2.compute.amazonaws.com:5000/upload1/

Predicted O/P

	id	age	tot_bilirubin	direct_bilirubin	tot_proteins	albumin	ag_ratio	sgpt	sgot	alkphos	gender_Male	is_patient
0	1	65	0.7	0.1	187	16	18	6.8	3.3	0.90	0	No
1	2	62	10.9	5.5	699	64	100	7.5	3.2	0.74	1	No
2	3	62	7.3	4.1	490	60	68	7.0	3.3	0.89	1	No
3	4	58	1.0	0.4	182	14	20	6.8	3.4	1.00	1	Yes
4	5	72	3.9	2.0	195	27	59	7.3	2.4	0.40	1	No
5	6	46	1.8	0.7	208	19	14	7.6	4.4	1.30	1	No
6	7	26	0.9	0.2	154	16	12	7.0	3.5	1.00	0	No
7	8	29	0.9	0.3	202	14	11	6.7	3.6	1.10	0	No
8	9	55	0.7	0.2	290	53	58	6.8	3.4	1.00	1	No
9	10	57	0.6	0.1	210	51	59	5.9	2.7	0.80	1	Yes
10	11	72	2.7	1.3	260	31	56	7.4	3.0	0.60	1	No
11	12	74	1.1	0.4	214	22	30	8.1	4.1	1.00	0	No
12	13	61	0.7	0.2	145	53	41	5.8	2.7	0.87	1	No
13	14	38	1.8	0.8	342	168	441	7.6	4.4	1.30	1	No
14	15	40	0.9	0.3	293	232	245	6.8	3.1	0.80	0	No
15	16	40	0.9	0.3	293	232	245	6.8	3.1	0.80	0	No
16	17	51	2.2	1.0	610	17	28	7.3	2.6	0.55	1	No
17	18	51	2.9	1.3	482	22	34	7.0	2.4	0.50	1	No
18	19	62	6.8	3.0	542	116	66	6.4	3.1	0.90	1	No
19	20	40	1.9	1.0	231	16	55	4.3	1.6	0.60	1	No
20	21	34	4.1	2.0	289	875	731	5.0	2.7	1.10	1	No
21	22	34	4.1	2.0	289	875	731	5.0	2.7	1.10	1	No
22	23	34	6.2	3.0	240	1680	850	7.2	4.0	1.20	1	No
23	24	57	4.0	1.9	190	45	111	5.2	1.5	0.40	1	No
24	25	52	0.9	0.2	156	35	44	4.9	2.9	1.40	1	No
25	26	30	1.3	0.4	482	102	80	6.9	3.3	0.90	1	No
26	27	46	14.2	7.8	374	38	77	4.3	2.0	0.80	0	No

Downloading the output files

Data Analytics | Data Analytics

Not secure | ec2-52-27-35-131.us-west-2.compute.amazonaws.com:5000/upload1/

Liver

Download-File

Error Metrics

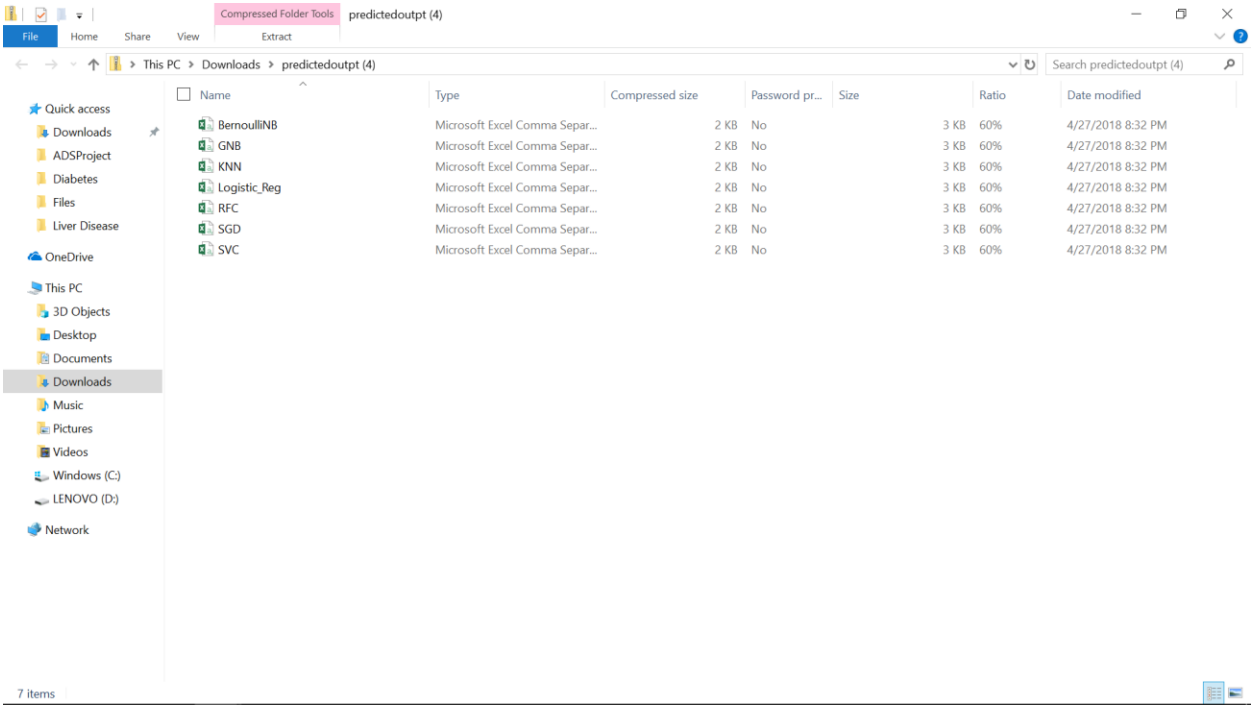
	Unnamed: 0	Model_Name	Dataset	F1_score	Precision_score	Recall_score	Accuracy_score	True_Positive	False_Positive	True_Negative	False_Negative
0	0	Random Forest Classifier	Training	1.000000	1.000000	1.000000	1.000000	145	0	119	0
1	1	Random Forest Classifier	Testing	0.540000	0.519231	0.562500	0.596491	41	25	27	21
2	2	Stochastic Gradient Decent	Training	0.529101	0.714286	0.420168	0.662879	125	20	50	69
3	3	Stochastic Gradient Decent	Testing	0.416667	0.625000	0.312500	0.631579	57	9	15	33
4	4	k nearest Neighbour	Training	0.839024	1.000000	0.722689	0.875000	145	0	86	33
5	5	k nearest Neighbour	Testing	0.404762	0.472222	0.354167	0.561404	47	19	17	31
6	6	Logistic Regression	Training	0.754717	0.684932	0.840336	0.753788	99	46	100	19
7	7	Logistc Regression	Testing	0.576923	0.535714	0.625000	0.614035	40	26	30	18
8	8	Support Vector Classifier	Training	0.737931	0.625731	0.899160	0.712121	81	64	107	12
9	9	Support Vector Classifier	Testing	0.672269	0.563380	0.833333	0.657895	35	31	40	8
10	10	Bernoulli Naive Bayes	Training	0.755245	0.646707	0.907563	0.734848	86	59	108	11
11	11	Bernoulli Naive Bayes	Testing	0.600000	0.500000	0.750000	0.578947	30	36	36	12
12	12	Gradient Naive Bayes	Training	0.748387	0.607330	0.974790	0.704545	70	75	116	3
13	13	Gradient Naive Bayes	Testing	0.619048	0.500000	0.812500	0.578947	27	39	39	9

Predicted O/P

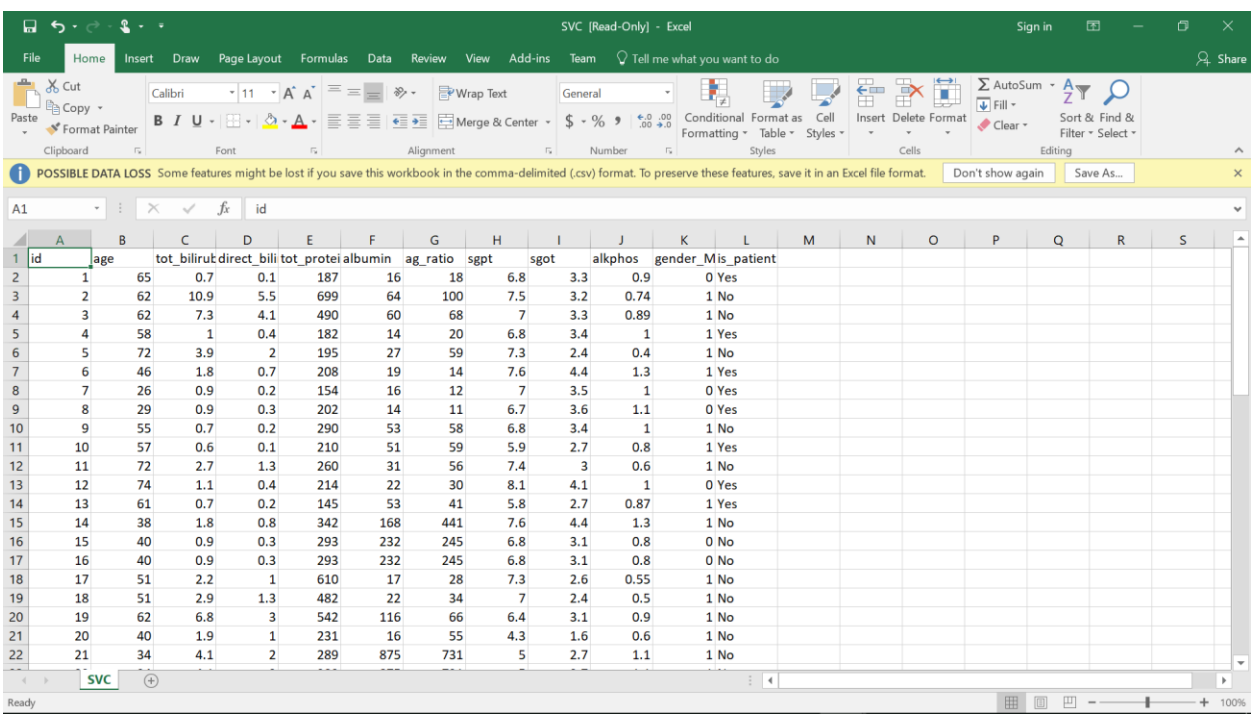
	id	age	tot_bilirubin	direct_bilirubin	tot_proteins	albumin	ag_ratio	sgpt	sgot	alkphos	gender_Male	is_patient
0	1	65	0.7	0.1	187	16	18	6.8	3.3	0.90	0	No

predictedoutpt (4).zip | Show all

Output files generated by each models

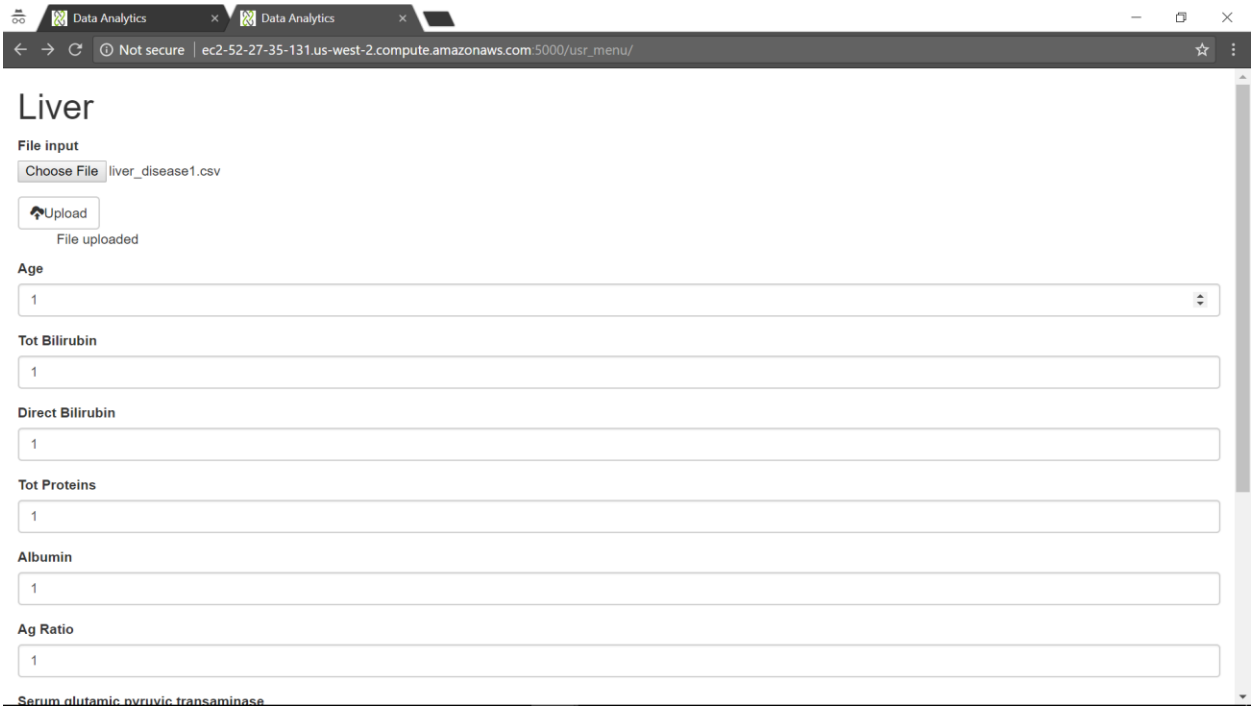


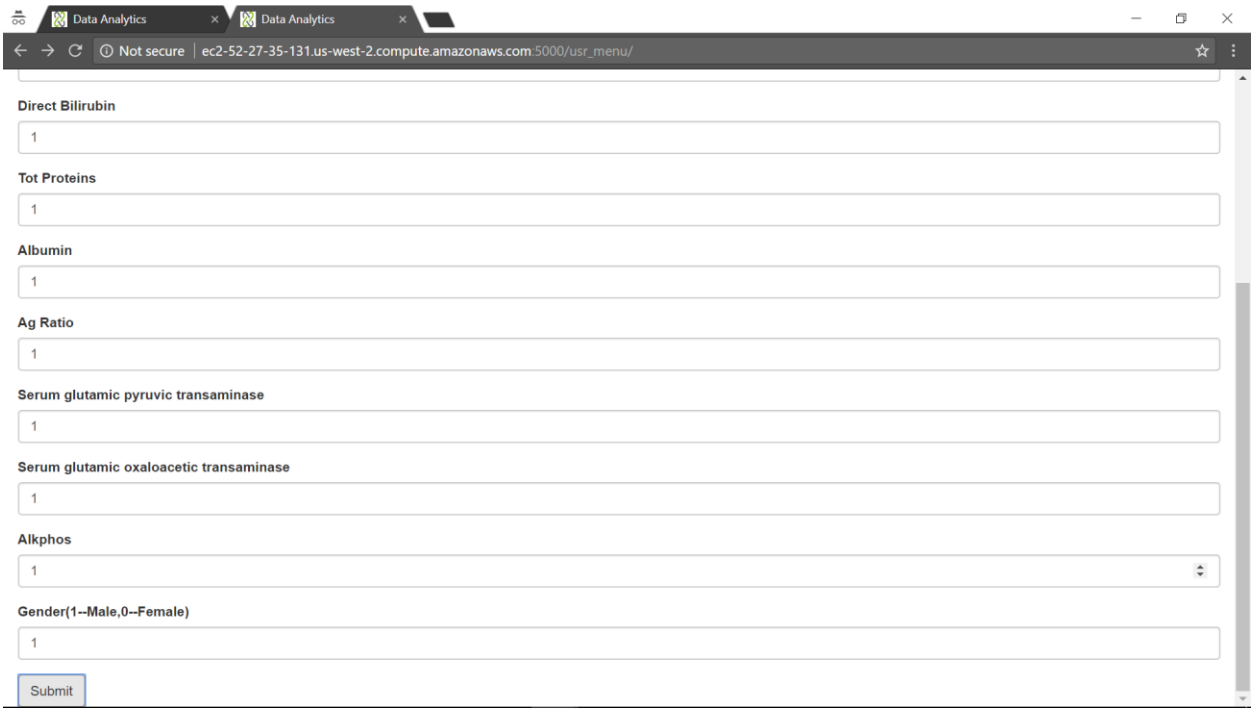
Output file generated by SVC model



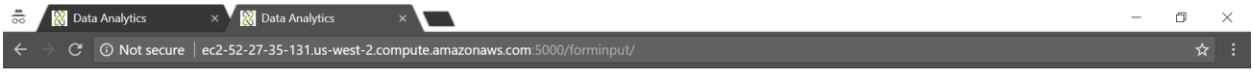
id	age	tot_bilirut	direct_bili	tot_protei	albumin	ag_ratio	sgpt	sgot	alkphos	gender_M	is_patient
1	65	0.7	0.1	187	16	18	6.8	3.3	0.9	0	Yes
2	62	10.9	5.5	699	64	100	7.5	3.2	0.74	1	No
3	62	7.3	4.1	490	60	68	7	3.3	0.89	1	No
4	58	1	0.4	182	14	20	6.8	3.4	1	1	Yes
5	72	3.9	2	195	27	59	7.3	2.4	0.4	1	No
6	46	1.8	0.7	208	19	14	7.6	4.4	1.3	1	Yes
7	26	0.9	0.2	154	16	12	7	3.5	1	0	Yes
8	29	0.9	0.3	202	14	11	6.7	3.6	1.1	0	Yes
9	55	0.7	0.2	290	53	58	6.8	3.4	1	1	No
10	57	0.6	0.1	210	51	59	5.9	2.7	0.8	1	Yes
11	72	2.7	1.3	260	31	56	7.4	3	0.6	1	No
12	74	1.1	0.4	214	22	30	8.1	4.1	1	0	Yes
13	61	0.7	0.2	145	53	41	5.8	2.7	0.87	1	Yes
14	38	1.8	0.8	342	168	441	7.6	4.4	1.3	1	No
15	40	0.9	0.3	293	232	245	6.8	3.1	0.8	0	No
16	40	0.9	0.3	293	232	245	6.8	3.1	0.8	0	No
17	51	2.2	1	610	17	28	7.3	2.6	0.55	1	No
18	51	2.9	1.3	482	22	34	7	2.4	0.5	1	No
19	62	6.8	3	542	116	66	6.4	3.1	0.9	1	No
20	40	1.9	1	231	16	55	4.3	1.6	0.6	1	No
21	34	4.1	2	289	875	731	5	2.7	1.1	1	No

Form Data entry:





Predicted output generated for form data input



Liver

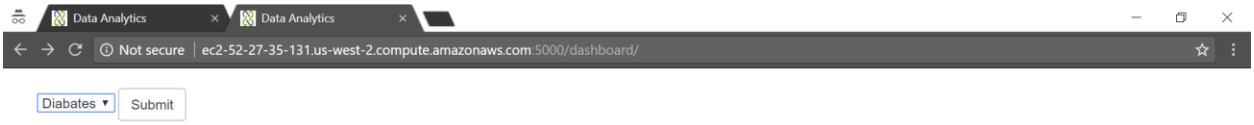
Inputs Given

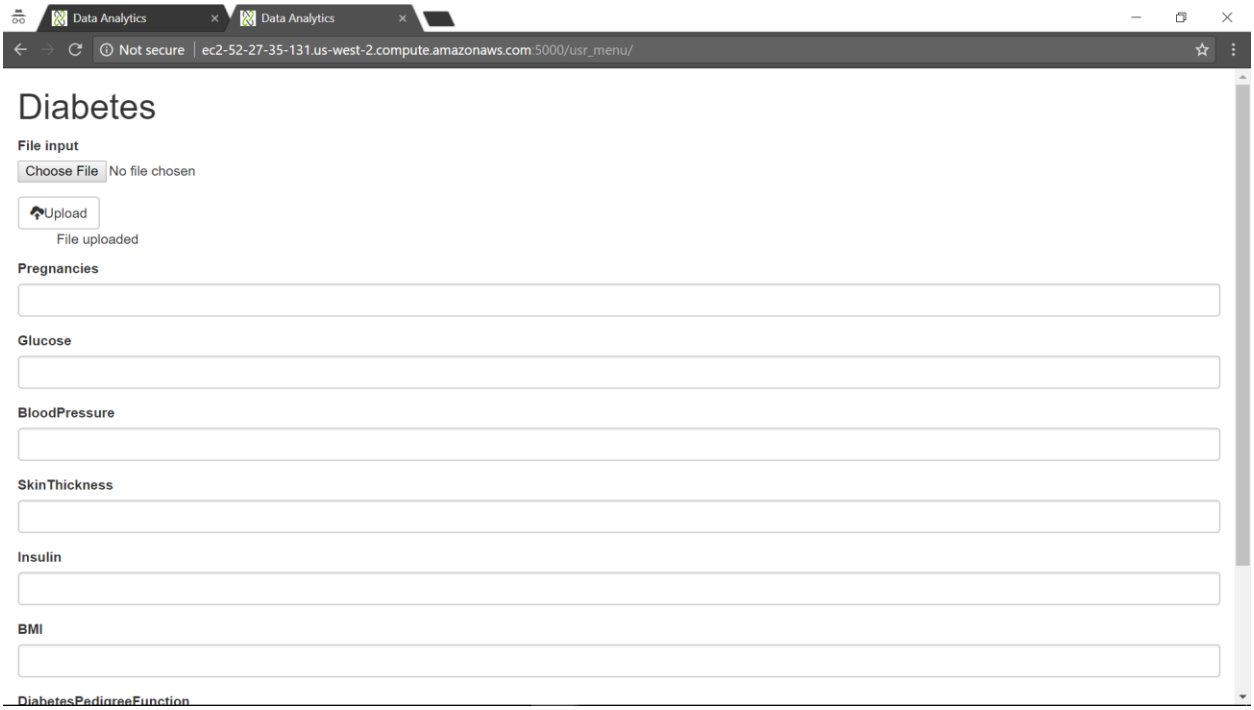
	age	tot_bilirubin	direct_bilirubin	tot_proteins	albumin	ag_ratio	sgpt	sgot	alkphos	gender_Male
0	1	1	1	1	1	1	1	1	1	1

O/p Predicted by different Models

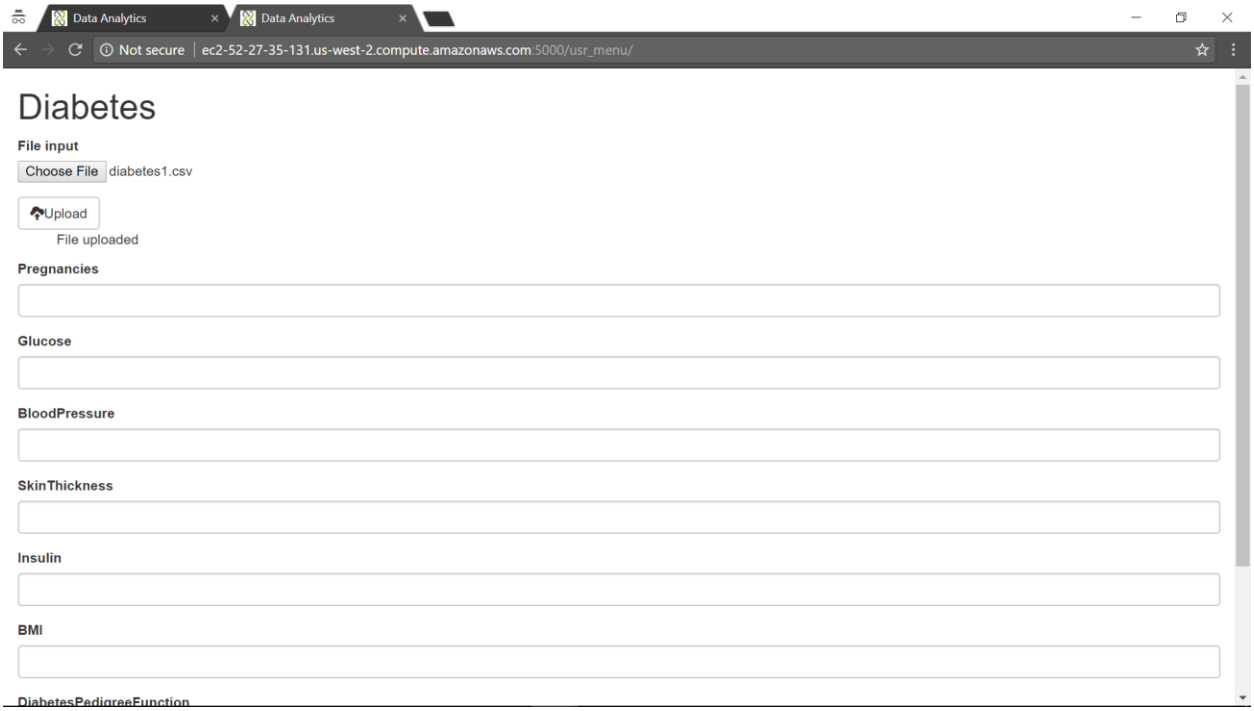
	Random Forest Classifier	Stochastic Gradient Decent	Logistic Regression	SVC	KNN	BNB	GNB
0	No	No	No	No	No	Yes	No

For Diabetes:





Uploading bulk data in form of csv file to get predicted output.



Predicted O/p with error metrics of the models that were trained by admin

Data Analytics | Data Analytics

Not secure | ec2-52-27-35-131.us-west-2.compute.amazonaws.com:5000/upload2/

Diabetes

Download-File

Error Metrics

	Unnamed: 0	Model_Name	Dataset	F1_score	Precision_score	Recall_score	Accuracy_score	True_Positive	False_Positive	True_Negative	False_Negative
0	0	Random Forest Classifier	Training	1.000000	1.000000	1.000000	1.000000	231	0	186	0
1	1	Random Forest Classifier	Testing	0.766667	0.750000	0.732558	0.768293	75	23	63	19
2	2	Stochastic Gradient Decent	Training	0.702638	0.635294	0.701299	0.580645	185	46	108	78
3	3	Stochastic Gradient Decent	Testing	0.694444	0.640523	0.690141	0.597561	76	22	49	33
4	4	k nearest Neighbour	Training	0.851319	0.834225	0.829787	0.838710	199	32	156	30
5	5	k nearest Neighbour	Testing	0.738889	0.700637	0.733333	0.670732	78	20	55	27
6	6	Logistic Regression	Training	0.760192	0.712644	0.765432	0.666667	193	38	124	62
7	7	Logistic Regression	Testing	0.772222	0.732026	0.788732	0.682927	83	15	56	26
8	8	Support Vector Classifier	Training	0.750600	0.699422	0.756250	0.650538	192	39	121	65
9	9	Support Vector Classifier	Testing	0.766667	0.723684	0.785714	0.670732	83	15	55	27
10	10	Bernoulli Naive Bayes	Training	0.700240	0.668435	0.659686	0.677419	166	65	126	60
11	11	Bernoulli Naive Bayes	Testing	0.711111	0.679012	0.687500	0.670732	73	25	55	27
12	12	Gradient Naive Bayes	Training	0.733813	0.676385	0.738854	0.623656	190	41	116	70
13	13	Gradient Naive Bayes	Testing	0.766667	0.727273	0.777778	0.682927	82	16	56	26

Predicted O/P

	id	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	1	6	148	72	35	0	33.6	0.627	50	Yes
1	2	1	85	66	29	0	26.6	0.351	31	No
2	3	8	183	64	0	0	23.3	0.672	32	Yes

Predicted O/p

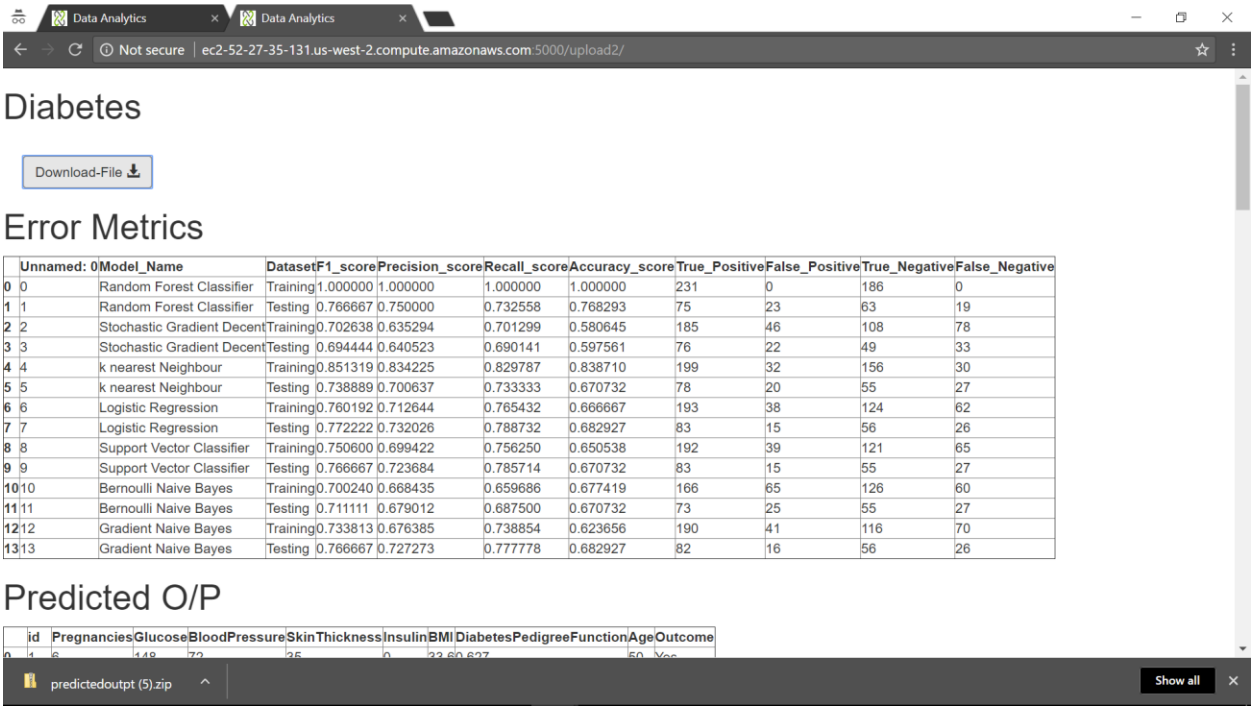
Data Analytics | Data Analytics

Not secure | ec2-52-27-35-131.us-west-2.compute.amazonaws.com:5000/upload2/

Predicted O/P

	id	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	1	6	148	72	35	0	33.6	0.627	50	Yes
1	2	1	85	66	29	0	26.6	0.351	31	No
2	3	8	183	64	0	0	23.3	0.672	32	Yes
3	4	1	89	66	23	94	28.1	0.167	21	No
4	5	0	137	40	35	168	43.1	2.288	33	Yes
5	6	5	116	74	0	0	25.6	0.201	30	No
6	7	3	78	50	32	88	31.0	0.248	26	No
7	8	10	115	0	0	0	35.3	0.134	29	No
8	9	2	197	70	45	543	30.5	0.158	53	Yes
9	10	8	125	96	0	0	0.0	0.232	54	No
10	11	4	110	92	0	0	37.6	0.191	30	No
11	12	10	168	74	0	0	38.0	0.537	34	Yes
12	13	10	139	80	0	0	27.1	1.441	57	No
13	14	1	189	60	23	846	30.1	0.398	59	Yes
14	15	5	166	72	19	175	25.8	0.587	51	Yes
15	16	7	100	0	0	0	30.0	0.484	32	Yes
16	17	0	118	84	47	230	45.8	0.551	31	Yes
17	18	7	107	74	0	0	29.6	0.254	31	Yes
18	19	1	103	30	38	83	43.3	0.183	33	No
19	20	1	115	70	30	96	34.6	0.529	32	Yes
20	21	3	126	88	41	235	39.3	0.704	27	Yes
21	22	8	99	84	0	0	35.4	0.388	50	No
22	23	7	196	90	0	0	39.8	0.451	41	Yes
23	24	9	119	80	35	0	29.0	0.263	29	Yes
24	25	11	143	94	33	146	36.6	0.254	51	Yes
25	26	10	125	70	26	115	31.1	0.205	41	Yes
26	27	7	147	76	0	0	39.4	0.257	43	Yes
27	28	1	97	66	15	140	23.2	0.487	22	No

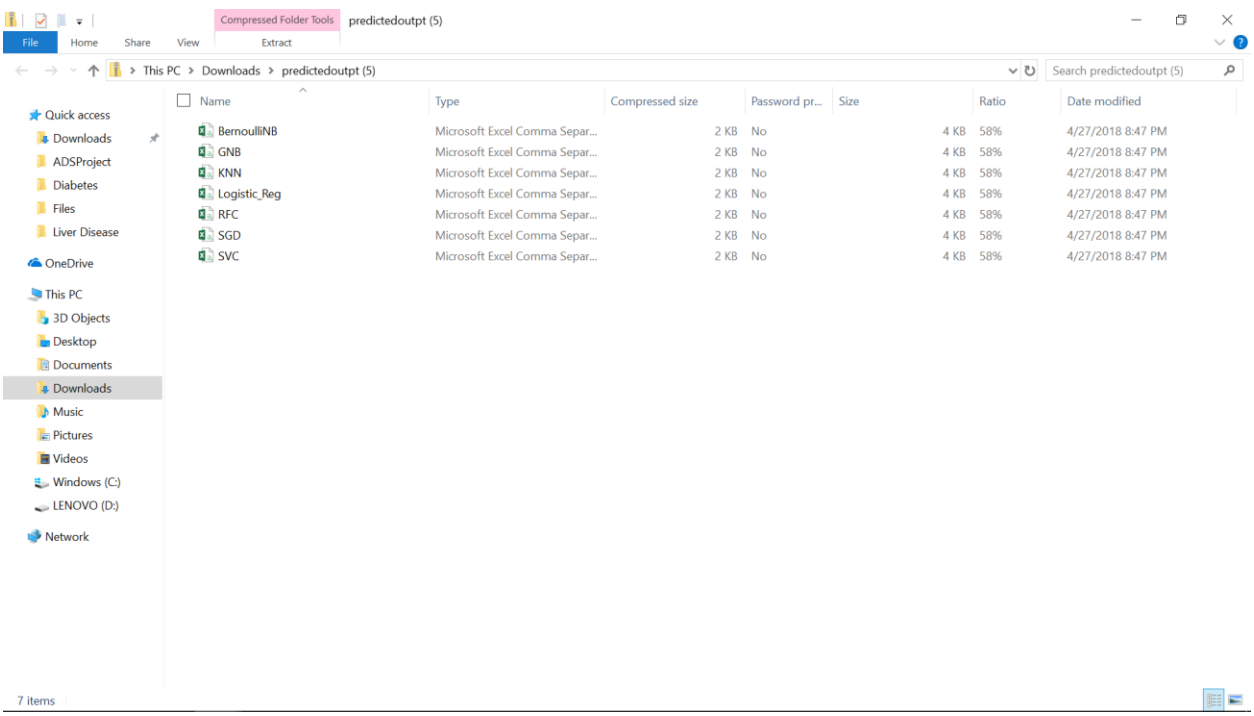
Downloading the output files



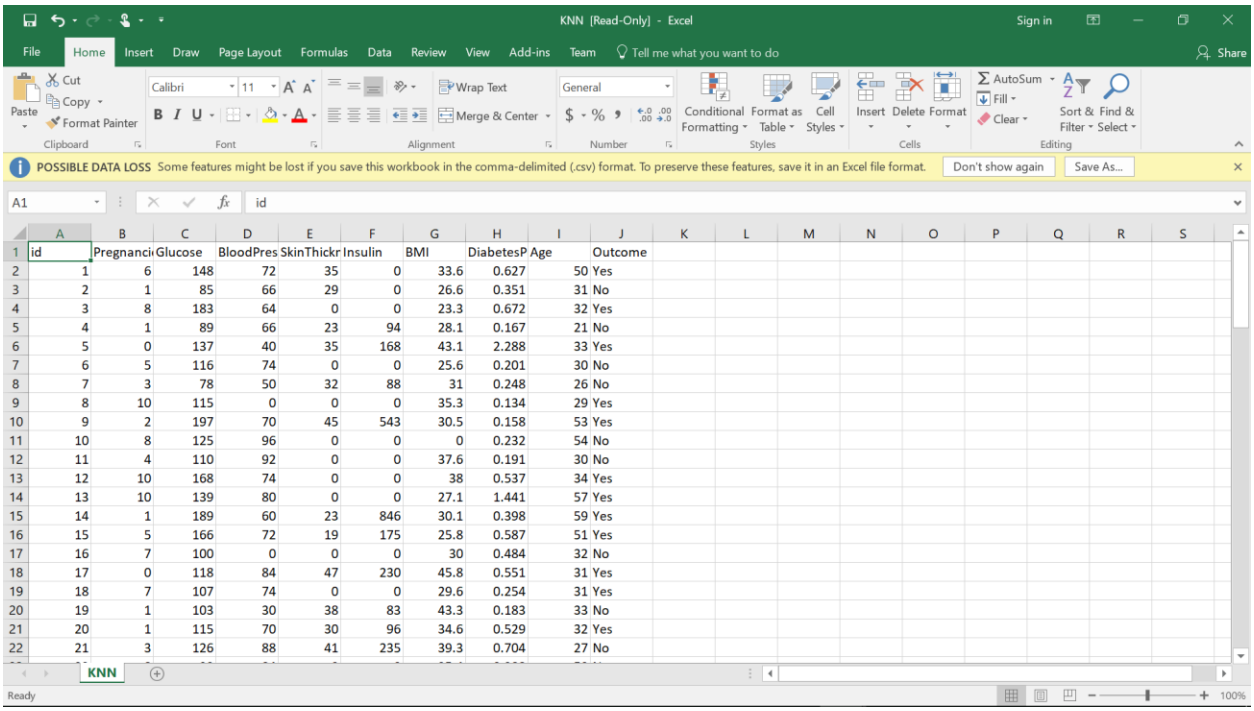
	Unnamed: 0	Model_Name	Dataset	F1_score	Precision_score	Recall_score	Accuracy_score	True_Positive	False_Positive	True_Negative	False_Negative
0	0	Random Forest Classifier	Training	1.000000	1.000000	1.000000	1.000000	231	0	186	0
1	1	Random Forest Classifier	Testing	0.766667	0.750000	0.732558	0.768293	75	23	63	19
2	2	Stochastic Gradient Decent	Training	0.702638	0.635294	0.701299	0.580645	185	46	108	78
3	3	Stochastic Gradient Decent	Testing	0.694444	0.640523	0.690141	0.597561	76	22	49	33
4	4	k nearest Neighbour	Training	0.851319	0.834225	0.829787	0.838710	199	32	156	30
5	5	k nearest Neighbour	Testing	0.738889	0.700637	0.733333	0.670732	78	20	55	27
6	6	Logistic Regression	Training	0.760192	0.712644	0.765432	0.666667	193	38	124	62
7	7	Logistic Regression	Testing	0.772222	0.732026	0.788732	0.682927	83	15	56	26
8	8	Support Vector Classifier	Training	0.750600	0.699422	0.756250	0.650538	192	39	121	65
9	9	Support Vector Classifier	Testing	0.766667	0.723684	0.785714	0.670732	83	15	55	27
10	10	Bernoulli Naive Bayes	Training	0.700240	0.668435	0.659686	0.677419	166	65	126	60
11	11	Bernoulli Naive Bayes	Testing	0.711111	0.679012	0.687500	0.670732	73	25	55	27
12	12	Gradient Naive Bayes	Training	0.733813	0.676385	0.738854	0.623656	190	41	116	70
13	13	Gradient Naive Bayes	Testing	0.766667	0.727273	0.777778	0.682927	82	16	56	26

	id	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	1	6	148	72	35	0	33.60	0.627	50	Yes

Output generated by each model

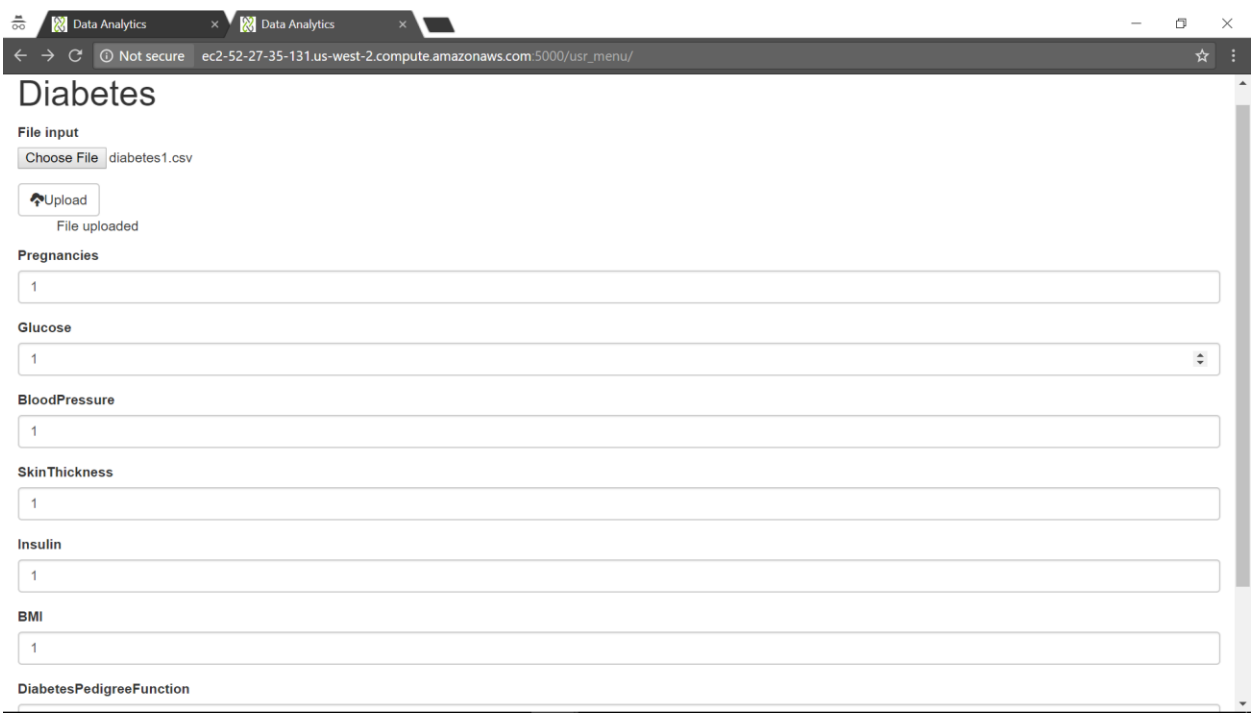


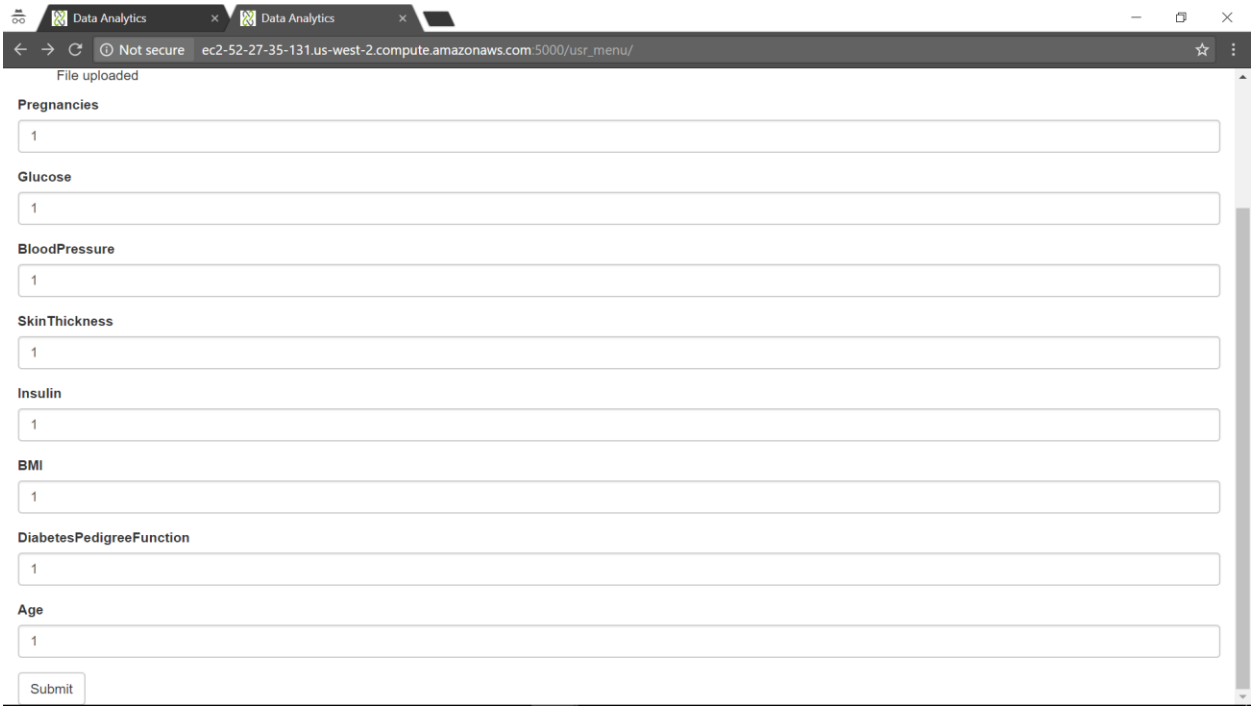
Output generated by KNN



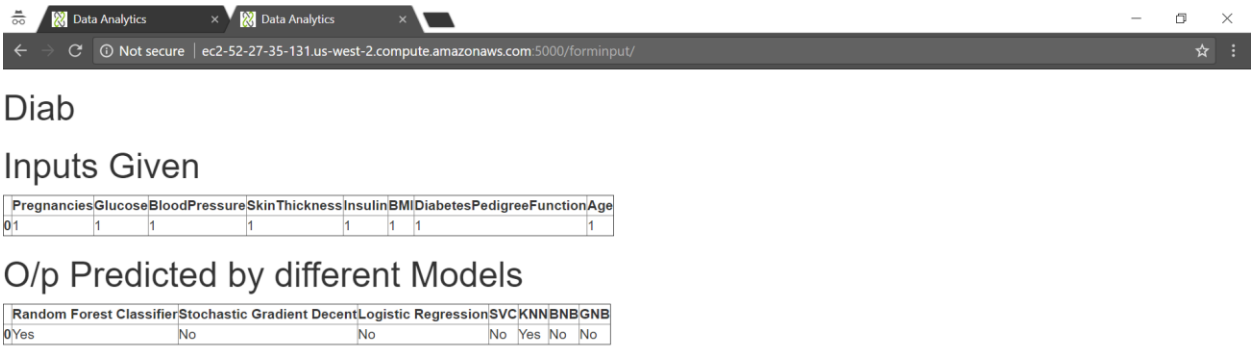
id	Pregnanci	Glucose	BloodPres	SkinThickr	Insulin	BMI	DiabetesP	Age	Outcome
1	6	148	72	35	0	33.6	0.627	50	Yes
2	1	85	66	29	0	26.6	0.351	31	No
3	8	183	64	0	0	23.3	0.672	32	Yes
4	1	89	66	23	94	28.1	0.167	21	No
5	0	137	40	35	168	43.1	2.288	33	Yes
6	5	116	74	0	0	25.6	0.201	30	No
7	3	78	50	32	88	31	0.248	26	No
8	10	115	0	0	0	35.3	0.134	29	Yes
9	2	197	70	45	543	30.5	0.158	53	Yes
10	8	125	96	0	0	0	0.232	54	No
11	4	110	92	0	0	37.6	0.191	30	No
12	10	168	74	0	0	38	0.537	34	Yes
13	10	139	80	0	0	27.1	1.441	57	Yes
14	1	189	60	23	846	30.1	0.398	59	Yes
15	5	166	72	19	175	25.8	0.587	51	Yes
16	7	100	0	0	0	30	0.484	32	No
17	0	118	84	47	230	45.8	0.551	31	Yes
18	7	107	74	0	0	29.6	0.254	31	Yes
19	1	103	30	38	83	43.3	0.183	33	No
20	1	115	70	30	96	34.6	0.529	32	Yes
21	3	126	88	41	235	39.3	0.704	27	No

Form Data entry:





Predicted output generated for form data input



Rest api:

For Kidney

Command:

curl -H "Content-Type: application/json" -X POST -d @z.json http://ec2-52-27-35-131.us-west-2.compute.amazonaws.com:5000/predictoutput/

```
MINGW64:/c/Users/Akash/Documents/ADS/ADSProject/Flask
    "wc": 7500,
    "rc": " ",
    "htn": "no",
    "dm": "yes",
    "cad": "no",
    "appet": "poor",

Akash@DESKTOP-BBTTUHF MINGW64 ~/Documents/ADS/ADSProject/Flask
$ cat z.json | head -30
[
  {
    "id": 2,
    "age": 62,
    "bp": 80,
    "sg": 1.01,
    "al": 2,
    "su": 3,
    "rbc": "normal",
    "pc": "normal",
    "pcc": "notpresent",
    "ba": "notpresent",
    "bgr": 423,
    "bu": 53,
    "sc": 1.8,
    "sod": " ",
    "pot": " ",
    "hemo": 9.6,
    "pcv": 31,
    "wc": 7500,
    "rc": " ",
    "htn": "no",
    "dm": "yes",
    "cad": "no",
    "appet": "poor",
    "pe": "no",
    "ane": "yes"
  },
  {
    "id": 3,

Akash@DESKTOP-BBTTUHF MINGW64 ~/Documents/ADS/ADSProject/Flask
$ curl -H "Content-Type: application/json" -X POST -d @z.json http://ec2-52-27-35-131.us-west-2.compute.amazonaws.com:5
000/predictoutput/
```

Output

```
MINGW64:/c/Users/Akash/Documents/ADS/ADSProject/Flask
    "su": 3,
    "rbc": "normal",
    "pc": "normal",
    "pcc": "notpresent",
    "ba": "notpresent",
    "bgr": 423,
    "bu": 53,
    "sc": 1.8,
    "sod": " ",
    "pot": " ",
    "hemo": 9.6,
    "pcv": 31,
    "wc": 7500,
    "rc": " ",
    "htn": "no",
    "dm": "yes",
    "cad": "no",
    "appet": "poor",
    "pe": "no",
    "ane": "yes"
  },
  {
    "id": 3,

Akash@DESKTOP-BBTTUHF MINGW64 ~/Documents/ADS/ADSProject/Flask
$ curl -H "Content-Type: application/json" -X POST -d @z.json http://ec2-52-27-35-131.us-west-2.compute.amazonaws.com:5
000/predictoutput/
[{"age": 62, "al": 2, "bgr": 423, "bp": 80, "bu": 53, "hemo": 9.6, "id": 2, "pcv": 31.0, "pot": 4.0166666667, "rc": 4.5166666667, "sc": 1.8, "sg": 1.01, "sod": 134.0, "su":
 3, "wc": 7500, "RBC_normal": 1, "PC_normal": 1, "PCC_present": 0, "HTN_yes": 0, "DM_yes": 1, "Appet_good": 0, "PE_yes": 0, "Ane_yes": 1, "Classification_ckd": "ckd"}, {"ag
e": 48, "al": 4, "bgr": 117, "bp": 70, "bu": 56, "hemo": 11.2, "id": 3, "pcv": 32.0, "pot": 2.5, "rc": 3.9, "sc": 3.8, "sg": 1.005, "sod": 111.0, "su": 0, "wc": 6700, "RBC_
normal": 1, "PC_normal": 0, "PCC_present": 1, "HTN_yes": 1, "DM_yes": 0, "Appet_good": 0, "PE_yes": 1, "Ane_yes": 1, "Classification_ckd": "ckd"}, {"age": 53, "al": 2, "bgr
": 70, "bp": 90, "bu": 107, "hemo": 9.5, "id": 9, "pcv": 29.0, "pot": 3.7, "rc": 3.7, "sc": 7.2, "sg": 1.02, "sod": 114.0, "su": 0, "wc": 12100, "RBC_normal": 0, "PC_normal
": 0, "PCC_present": 1, "HTN_yes": 1, "DM_yes": 1, "Appet_good": 0, "PE_yes": 0, "Ane_yes": 1, "Classification_ckd": "ckd"}, {"age": 29, "al": 0, "bgr": 83, "bp": 80, "bu":
 49, "hemo": 17.5, "id": 351, "pcv": 40.0, "pot": 3.3, "rc": 4.7, "sc": 0.9, "sg": 1.02, "sod": 139.0, "su": 0, "wc": 9900, "RBC_normal": 1, "PC_normal": 1, "PCC_present":
0, "HTN_yes": 0, "DM_yes": 0, "Appet_good": 1, "PE_yes": 0, "Ane_yes": 0, "Classification_ckd": "nckd"}, {"age": 37, "al": 0, "bgr": 109, "bp": 60, "bu": 47, "hemo": 15.0,
"id": 352, "pcv": 48.0, "pot": 4.9, "rc": 5.2, "sc": 1.1, "sg": 1.02, "sod": 141.0, "su": 0, "wc": 7000, "RBC_normal": 1, "PC_normal": 1, "PCC_present": 0, "HTN_yes": 0, "D
M_yes": 0, "Appet_good": 1, "PE_yes": 0, "Ane_yes": 0, "Classification_ckd": "nckd"}, {"age": 39, "al": 0, "bgr": 86, "bp": 60, "bu": 37, "hemo": 13.6, "id": 353, "pcv": 51
.0, "pot": 5.0, "rc": 4.5, "sc": 0.6, "sg": 1.02, "sod": 150.0, "su": 0, "wc": 5800, "RBC_normal": 1, "PC_normal": 1, "PCC_present": 0, "HTN_yes": 0, "DM_yes": 0, "Appet_go
od": 1, "PE_yes": 0, "Ane_yes": 0, "Classification_ckd": "nckd"}, {"age": 32, "al": 0, "bgr": 102, "bp": 60, "bu": 17, "hemo": 14.6, "id": 354, "pcv": 41.0, "pot": 4.7, "rc
": 5.1, "sc": 0.4, "sg": 1.025, "sod": 147.0, "su": 0, "wc": 6800, "RBC_normal": 1, "PC_normal": 1, "PCC_present": 0, "HTN_yes": 0, "DM_yes": 0, "Appet_good": 1, "PE_yes":
0, "Ane_yes": 0, "Classification_ckd": "nckd"}]
Akash@DESKTOP-BBTTUHF MINGW64 ~/Documents/ADS/ADSProject/Flask
$
```

For Liver:

Command:

curl -H "Content-Type: application/json" -X POST -d @z1.json http://ec2-52-27-35-131.us-west-2.compute.amazonaws.com:5000/predictoutput1/

```
MINGW64:/c/Users/Akash/Documents/ADS/ADSProject/Flask
0, "HTN_yes": 0, "DM_yes": 0, "Appet_good": 1, "PE_yes": 0, "Ane_yes": 0, "Classification_ckd": "nckd"}, {"age": 37, "al": 0, "bgr": 109, "bp": 60, "bu": 47, "hemo": 15.0,
"id": 352, "pcv": 48.0, "pot": 4.9, "rc": 5.2, "sc": 1.1, "sg": 1.02, "sod": 141.0, "su": 0, "wc": 7000, "RBC_normal": 1, "PC_normal": 1, "PCC_present": 0, "HTN_yes": 0, "D
M_yes": 0, "Appet_good": 1, "PE_yes": 0, "Ane_yes": 0, "Classification_ckd": "nckd"}, {"age": 39, "al": 0, "bgr": 86, "bp": 60, "bu": 37, "hemo": 13.6, "id": 353, "pcv": 51
.0, "pot": 5.0, "rc": 4.5, "sc": 0.6, "sg": 1.02, "sod": 150.0, "su": 0, "wc": 5800, "RBC_normal": 1, "PC_normal": 1, "PCC_present": 0, "HTN_yes": 0, "DM_yes": 0, "Appet_go
od": 1, "PE_yes": 0, "Ane_yes": 0, "Classification_ckd": "nckd"}, {"age": 32, "al": 0, "bgr": 102, "bp": 60, "bu": 17, "hemo": 14.6, "id": 354, "pcv": 41.0, "pot": 4.7, "rc
": 5.1, "sc": 0.4, "sg": 1.025, "sod": 147.0, "su": 0, "wc": 6800, "RBC_normal": 1, "PC_normal": 1, "PCC_present": 0, "HTN_yes": 0, "DM_yes": 0, "Appet_good": 1, "PE_yes":
0, "Ane_yes": 0, "Classification_ckd": "nckd"}]
Akash@DESKTOP-BBTTUHF MINGW64 ~/Documents/ADS/ADSProject/Flask
$ cat z1.json | head -30
[
  {
    "id": 1,
    "age": 65,
    "gender": "Female",
    "tot_bilirubin": 0.7,
    "direct_bilirubin": 0.1,
    "tot_proteins": 187,
    "albumin": 16,
    "ag_ratio": 18,
    "sgpt": 6.8,
    "sgot": 3.3,
    "alkphos": 0.9
  },
  {
    "id": 2,
    "age": 62,
    "gender": "Male",
    "tot_bilirubin": 10.9,
    "direct_bilirubin": 5.5,
    "tot_proteins": 699,
    "albumin": 64,
    "ag_ratio": 100,
    "sgpt": 7.5,
    "sgot": 3.2,
    "alkphos": 0.74
  },
  {
    "id": 3,
    "age": 62,

Akash@DESKTOP-BBTTUHF MINGW64 ~/Documents/ADS/ADSProject/Flask
$ curl -H "Content-Type: application/json" -X POST -d @z1.json http://ec2-52-27-35-131.us-west-2.compute.amazonaws.com:
5000/predictoutput1/
```

Output

```
MINGW64:/c/Users/Akash/Documents/ADS/ADSProject/Flask
0, "Ane_yes": 0, "Classification_ckd": "nckd"}]
Akash@DESKTOP-BBTTUHF MINGW64 ~/Documents/ADS/ADSProject/Flask
$ cat z1.json | head -30
[
  {
    "id": 1,
    "age": 65,
    "gender": "Female",
    "tot_bilirubin": 0.7,
    "direct_bilirubin": 0.1,
    "tot_proteins": 187,
    "albumin": 16,
    "ag_ratio": 18,
    "sgpt": 6.8,
    "sgot": 3.3,
    "alkphos": 0.9
  },
  {
    "id": 2,
    "age": 62,
    "gender": "Male",
    "tot_bilirubin": 10.9,
    "direct_bilirubin": 5.5,
    "tot_proteins": 699,
    "albumin": 64,
    "ag_ratio": 100,
    "sgpt": 7.5,
    "sgot": 3.2,
    "alkphos": 0.74
  },
  {
    "id": 3,
    "age": 62,

Akash@DESKTOP-BBTTUHF MINGW64 ~/Documents/ADS/ADSProject/Flask
$ curl -H "Content-Type: application/json" -X POST -d @z1.json http://ec2-52-27-35-131.us-west-2.compute.amazonaws.com:
5000/predictoutput1/
[{"ag_ratio": 18, "age": 65, "albumin": 16, "alkphos": 0.9, "direct_bilirubin": 0.1, "id": 1, "sgot": 3.3, "sgpt": 6.8, "tot_bilirubin": 0.7, "tot_proteins": 187, "gender_M
ale": 0, "is_patient": "Yes"}, {"ag_ratio": 100, "age": 62, "albumin": 64, "alkphos": 0.74, "direct_bilirubin": 5.5, "id": 2, "sgot": 3.2, "sgpt": 7.5, "tot_bilirubin": 10.
9, "tot_proteins": 699, "gender_Male": 1, "is_patient": "No"}, {"ag_ratio": 68, "age": 62, "albumin": 60, "alkphos": 0.89, "direct_bilirubin": 4.1, "id": 3, "sgot": 3.3, "s
gpt": 7.0, "tot_bilirubin": 7.3, "tot_proteins": 490, "gender_Male": 1, "is_patient": "No"}]
Akash@DESKTOP-BBTTUHF MINGW64 ~/Documents/ADS/ADSProject/Flask
$
```

For Diabetes:

curl -H "Content-Type: application/json" -X POST -d @z2.json http://ec2-52-27-35-131.us-west-2.compute.amazonaws.com:5000/predictoutput2/

```
MINGW64:/c/Users/Akash/Documents/ADS/ADSProject/Flask
Akash@DESKTOP-BBTTUHF MINGW64 ~/Documents/ADS/ADSProject/Flask
$ curl -H "Content-Type: application/json" -X POST -d @z1.json http://ec2-52-27-35-131.us-west-2.compute.amazonaws.com:
5000/predictoutput1/
[{"ag_ratio": 18, "age": 65, "albumin": 16, "alkphos": 0.9, "direct_bilirubin": 0.1, "id": 1, "sgot": 3.3, "sgpt": 6.8, "tot_bilirubin": 0.7, "tot_proteins": 187, "gender_M
ale": 0, "is_patient": "Yes"}, {"ag_ratio": 100, "age": 62, "albumin": 64, "alkphos": 0.74, "direct_bilirubin": 5.5, "id": 2, "sgot": 3.2, "sgpt": 7.5, "tot_bilirubin": 10.
9, "tot_proteins": 699, "gender_Male": 1, "is_patient": "No"}, {"ag_ratio": 68, "age": 62, "albumin": 60, "alkphos": 0.89, "direct_bilirubin": 4.1, "id": 3, "sgot": 3.3, "s
gpt": 7.0, "tot_bilirubin": 7.3, "tot_proteins": 490, "gender_Male": 1, "is_patient": "No"}]
Akash@DESKTOP-BBTTUHF MINGW64 ~/Documents/ADS/ADSProject/Flask
$ cat z2.json | head -30
[
  {
    "id": 1,
    "Pregnancies": 6,
    "Glucose": 148,
    "BloodPressure": 72,
    "SkinThickness": 35,
    "Insulin": 0,
    "BMI": 33.6,
    "DiabetesPedigreeFunction": 0.627,
    "Age": 50
  },
  {
    "id": 2,
    "Pregnancies": 1,
    "Glucose": 85,
    "BloodPressure": 66,
    "SkinThickness": 29,
    "Insulin": 0,
    "BMI": 26.6,
    "DiabetesPedigreeFunction": 0.351,
    "Age": 31
  },
  {
    "id": 3,
    "Pregnancies": 8,
    "Glucose": 183,
    "BloodPressure": 64,
    "SkinThickness": 0,
    "Insulin": 0,

Akash@DESKTOP-BBTTUHF MINGW64 ~/Documents/ADS/ADSProject/Flask
$ curl -H "Content-Type: application/json" -X POST -d @z2.json http://ec2-52-27-35-131.us-west-2.compute.amazonaws.com:
5000/predictoutput2/
```

Output


```
MINGW64:/c/Users/Akash/Documents/ADS/ADSProject/Flask
Akash@DESKTOP-BBTTUHF MINGW64 ~/Documents/ADS/ADSProject/Flask
$ curl -H "Content-Type: application/json" -X POST -d @z1.json http://ec2-52-27-35-131.us-west-2.compute.amazonaws.com:
5000/predictoutput1/
[{"ag_ratio": 18, "age": 65, "albumin": 16, "alkphos": 0.9, "direct_bilirubin": 0.1, "id": 1, "sgot": 3.3, "sgpt": 6.8, "tot_bilirubin": 0.7, "tot_proteins": 187, "gender_M
ale": 0, "is_patient": "Yes"}, {"ag_ratio": 100, "age": 62, "albumin": 64, "alkphos": 0.74, "direct_bilirubin": 5.5, "id": 2, "sgot": 3.2, "sgpt": 7.5, "tot_bilirubin": 10.
9, "tot_proteins": 699, "gender_Male": 1, "is_patient": "No"}, {"ag_ratio": 68, "age": 62, "albumin": 60, "alkphos": 0.89, "direct_bilirubin": 4.1, "id": 3, "sgot": 3.3, "s
gpt": 7.0, "tot_bilirubin": 7.3, "tot_proteins": 490, "gender_Male": 1, "is_patient": "No"}]
Akash@DESKTOP-BBTTUHF MINGW64 ~/Documents/ADS/ADSProject/Flask
$ cat z2.json | head -30
[
  {
    "id": 1,
    "Pregnancies": 6,
    "Glucose": 148,
    "BloodPressure": 72,
    "SkinThickness": 35,
    "Insulin": 0,
    "BMI": 33.6,
    "DiabetesPedigreeFunction": 0.627,
    "Age": 50
  },
  {
    "id": 2,
    "Pregnancies": 1,
    "Glucose": 85,
    "BloodPressure": 66,
    "SkinThickness": 29,
    "Insulin": 0,
    "BMI": 26.6,
    "DiabetesPedigreeFunction": 0.351,
    "Age": 31
  },
  {
    "id": 3,
    "Pregnancies": 8,
    "Glucose": 183,
    "BloodPressure": 64,
    "SkinThickness": 0,
    "Insulin": 0,

Akash@DESKTOP-BBTTUHF MINGW64 ~/Documents/ADS/ADSProject/Flask
$ curl -H "Content-Type: application/json" -X POST -d @z2.json http://ec2-52-27-35-131.us-west-2.compute.amazonaws.com:
5000/predictoutput2/
```


Admin Retraining the models:

(please earlier screen shots where admin trained the models for comparison)

Retraining Models for Kidney

Selecting the file for training the models

Liver

Error Metrics

	Model_Name	Dataset	F1_score	Precision_score	Recall_score	Accuracy_score	True_Positive	False_Positive	True_Negative	False_Negative
0	Random Forest Classifier	Training	1.000000	1.000000	1.000000	1.000000	145	0	119	0
1	Random Forest Classifier	Testing	0.596491	0.540000	0.519231	0.562500	41	25	27	21
2	Stochastic Gradient Decent	Training	0.700758	0.590674	0.770270	0.478992	128	17	57	62
3	Stochastic Gradient Decent	Testing	0.631579	0.400000	0.636364	0.291667	58	8	14	34
4	k nearest Neighbour	Training	0.829545	0.817814	0.789062	0.848739	118	27	101	18
5	k nearest Neighbour	Testing	0.561404	0.528302	0.482759	0.583333	36	30	28	20
6	Logistic Regression	Training	0.753788	0.754717	0.684932	0.840336	99	46	100	19
7	Logistic Regression	Testing	0.614035	0.576923	0.535714	0.625000	40	26	30	18
8	Support Vector Classifier	Training	0.712121	0.737931	0.625731	0.899160	81	64	107	12
9	Support Vector Classifier	Testing	0.657895	0.672269	0.563380	0.833333	35	31	40	8
10	Bernoulli Naive Bayes	Training	0.734848	0.755245	0.646707	0.907563	86	59	108	11
11	Bernoulli Naive Bayes	Testing	0.578947	0.600000	0.500000	0.750000	30	36	36	12
12	Gradient Naive Bayes	Training	0.704545	0.748387	0.607330	0.974790	70	75	116	3
13	Gradient Naive Bayes	Testing	0.578947	0.619048	0.500000	0.812500	27	39	39	9

Error Metrics After models get trained

Kidney

Error Metrics

	Model_Name	Dataset	F1_score	Precision_score	Recall_score	Accuracy_score	True_Positive	False_Positive	True_Negative	False_Negative
0	Random Forest Classifier	Training	1.000000	1.000000	1.00000	1.000000	42	0	32	0
1	Random Forest Classifier	Testing	0.968750	0.969697	1.00000	0.941176	15	0	16	1
2	Stochastic Gradient Decent	Training	1.000000	1.000000	1.00000	1.000000	42	0	32	0
3	Stochastic Gradient Decent	Testing	1.000000	1.000000	1.00000	1.000000	15	0	17	0
4	k nearest Neighbour	Training	1.000000	1.000000	1.00000	1.000000	42	0	32	0
5	k nearest Neighbour	Testing	1.000000	1.000000	1.00000	1.000000	15	0	17	0
6	Logistic Regression	Training	1.000000	1.000000	1.00000	1.000000	42	0	32	0
7	Logistic Regression	Testing	1.000000	1.000000	1.00000	1.000000	15	0	17	0
8	Support Vector Classifier	Training	0.959459	0.950820	1.00000	0.906250	42	0	29	3
9	Support Vector Classifier	Testing	1.000000	1.000000	1.00000	1.000000	15	0	17	0
10	Bernoulli Naive Bayes	Training	1.000000	1.000000	1.00000	1.000000	42	0	32	0
11	Bernoulli Naive Bayes	Testing	0.937500	0.937500	1.00000	0.882353	15	0	15	2
12	Gradient Naive Bayes	Training	0.986486	0.984127	1.00000	0.968750	42	0	31	1
13	Gradient Naive Bayes	Testing	0.531250	0.693878	0.53125	1.000000	0	15	17	0

Retraining Models for Liver

Error Metrics After models get trained

Liver

Error Metrics

	Model_Name	Dataset	F1_score	Precision_score	Recall_score	Accuracy_score	True_Positive	False_Positive	True_Negative	False_Negative
0	Random Forest Classifier	Training	1.000000	1.000000	1.000000	1.000	21	0	20	0
1	Random Forest Classifier	Testing	0.666667	0.625000	0.625000	0.625	7	3	5	3
2	Stochastic Gradient Decent	Training	0.951220	0.952381	0.909091	1.000	19	2	20	0
3	Stochastic Gradient Decent	Testing	0.777778	0.750000	0.750000	0.750	8	2	6	2
4	k nearest Neighbour	Training	0.853659	0.857143	0.818182	0.900	17	4	18	2
5	k nearest Neighbour	Testing	0.722222	0.705882	0.666667	0.750	7	3	6	2
6	Logistic Regression	Training	0.951220	0.952381	0.909091	1.000	19	2	20	0
7	Logistic Regression	Testing	0.777778	0.750000	0.750000	0.750	8	2	6	2
8	Support Vector Classifier	Training	0.878049	0.883721	0.826087	0.950	17	4	19	1
9	Support Vector Classifier	Testing	0.722222	0.705882	0.666667	0.750	7	3	6	2
10	Bernoulli Naive Bayes	Training	0.829268	0.829268	0.809524	0.850	17	4	17	3
11	Bernoulli Naive Bayes	Testing	0.777778	0.750000	0.750000	0.750	8	2	6	2
12	Gradient Naive Bayes	Training	0.853659	0.869565	0.769231	1.000	15	6	20	0
13	Gradient Naive Bayes	Testing	0.611111	0.461538	0.600000	0.375	8	2	3	5

Retraining Models for Diabetes

Error Metrics After models get trained

Diab

Error Metrics

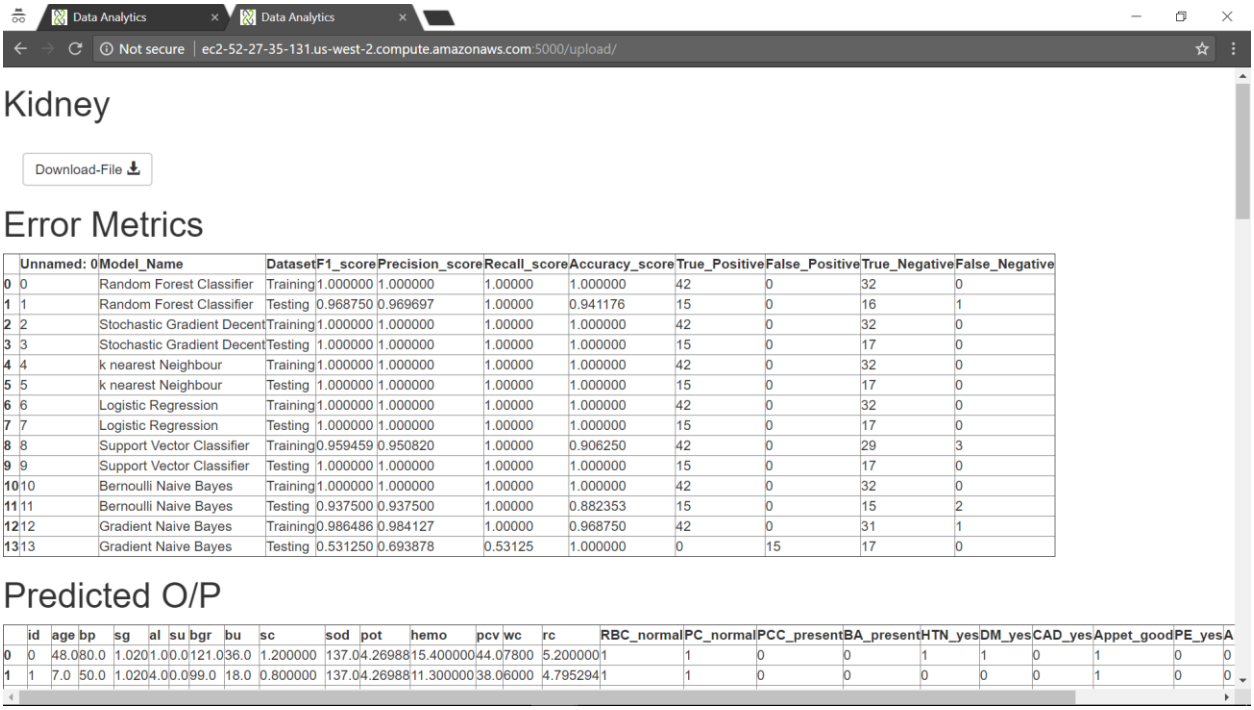
	Model_Name	Dataset	F1_score	Precision_score	Recall_score	Accuracy_score	True_Positive	False_Positive	True_Negative	False_Negative
0	Random Forest Classifier	Training	1.000000	1.000000	1.000000	1.000000	48	0	22	0
1	Random Forest Classifier	Testing	0.451613	0.000000	0.000000	0.000000	14	1	0	16
2	Stochastic Gradient Decent	Training	0.714286	0.583333	0.538462	0.636364	36	12	14	8
3	Stochastic Gradient Decent	Testing	0.645161	0.620690	0.692308	0.562500	11	4	9	7
4	k nearest Neighbour	Training	0.785714	0.651163	0.666667	0.636364	41	7	14	8
5	k nearest Neighbour	Testing	0.548387	0.363636	0.666667	0.250000	13	2	4	12
6	Logistic Regression	Training	0.757143	0.540541	0.666667	0.454545	43	5	10	12
7	Logistic Regression	Testing	0.645161	0.560000	0.777778	0.437500	13	2	7	9
8	Support Vector Classifier	Training	0.771429	0.428571	1.000000	0.272727	48	0	6	16
9	Support Vector Classifier	Testing	0.483871	0.000000	0.000000	0.000000	15	0	0	16
10	Bernoulli Naive Bayes	Training	0.685714	0.521739	0.500000	0.545455	36	12	12	10
11	Bernoulli Naive Bayes	Testing	0.612903	0.454545	0.833333	0.312500	14	1	5	11
12	Gradient Naive Bayes	Training	0.742857	0.590909	0.590909	0.590909	39	9	13	9
13	Gradient Naive Bayes	Testing	0.645161	0.592593	0.727273	0.500000	12	3	8	8

Perdicting Output after retraining models

Using user2

(Please notice the changes in the error metrics of the models with that in earlier screen shots)

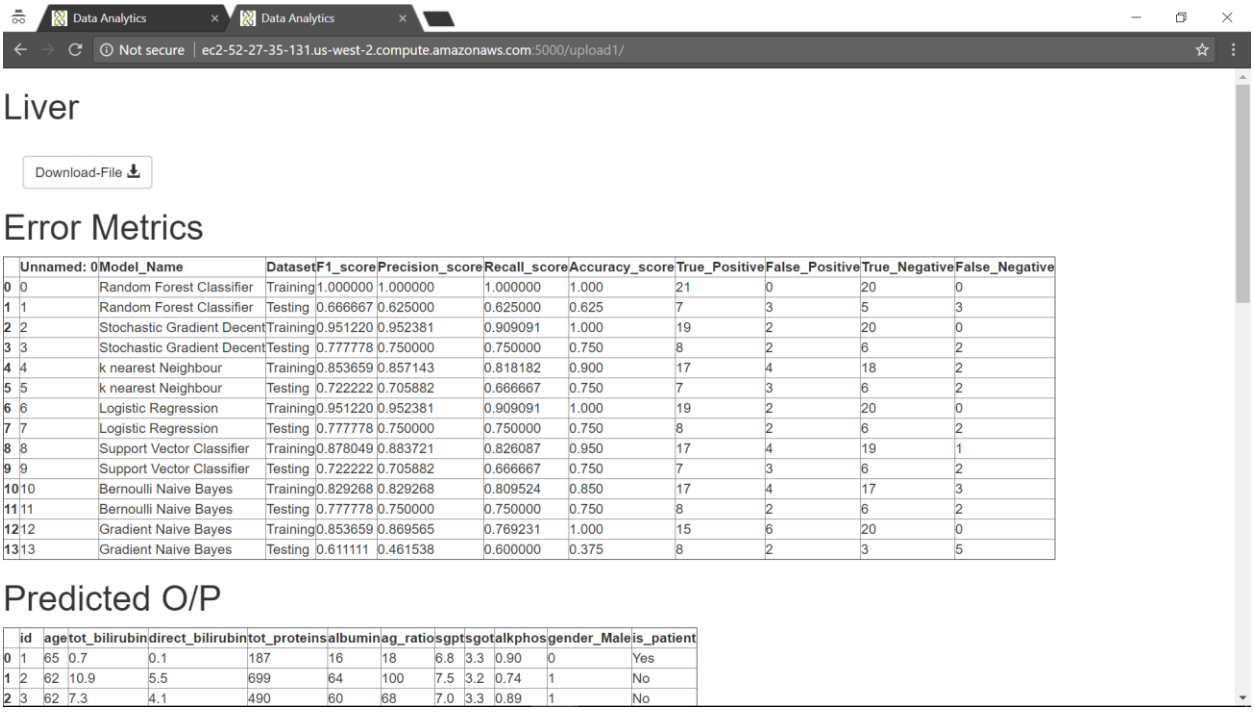
For kidney



	Unnamed: 0	Model_Name	Dataset	F1_score	Precision_score	Recall_score	Accuracy_score	True_Positive	False_Positive	True_Negative	False_Negative
0	0	Random Forest Classifier	Training	1.000000	1.000000	1.00000	1.000000	42	0	32	0
1	1	Random Forest Classifier	Testing	0.968750	0.969697	1.00000	0.941176	15	0	16	1
2	2	Stochastic Gradient Decent	Training	1.000000	1.000000	1.00000	1.000000	42	0	32	0
3	3	Stochastic Gradient Decent	Testing	1.000000	1.000000	1.00000	1.000000	15	0	17	0
4	4	k nearest Neighbour	Training	1.000000	1.000000	1.00000	1.000000	42	0	32	0
5	5	k nearest Neighbour	Testing	1.000000	1.000000	1.00000	1.000000	15	0	17	0
6	6	Logistic Regression	Training	1.000000	1.000000	1.00000	1.000000	42	0	32	0
7	7	Logistic Regression	Testing	1.000000	1.000000	1.00000	1.000000	15	0	17	0
8	8	Support Vector Classifier	Training	0.959459	0.950820	1.00000	0.906250	42	0	29	3
9	9	Support Vector Classifier	Testing	1.000000	1.000000	1.00000	1.000000	15	0	17	0
10	10	Bernoulli Naive Bayes	Training	1.000000	1.000000	1.00000	1.000000	42	0	32	0
11	11	Bernoulli Naive Bayes	Testing	0.937500	0.937500	1.00000	0.882353	15	0	15	2
12	12	Gradient Naive Bayes	Training	0.986486	0.984127	1.00000	0.968750	42	0	31	1
13	13	Gradient Naive Bayes	Testing	0.531250	0.693878	0.53125	1.000000	0	15	17	0

	id	age	bp	sg	al	su	bgr	bu	sc	sod	pot	hemo	pcv	wc	rc	RBC_normal	PC_normal	PCC_present	BA_present	HTN_yes	DM_yes	CAD_yes	Appet_good	PE_yes	A
0	0	48.0	80.0	1.020	1.00	0.0	121.0	36.0	1.200000	137.0	4.26988	15.400000	44.0	7800	5.200000	1	1	0	0	1	1	0	1	0	0
1	1	7.0	50.0	1.020	4.0	0.0	99.0	18.0	0.800000	137.0	4.26988	11.300000	38.0	6000	4.795294	1	1	0	0	0	0	0	1	0	0

For Liver



	Unnamed: 0	Model_Name	Dataset	F1_score	Precision_score	Recall_score	Accuracy_score	True_Positive	False_Positive	True_Negative	False_Negative
0	0	Random Forest Classifier	Training	1.000000	1.000000	1.000000	1.000	21	0	20	0
1	1	Random Forest Classifier	Testing	0.666667	0.625000	0.625000	0.625	7	3	5	3
2	2	Stochastic Gradient Decent	Training	0.951220	0.952381	0.909091	1.000	19	2	20	0
3	3	Stochastic Gradient Decent	Testing	0.777778	0.750000	0.750000	0.750	8	2	6	2
4	4	k nearest Neighbour	Training	0.853659	0.857143	0.818182	0.900	17	4	18	2
5	5	k nearest Neighbour	Testing	0.722222	0.705882	0.666667	0.750	7	3	6	2
6	6	Logistic Regression	Training	0.951220	0.952381	0.909091	1.000	19	2	20	0
7	7	Logistic Regression	Testing	0.777778	0.750000	0.750000	0.750	8	2	6	2
8	8	Support Vector Classifier	Training	0.878049	0.883721	0.826087	0.950	17	4	19	1
9	9	Support Vector Classifier	Testing	0.722222	0.705882	0.666667	0.750	7	3	6	2
10	10	Bernoulli Naive Bayes	Training	0.829268	0.829268	0.809524	0.850	17	4	17	3
11	11	Bernoulli Naive Bayes	Testing	0.777778	0.750000	0.750000	0.750	8	2	6	2
12	12	Gradient Naive Bayes	Training	0.853659	0.869565	0.769231	1.000	15	6	20	0
13	13	Gradient Naive Bayes	Testing	0.611111	0.461538	0.600000	0.375	8	2	3	5

	id	age	tot_bilirubin	direct_bilirubin	tot_proteins	albumin	ag_ratio	sgpt	sgot	alkphos	gender_Male	is_patient
0	1	65	0.7	0.1	187	16	18	6.8	3.3	0.90	0	Yes
1	2	62	10.9	5.5	699	64	100	7.5	3.2	0.74	1	No
2	3	62	7.3	4.1	490	60	68	7.0	3.3	0.89	1	No

For Diabetes

Data Analytics | Data Analytics

Not secure | ec2-52-27-35-131.us-west-2.compute.amazonaws.com:5000/upload2/

Diabetes

Download-File

Error Metrics

	Unnamed: 0	Model_Name	Dataset	F1_score	Precision_score	Recall_score	Accuracy_score	True_Positive	False_Positive	True_Negative	False_Negative
0	0	Random Forest Classifier	Training	1.000000	1.000000	1.000000	1.000000	48	0	22	0
1	1	Random Forest Classifier	Testing	0.451613	0.000000	0.000000	0.000000	14	1	0	16
2	2	Stochastic Gradient Decent	Training	0.714286	0.583333	0.538462	0.636364	36	12	14	8
3	3	Stochastic Gradient Decent	Testing	0.645161	0.620690	0.692308	0.562500	11	4	9	7
4	4	k nearest Neighbour	Training	0.785714	0.651163	0.666667	0.636364	41	7	14	8
5	5	k nearest Neighbour	Testing	0.548387	0.363636	0.666667	0.250000	13	2	4	12
6	6	Logistic Regression	Training	0.757143	0.540541	0.666667	0.454545	43	5	10	12
7	7	Logistic Regression	Testing	0.645161	0.560000	0.777778	0.437500	13	2	7	9
8	8	Support Vector Classifier	Training	0.771429	0.428571	1.000000	0.272727	48	0	6	16
9	9	Support Vector Classifier	Testing	0.483871	0.000000	0.000000	0.000000	15	0	0	16
10	10	Bernoulli Naive Bayes	Training	0.685714	0.521739	0.500000	0.545455	36	12	12	10
11	11	Bernoulli Naive Bayes	Testing	0.612903	0.454545	0.833333	0.312500	14	1	5	11
12	12	Gradient Naive Bayes	Training	0.742857	0.590909	0.590909	0.590909	39	9	13	9
13	13	Gradient Naive Bayes	Testing	0.645161	0.592593	0.727273	0.500000	12	3	8	8

Predicted O/P

	id	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	1	6	148	72	35	0	33.6	0.627	50	Yes
1	2	1	85	66	29	0	26.6	0.351	31	No
2	3	8	183	64	0	0	23.3	0.672	32	Yes

Rest Api output with retrained models

For Kidney

MINGW64:/c/Users/Akash/Documents/ADS/ADSProject/Flask


```
Yes"}, {"Age": 31, "BMI": 26.6, "BloodPressure": 66, "DiabetesPedigreeFunction": 0.351, "Glucose": 85, "Insulin": 0, "Pregnancies": 1, "SkinThickness": 29, "id": 2, "Outcom
e": "No"}, {"Age": 32, "BMI": 23.3, "BloodPressure": 64, "DiabetesPedigreeFunction": 0.672, "Glucose": 183, "Insulin": 0, "Pregnancies": 8, "SkinThickness": 0, "id": 3, "Ou
tcome": "No"}]
Akash@DESKTOP-BBTTUHF MINGW64 ~/Documents/ADS/ADSProject/Flask
$

Akash@DESKTOP-BBTTUHF MINGW64 ~/Documents/ADS/ADSProject/Flask
$

Akash@DESKTOP-BBTTUHF MINGW64 ~/Documents/ADS/ADSProject/Flask
$

Akash@DESKTOP-BBTTUHF MINGW64 ~/Documents/ADS/ADSProject/Flask
$

Akash@DESKTOP-BBTTUHF MINGW64 ~/Documents/ADS/ADSProject/Flask
$

Akash@DESKTOP-BBTTUHF MINGW64 ~/Documents/ADS/ADSProject/Flask
$

Akash@DESKTOP-BBTTUHF MINGW64 ~/Documents/ADS/ADSProject/Flask
$

Akash@DESKTOP-BBTTUHF MINGW64 ~/Documents/ADS/ADSProject/Flask
$ curl -H "Content-Type: application/json" -X POST -d @z.json http://ec2-52-27-35-131.us-west-2.compute.amazonaws.com:5
000/predictoutput/
[{"age": 62, "al": 2, "bgr": 423, "bp": 80, "bu": 53, "hemo": 9.6, "id": 2, "pcv": 31.0, "pot": 4.0166666667, "rc": 4.5166666667, "sc": 1.8, "sg": 1.01, "sod": 134.0, "su":
 3, "wc": 7500, "RBC_normal": 1, "PC_normal": 1, "PCC_present": 0, "HTN_yes": 0, "DM_yes": 1, "Appet_good": 0, "PE_yes": 0, "Ane_yes": 1, "Classification_ckd": "ckd"}, {"ag
e": 48, "al": 4, "bgr": 117, "bp": 70, "bu": 56, "hemo": 11.2, "id": 3, "pcv": 32.0, "pot": 2.5, "rc": 3.9, "sc": 3.8, "sg": 1.005, "sod": 111.0, "su": 0, "wc": 6700, "RBC_
normal": 1, "PC_normal": 0, "PCC_present": 1, "HTN_yes": 1, "DM_yes": 0, "Appet_good": 0, "PE_yes": 1, "Ane_yes": 1, "Classification_ckd": "ckd"}, {"age": 53, "al": 2, "bgr
": 70, "bp": 90, "bu": 107, "hemo": 9.5, "id": 9, "pcv": 29.0, "pot": 3.7, "rc": 3.7, "sc": 7.2, "sg": 1.02, "sod": 114.0, "su": 0, "wc": 12100, "RBC_normal": 0, "PC_normal
": 0, "PCC_present": 1, "HTN_yes": 1, "DM_yes": 1, "Appet_good": 0, "PE_yes": 0, "Ane_yes": 1, "Classification_ckd": "ckd"}, {"age": 29, "al": 0, "bgr": 83, "bp": 80, "bu":
 49, "hemo": 17.5, "id": 351, "pcv": 40.0, "pot": 3.3, "rc": 4.7, "sc": 0.9, "sg": 1.02, "sod": 139.0, "su": 0, "wc": 9900, "RBC_normal": 1, "PC_normal": 1, "PCC_present":
0, "HTN_yes": 0, "DM_yes": 0, "Appet_good": 1, "PE_yes": 0, "Ane_yes": 0, "Classification_ckd": "nckd"}, {"age": 37, "al": 0, "bgr": 109, "bp": 60, "bu": 47, "hemo": 15.0,
"id": 352, "pcv": 48.0, "pot": 4.9, "rc": 5.2, "sc": 1.1, "sg": 1.02, "sod": 141.0, "su": 0, "wc": 7000, "RBC_normal": 1, "PC_normal": 1, "PCC_present": 0, "HTN_yes": 0, "D
M_yes": 0, "Appet_good": 1, "PE_yes": 0, "Ane_yes": 0, "Classification_ckd": "nckd"}, {"age": 39, "al": 0, "bgr": 86, "bp": 60, "bu": 37, "hemo": 13.6, "id": 353, "pcv": 51
.0, "pot": 5.0, "rc": 4.5, "sc": 0.6, "sg": 1.02, "sod": 150.0, "su": 0, "wc": 5800, "RBC_normal": 1, "PC_normal": 1, "PCC_present": 0, "HTN_yes": 0, "DM_yes": 0, "Appet_go
od": 1, "PE_yes": 0, "Ane_yes": 0, "Classification_ckd": "nckd"}, {"age": 32, "al": 0, "bgr": 102, "bp": 60, "bu": 17, "hemo": 14.6, "id": 354, "pcv": 41.0, "pot": 4.7, "rc
": 5.1, "sc": 0.4, "sg": 1.025, "sod": 147.0, "su": 0, "wc": 6800, "RBC_normal": 1, "PC_normal": 1, "PCC_present": 0, "HTN_yes": 0, "DM_yes": 0, "Appet_good": 1, "PE_yes":
0, "Ane_yes": 0, "Classification_ckd": "nckd"}]
Akash@DESKTOP-BBTTUHF MINGW64 ~/Documents/ADS/ADSProject/Flask
$
```


For liver

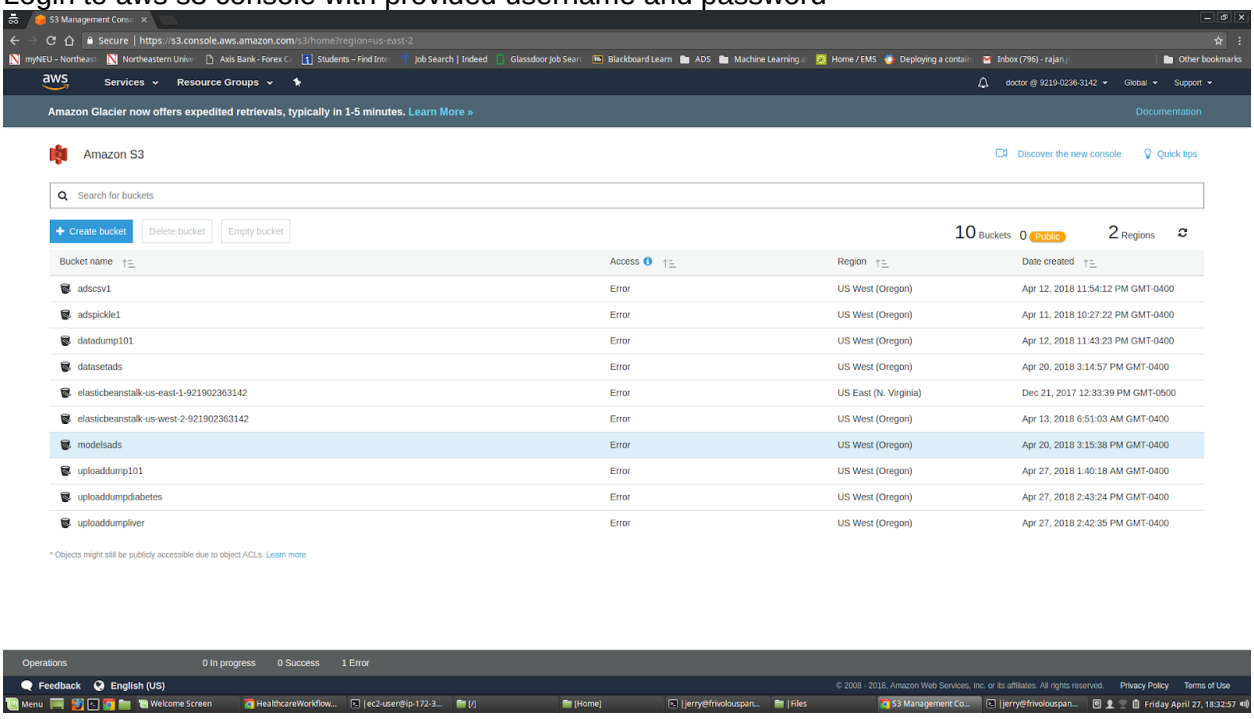
```
MINGW64:/c/Users/Akash/Documents/ADS/ADSProject/Flask
$

Akash@DESKTOP-BBTTUHF MINGW64 ~/Documents/ADS/ADSProject/Flask
$

Akash@DESKTOP-BBTTUHF MINGW64 ~/Documents/ADS/ADSProject/Flask
$

Akash@DESKTOP-BBTTUHF MINGW64 ~/Documents/ADS/ADSProject/Flask
$

Akash@DESKTOP-BBTTUHF MINGW64 ~/Documents/ADS/ADSProject/Flask
$

Akash@DESKTOP-BBTTUHF MINGW64 ~/Documents/ADS/ADSProject/Flask
$

Akash@DESKTOP-BBTTUHF MINGW64 ~/Documents/ADS/ADSProject/Flask
$ curl -H "Content-Type: application/json" -X POST -d @z.json http://ec2-52-27-35-131.us-west-2.compute.amazonaws.com:5
000/predictoutput/
[{"age": 62, "al": 2, "bgr": 423, "bp": 80, "bu": 53, "hemo": 9.6, "id": 2, "pcv": 31.0, "pot": 4.0166666667, "rc": 4.5166666667, "sc": 1.8, "sg": 1.01, "sod": 134.0, "su":
 3, "wc": 7500, "RBC_normal": 1, "PC_normal": 1, "PCC_present": 0, "HTN_yes": 0, "DM_yes": 1, "Appet_good": 0, "PE_yes": 0, "Ane_yes": 1, "Classification_ckd": "ckd"}, {"ag
e": 48, "al": 4, "bgr": 117, "bp": 70, "bu": 56, "hemo": 11.2, "id": 3, "pcv": 32.0, "pot": 2.5, "rc": 3.9, "sc": 3.8, "sg": 1.005, "sod": 111.0, "su": 0, "wc": 6700, "RBC_
normal": 1, "PC_normal": 0, "PCC_present": 1, "HTN_yes": 1, "DM_yes": 0, "Appet_good": 0, "PE_yes": 1, "Ane_yes": 1, "Classification_ckd": "ckd"}, {"age": 53, "al": 2, "bgr
": 70, "bp": 90, "bu": 107, "hemo": 9.5, "id": 9, "pcv": 29.0, "pot": 3.7, "rc": 3.7, "sc": 7.2, "sg": 1.02, "sod": 114.0, "su": 0, "wc": 12100, "RBC_normal": 0, "PC_normal
": 0, "PCC_present": 1, "HTN_yes": 1, "DM_yes": 1, "Appet_good": 0, "PE_yes": 0, "Ane_yes": 1, "Classification_ckd": "ckd"}, {"age": 29, "al": 0, "bgr": 83, "bp": 80, "bu":
 49, "hemo": 17.5, "id": 351, "pcv": 40.0, "pot": 3.3, "rc": 4.7, "sc": 0.9, "sg": 1.02, "sod": 139.0, "su": 0, "wc": 9900, "RBC_normal": 1, "PC_normal": 1, "PCC_present":
0, "HTN_yes": 0, "DM_yes": 0, "Appet_good": 1, "PE_yes": 0, "Ane_yes": 0, "Classification_ckd": "nckd"}, {"age": 37, "al": 0, "bgr": 109, "bp": 60, "bu": 47, "hemo": 15.0,
"id": 352, "pcv": 48.0, "pot": 4.9, "rc": 5.2, "sc": 1.1, "sg": 1.02, "sod": 141.0, "su": 0, "wc": 7000, "RBC_normal": 1, "PC_normal": 1, "PCC_present": 0, "HTN_yes": 0, "D
M_yes": 0, "Appet_good": 1, "PE_yes": 0, "Ane_yes": 0, "Classification_ckd": "nckd"}, {"age": 39, "al": 0, "bgr": 86, "bp": 60, "bu": 37, "hemo": 13.6, "id": 353, "pcv": 51
.0, "pot": 5.0, "rc": 4.5, "sc": 0.6, "sg": 1.02, "sod": 150.0, "su": 0, "wc": 5800, "RBC_normal": 1, "PC_normal": 1, "PCC_present": 0, "HTN_yes": 0, "DM_yes": 0, "Appet_go
od": 1, "PE_yes": 0, "Ane_yes": 0, "Classification_ckd": "nckd"}, {"age": 32, "al": 0, "bgr": 102, "bp": 60, "bu": 17, "hemo": 14.6, "id": 354, "pcv": 41.0, "pot": 4.7, "rc
": 5.1, "sc": 0.4, "sg": 1.025, "sod": 147.0, "su": 0, "wc": 6800, "RBC_normal": 1, "PC_normal": 1, "PCC_present": 0, "HTN_yes": 0, "DM_yes": 0, "Appet_good": 1, "PE_yes":
0, "Ane_yes": 0, "Classification_ckd": "nckd"}]
Akash@DESKTOP-BBTTUHF MINGW64 ~/Documents/ADS/ADSProject/Flask
$ curl -H "Content-Type: application/json" -X POST -d @z1.json http://ec2-52-27-35-131.us-west-2.compute.amazonaws.com:
5000/predictoutput1/
[{"ag_ratio": 18, "age": 65, "albumin": 16, "alkphos": 0.9, "direct_bilirubin": 0.1, "id": 1, "sgot": 3.3, "sgpt": 6.8, "tot_bilirubin": 0.7, "tot_proteins": 187, "gender_M
ale": 0, "is_patient": "Yes"}, {"ag_ratio": 100, "age": 62, "albumin": 64, "alkphos": 0.74, "direct_bilirubin": 5.5, "id": 2, "sgot": 3.2, "sgpt": 7.5, "tot_bilirubin": 10.
9, "tot_proteins": 699, "gender_Male": 1, "is_patient": "No"}, {"ag_ratio": 68, "age": 62, "albumin": 60, "alkphos": 0.89, "direct_bilirubin": 4.1, "id": 3, "sgot": 3.3, "s
gpt": 7.0, "tot_bilirubin": 7.3, "tot_proteins": 490, "gender_Male": 1, "is_patient": "No"}]
Akash@DESKTOP-BBTTUHF MINGW64 ~/Documents/ADS/ADSProject/Flask
$
```

For Diabetes

```
MINGW64:/c/Users/Akash/Documents/ADS/ADSProject/Flask

Akash@DESKTOP-BBTTUHF MINGW64 ~/Documents/ADS/ADSProject/Flask
$

Akash@DESKTOP-BBTTUHF MINGW64 ~/Documents/ADS/ADSProject/Flask
$

Akash@DESKTOP-BBTTUHF MINGW64 ~/Documents/ADS/ADSProject/Flask
$

Akash@DESKTOP-BBTTUHF MINGW64 ~/Documents/ADS/ADSProject/Flask
$ curl -H "Content-Type: application/json" -X POST -d @z.json http://ec2-52-27-35-131.us-west-2.compute.amazonaws.com:5
000/predictoutput/
[{"age": 62, "al": 2, "bgr": 423, "bp": 80, "bu": 53, "hemo": 9.6, "id": 2, "pcv": 31.0, "pot": 4.0166666667, "rc": 4.5166666667, "sc": 1.8, "sg": 1.01, "sod": 134.0, "su":
 3, "wc": 7500, "RBC_normal": 1, "PC_normal": 1, "PCC_present": 0, "HTN_yes": 0, "DM_yes": 1, "Appet_good": 0, "PE_yes": 0, "Ane_yes": 1, "Classification_ckd": "ckd"}, {"ag
e": 48, "al": 4, "bgr": 117, "bp": 70, "bu": 56, "hemo": 11.2, "id": 3, "pcv": 32.0, "pot": 2.5, "rc": 3.9, "sc": 3.8, "sg": 1.005, "sod": 111.0, "su": 0, "wc": 6700, "RBC_
normal": 1, "PC_normal": 0, "PCC_present": 1, "HTN_yes": 1, "DM_yes": 0, "Appet_good": 0, "PE_yes": 1, "Ane_yes": 1, "Classification_ckd": "ckd"}, {"age": 53, "al": 2, "bgr
": 70, "bp": 90, "bu": 107, "hemo": 9.5, "id": 9, "pcv": 29.0, "pot": 3.7, "rc": 3.7, "sc": 7.2, "sg": 1.02, "sod": 114.0, "su": 0, "wc": 12100, "RBC_normal": 0, "PC_normal
": 0, "PCC_present": 1, "HTN_yes": 1, "DM_yes": 1, "Appet_good": 0, "PE_yes": 0, "Ane_yes": 1, "Classification_ckd": "ckd"}, {"age": 29, "al": 0, "bgr": 83, "bp": 80, "bu":
 49, "hemo": 17.5, "id": 351, "pcv": 40.0, "pot": 3.3, "rc": 4.7, "sc": 0.9, "sg": 1.02, "sod": 139.0, "su": 0, "wc": 9900, "RBC_normal": 1, "PC_normal": 1, "PCC_present":
0, "HTN_yes": 0, "DM_yes": 0, "Appet_good": 1, "PE_yes": 0, "Ane_yes": 0, "Classification_ckd": "nckd"}, {"age": 37, "al": 0, "bgr": 109, "bp": 60, "bu": 47, "hemo": 15.0,
"id": 352, "pcv": 48.0, "pot": 4.9, "rc": 5.2, "sc": 1.1, "sg": 1.02, "sod": 141.0, "su": 0, "wc": 7000, "RBC_normal": 1, "PC_normal": 1, "PCC_present": 0, "HTN_yes": 0, "D
M_yes": 0, "Appet_good": 1, "PE_yes": 0, "Ane_yes": 0, "Classification_ckd": "nckd"}, {"age": 39, "al": 0, "bgr": 86, "bp": 60, "bu": 37, "hemo": 13.6, "id": 353, "pcv": 51
.0, "pot": 5.0, "rc": 4.5, "sc": 0.6, "sg": 1.02, "sod": 150.0, "su": 0, "wc": 5800, "RBC_normal": 1, "PC_normal": 1, "PCC_present": 0, "HTN_yes": 0, "DM_yes": 0, "Appet_go
od": 1, "PE_yes": 0, "Ane_yes": 0, "Classification_ckd": "nckd"}, {"age": 32, "al": 0, "bgr": 102, "bp": 60, "bu": 17, "hemo": 14.6, "id": 354, "pcv": 41.0, "pot": 4.7, "rc
": 5.1, "sc": 0.4, "sg": 1.025, "sod": 147.0, "su": 0, "wc": 6800, "RBC_normal": 1, "PC_normal": 1, "PCC_present": 0, "HTN_yes": 0, "DM_yes": 0, "Appet_good": 1, "PE_yes":
0, "Ane_yes": 0, "Classification_ckd": "nckd"}]
Akash@DESKTOP-BBTTUHF MINGW64 ~/Documents/ADS/ADSProject/Flask
$ curl -H "Content-Type: application/json" -X POST -d @z1.json http://ec2-52-27-35-131.us-west-2.compute.amazonaws.com:
5000/predictoutput1/
[{"ag_ratio": 18, "age": 65, "albumin": 16, "alkphos": 0.9, "direct_bilirubin": 0.1, "id": 1, "sgot": 3.3, "sgpt": 6.8, "tot_bilirubin": 0.7, "tot_proteins": 187, "gender_M
ale": 0, "is_patient": "Yes"}, {"ag_ratio": 100, "age": 62, "albumin": 64, "alkphos": 0.74, "direct_bilirubin": 5.5, "id": 2, "sgot": 3.2, "sgpt": 7.5, "tot_bilirubin": 10.
9, "tot_proteins": 699, "gender_Male": 1, "is_patient": "No"}, {"ag_ratio": 68, "age": 62, "albumin": 60, "alkphos": 0.89, "direct_bilirubin": 4.1, "id": 3, "sgot": 3.3, "s
gpt": 7.0, "tot_bilirubin": 7.3, "tot_proteins": 490, "gender_Male": 1, "is_patient": "No"}]
Akash@DESKTOP-BBTTUHF MINGW64 ~/Documents/ADS/ADSProject/Flask
$ curl -H "Content-Type: application/json" -X POST -d @z2.json http://ec2-52-27-35-131.us-west-2.compute.amazonaws.com:
5000/predictoutput2/
[{"Age": 50, "BMI": 33.6, "BloodPressure": 72, "DiabetesPedigreeFunction": 0.627, "Glucose": 148, "Insulin": 0, "Pregnancies": 6, "SkinThickness": 35, "id": 1, "Outcome": "
Yes"}, {"Age": 31, "BMI": 26.6, "BloodPressure": 66, "DiabetesPedigreeFunction": 0.351, "Glucose": 85, "Insulin": 0, "Pregnancies": 1, "SkinThickness": 29, "id": 2, "Outcom
e": "No"}, {"Age": 32, "BMI": 23.3, "BloodPressure": 64, "DiabetesPedigreeFunction": 0.672, "Glucose": 183, "Insulin": 0, "Pregnancies": 8, "SkinThickness": 0, "id": 3, "Ou
tcome": "No"}]
Akash@DESKTOP-BBTTUHF MINGW64 ~/Documents/ADS/ADSProject/Flask
$
```

Airflow & Lambda

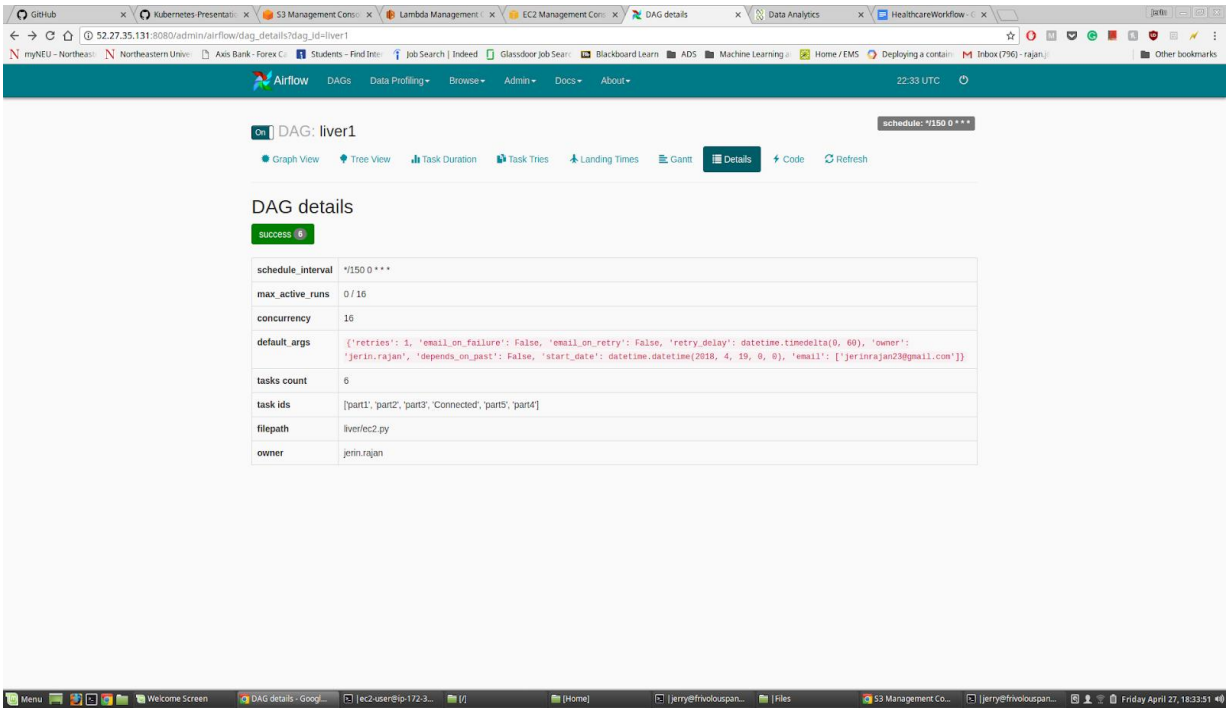
Step 1 :
Login to aws s3 console with provided username and password



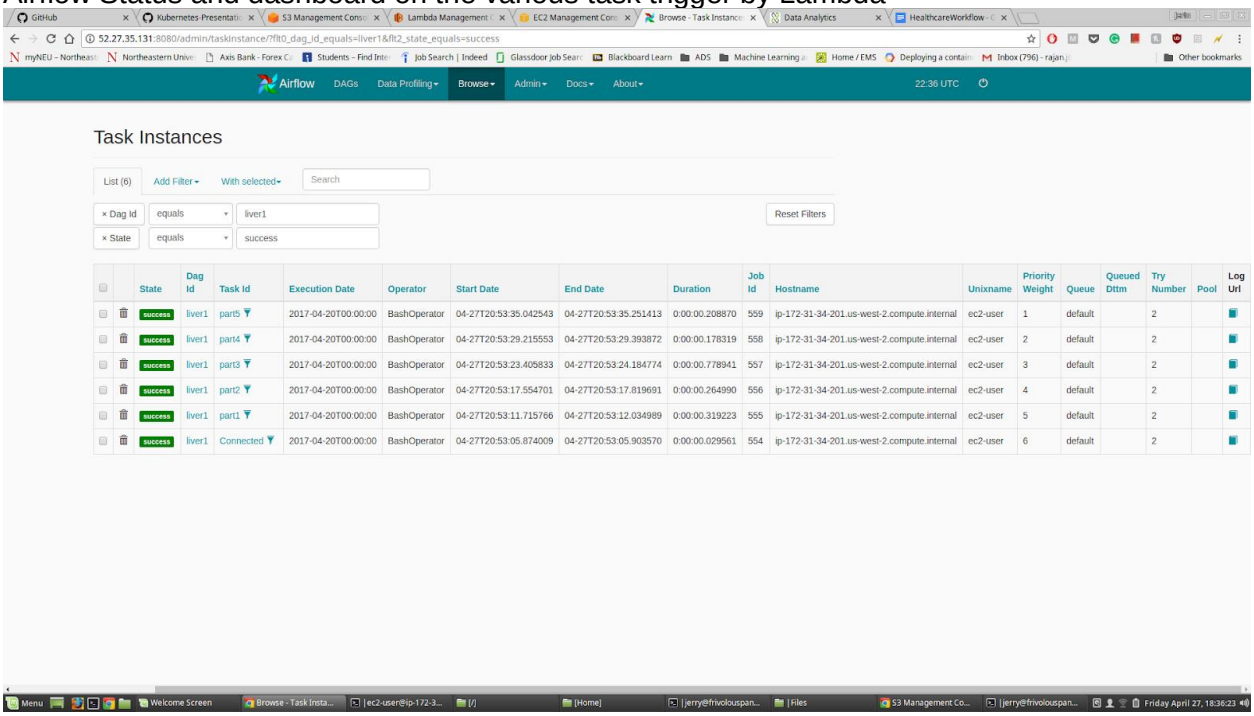
Step 2 :
Upload the files into the desired folder according to the disease

Step 3 :
Once the data is uploaded, the lambda function will be triggered and the data will scripts present in the lambda function.
Lambda features:

1. It work download private key from S3 bucket where it is stored
2. Create a connection to the system where the system is hosted via ssh-ing into it with the help of paramiko package
3. With the help of ssh we will trigger the airflow scripts
4. Airflow scripts will perform the execution of various tasks in the created DAG.
5. Dag consists of task like downloading new data from the s3, data cleaning, modeling, storing the various measures and restoring the new matrix of data to S3


52.27.35.131:8080/admin/airflow/dag_details?dag_id=liver1
Airflow
DAGs
Data Profiling
Browse
Admin
Docs
About
22:33 UTC
On | DAG: liver1
schedule: */150 0 * * *
Graph View
Tree View
Task Duration
Task Tries
Landing Times
Gantt
Details
Code
Refresh
DAG details
success 6
schedule_interval
*/150 0 * * *
max_active_runs
0 / 16
concurrency
16
default_args
{'retries': 1, 'email_on_failure': False, 'email_on_retry': False, 'retry_delay': datetime.timedelta(0, 60), 'owner': 'jerin.rajan', 'depends_on_past': False, 'start_date': datetime.datetime(2018, 4, 19, 0, 0), 'email': ['jerinrajan23@gmail.com']}
tasks count
6
task ids
['part1', 'part2', 'part3', 'Connected', 'part5', 'part4']
filepath
liver/ec2.py
owner
jerin.rajan

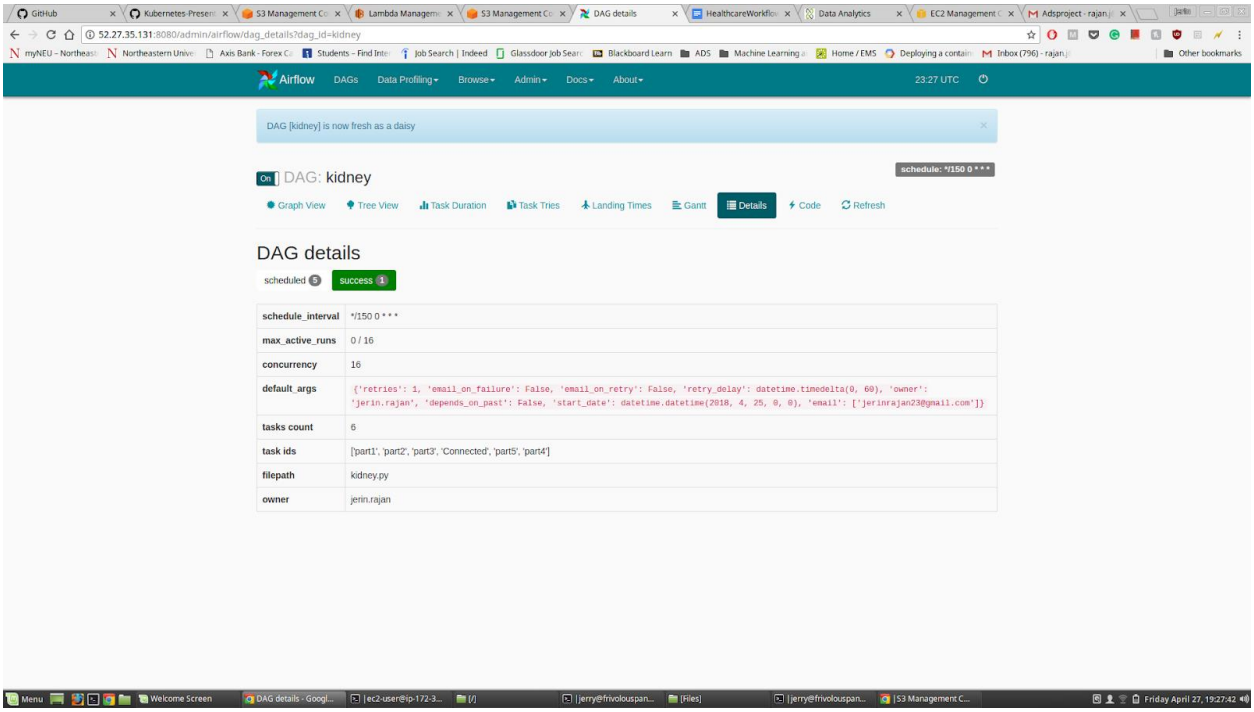
Airflow Status and dashboard on the various task trigger by Lambda

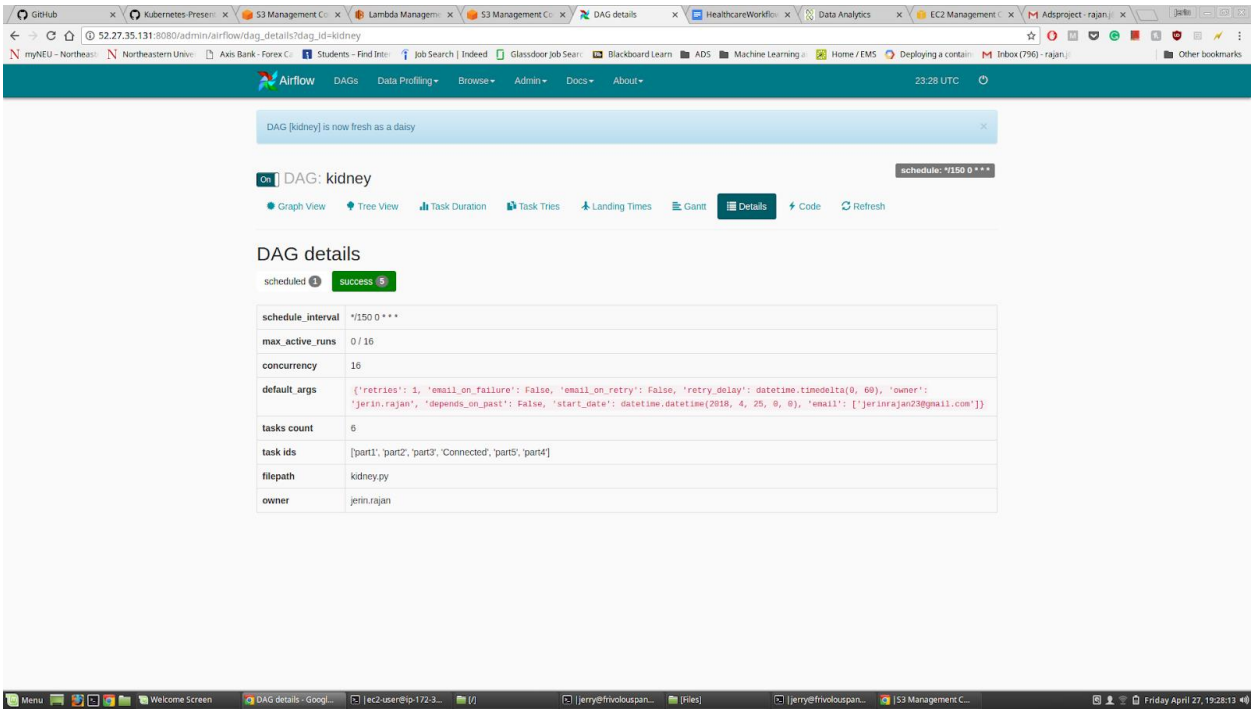


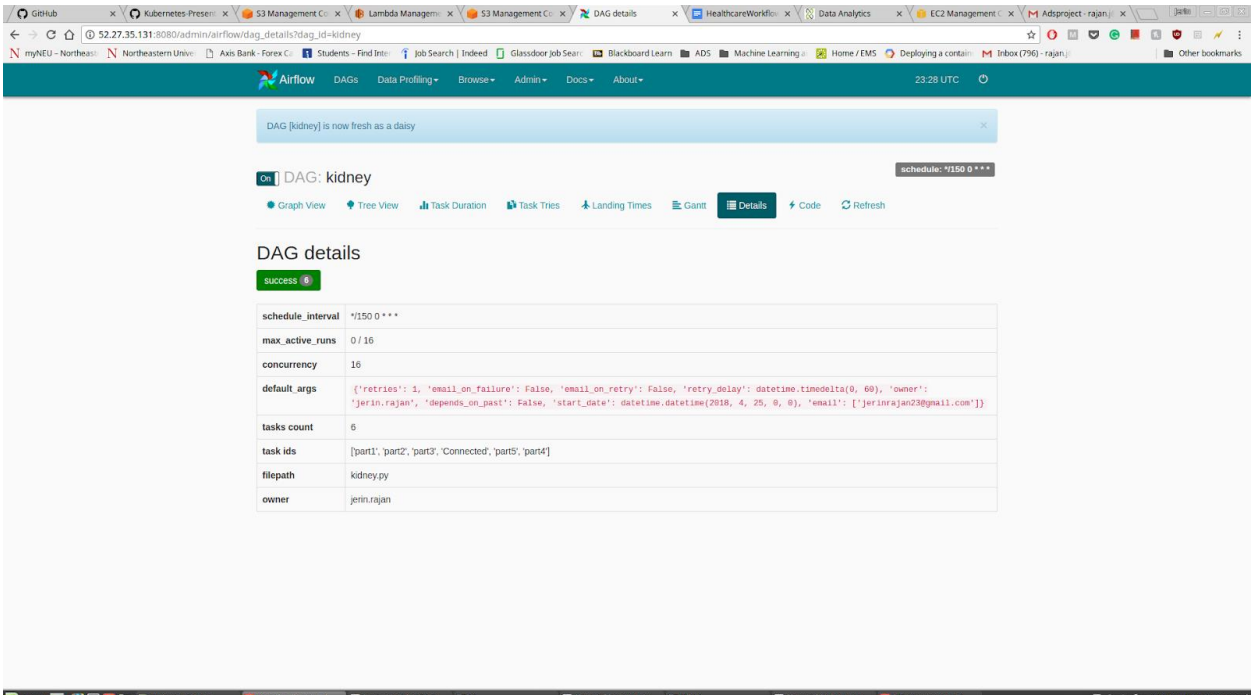
State	Dag Id	Task Id	Execution Date	Operator	Start Date	End Date	Duration	Job Id	Hostname	Unixname	Priority Weight	Queue	Queued Dttm	Try Number	Pool	Log Url
success	liver1	part5	2017-04-20T00:00:00	BashOperator	04-27T20:53:35.042543	04-27T20:53:35.251413	0:00:00.208870	559	ip-172-31-34-201.us-west-2.compute.internal	ec2-user	1	default		2		
success	liver1	part4	2017-04-20T00:00:00	BashOperator	04-27T20:53:29.215553	04-27T20:53:29.393872	0:00:00.178319	558	ip-172-31-34-201.us-west-2.compute.internal	ec2-user	2	default		2		
success	liver1	part3	2017-04-20T00:00:00	BashOperator	04-27T20:53:23.405833	04-27T20:53:24.184774	0:00:00.778941	557	ip-172-31-34-201.us-west-2.compute.internal	ec2-user	3	default		2		
success	liver1	part2	2017-04-20T00:00:00	BashOperator	04-27T20:53:17.554701	04-27T20:53:17.819691	0:00:00.264990	556	ip-172-31-34-201.us-west-2.compute.internal	ec2-user	4	default		2		
success	liver1	part1	2017-04-20T00:00:00	BashOperator	04-27T20:53:11.715766	04-27T20:53:12.034989	0:00:00.319223	555	ip-172-31-34-201.us-west-2.compute.internal	ec2-user	5	default		2		
success	liver1	Connected	2017-04-20T00:00:00	BashOperator	04-27T20:53:05.874009	04-27T20:53:05.903570	0:00:00.029561	554	ip-172-31-34-201.us-west-2.compute.internal	ec2-user	6	default		2		

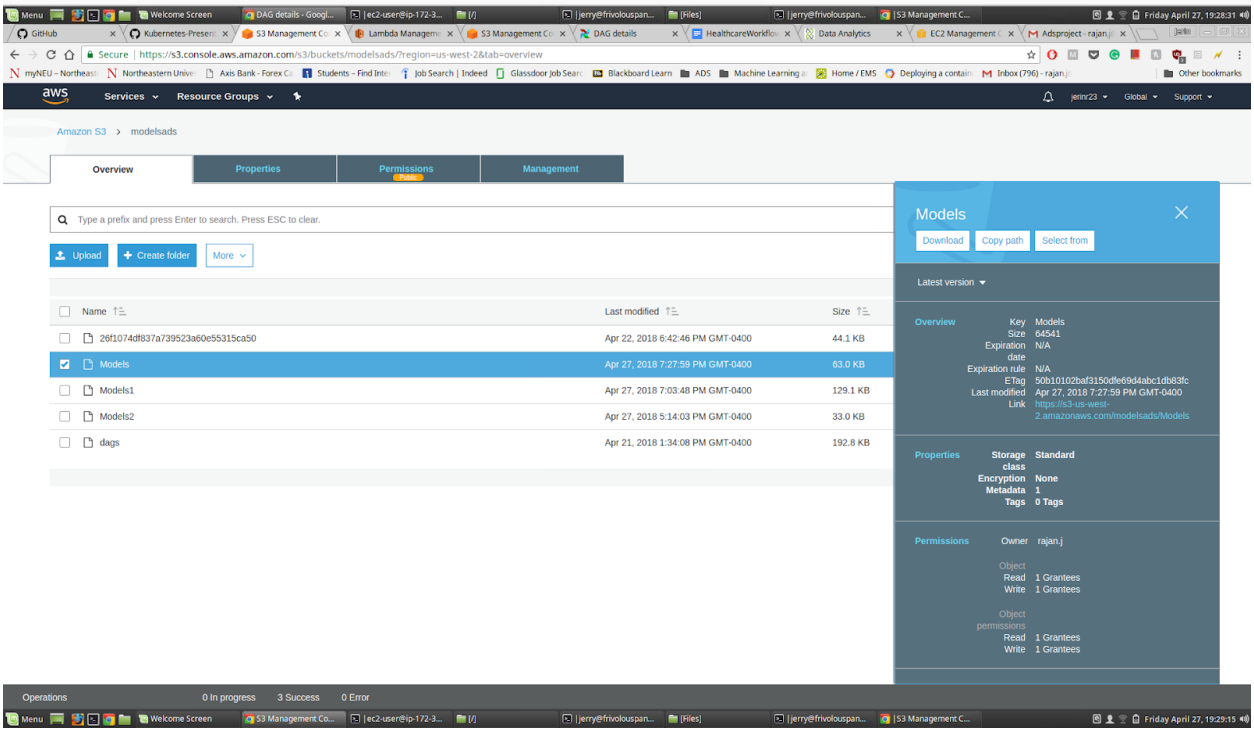
Step 4 :
Login to the Portal which has flask application hosted http://52.27.35.131:5000
1. Login user with the provided credentials.
2. Upload the data for error matrix and values others inferences.

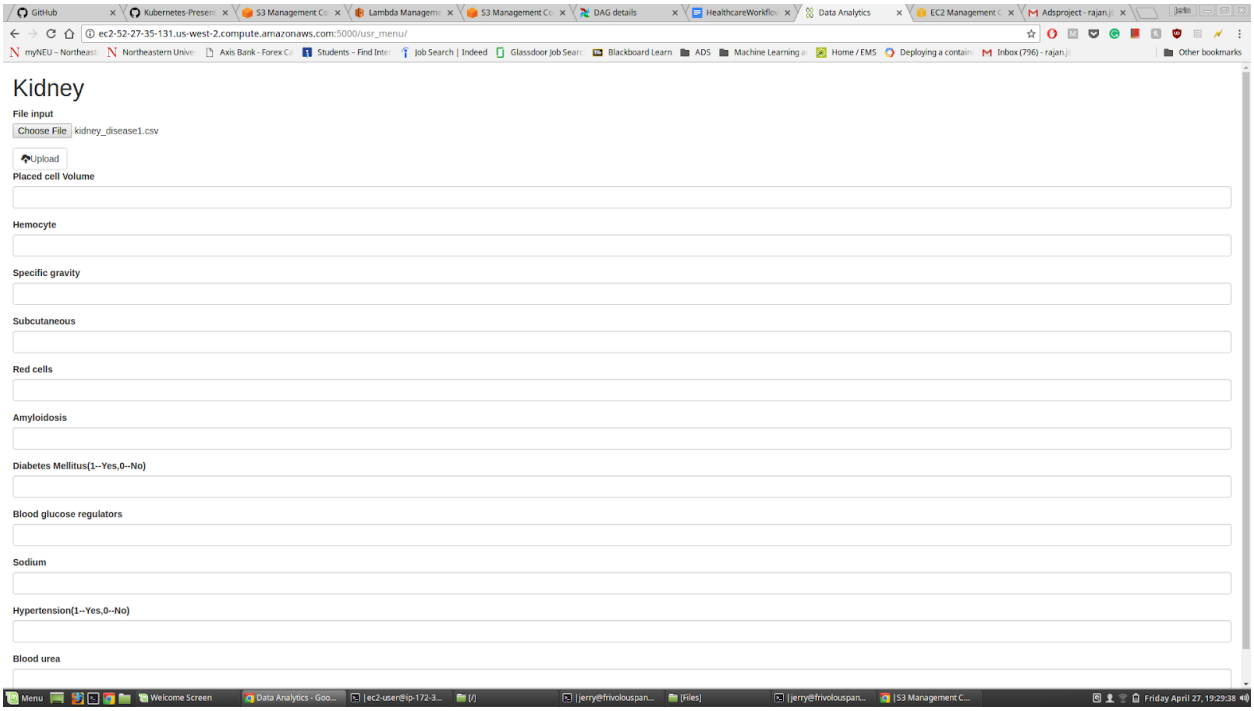
i. Airflow is working

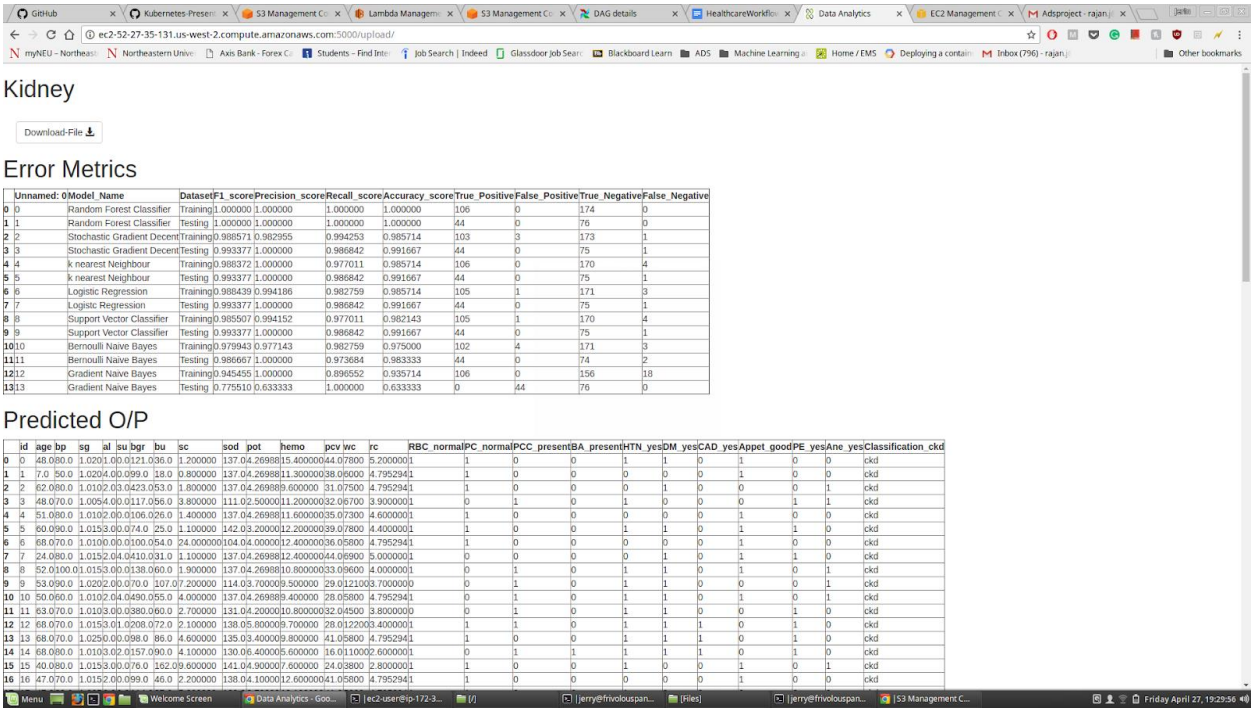



52.27.35.131:8080/admin/airflow/dag_details?dag_id=kidney
Airflow
DAGs
Data Profiling
Browse
Admin
Docs
About
23:28 UTC
DAG [kidney] is now fresh as a daisy
On
DAG: kidney
schedule: */150 0 * * *
Graph View
Tree View
Task Duration
Task Tries
Landing Times
Gantt
Details
Code
Refresh
DAG details
scheduled 1
success 5
schedule_interval
*/150 0 * * *
max_active_runs
0 / 16
concurrency
16
default_args
{'retries': 1, 'email_on_failure': False, 'email_on_retry': False, 'retry_delay': datetime.timedelta(0, 60), 'owner': 'jerin.rajan', 'depends_on_past': False, 'start_date': datetime.datetime(2018, 4, 25, 0, 0), 'email': ['jerinrajan23@gmail.com']}
tasks count
6
task ids
['part1', 'part2', 'part3', 'Connected', 'part5', 'part4']
filepath
kidney.py
owner
jerin.rajan


52.27.35.131:8080/admin/airflow/dag_details?dag_id=kidney
Airflow
DAGs
Data Profiling
Browse
Admin
Docs
About
23:28 UTC
DAG [kidney] is now fresh as a daisy
On
DAG: kidney
schedule: */150 0 * * *
Graph View
Tree View
Task Duration
Task Tries
Landing Times
Gantt
Details
Code
Refresh
DAG details
success 6
schedule_interval */150 0 * * *
max_active_runs 0 / 16
concurrency 16
default_args {'retries': 1, 'email_on_failure': False, 'email_on_retry': False, 'retry_delay': datetime.timedelta(0, 60), 'owner': 'jerin.rajan', 'depends_on_past': False, 'start_date': datetime.datetime(2018, 4, 25, 0, 0), 'email': ['jerinrajan23@gmail.com']}
tasks count 6
task ids ['part1', 'part2', 'part3', 'Connected', 'part5', 'part4']
filepath kidney.py
owner jerin.rajan

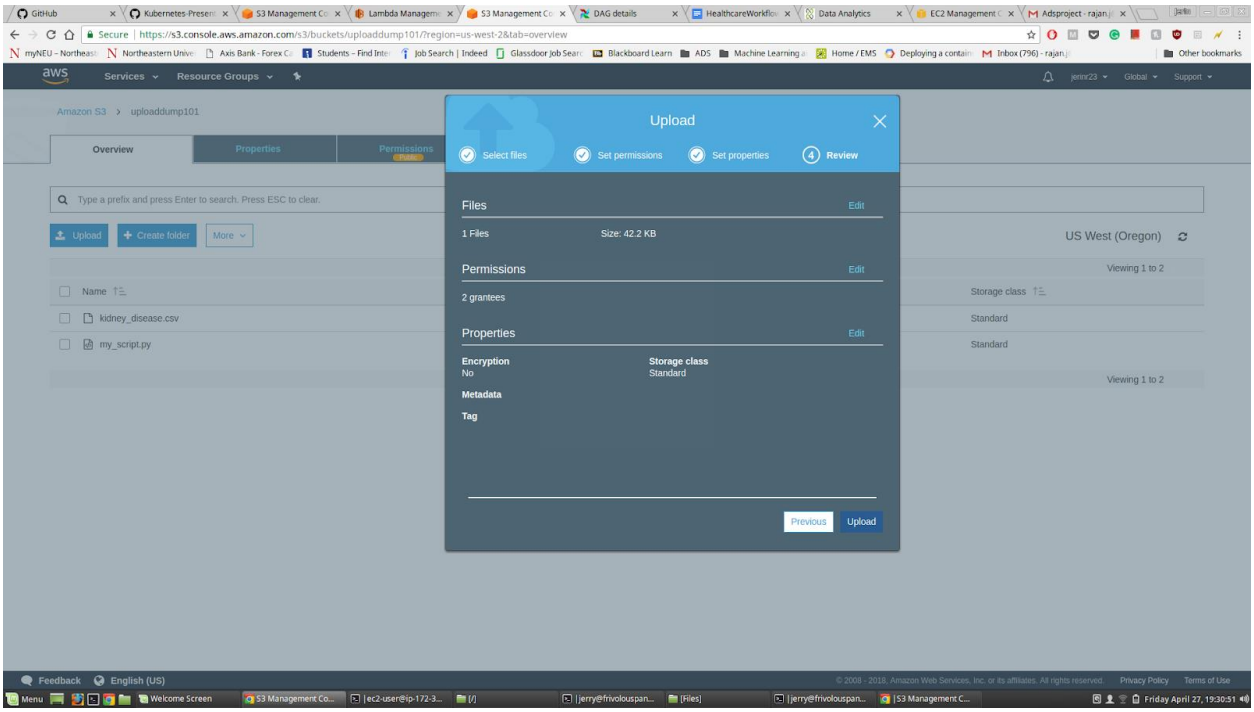

Secure | https://s3.console.aws.amazon.com/s3/buckets/modelsads/?region=us-west-2&tab=overview
aws
Services
Resource Groups
Amazon S3 > modelsads
Overview
Properties
Permissions
Management
Type a prefix and press Enter to search. Press ESC to clear.
Upload
Create folder
More
Name
Last modified
Size
26f1074df837a739523a60e55315ca50 Apr 22, 2018 6:42:46 PM GMT-0400 44.1 KB
Models Apr 27, 2018 7:27:59 PM GMT-0400 63.0 KB
Models1 Apr 27, 2018 7:03:48 PM GMT-0400 129.1 KB
Models2 Apr 27, 2018 5:14:03 PM GMT-0400 33.0 KB
dags Apr 21, 2018 1:34:08 PM GMT-0400 192.8 KB
Models
Download
Copy path
Select from
Latest version
Overview
Key Models
Size 64541
Expiration date N/A
Expiration rule N/A
ETag 50b10102baf3150dfe69d4abc1db83fc
Last modified Apr 27, 2018 7:27:59 PM GMT-0400
Link https://s3-us-west-2.amazonaws.com/modelsads/Models
Properties
Storage class Standard
Encryption None
Metadata 1
Tags 0 Tags
Permissions
Owner rajan.j
Object
Read 1 Grantees
Write 1 Grantees
Object permissions
Read 1 Grantees
Write 1 Grantees
Operations
0 In progress
3 Success
0 Error

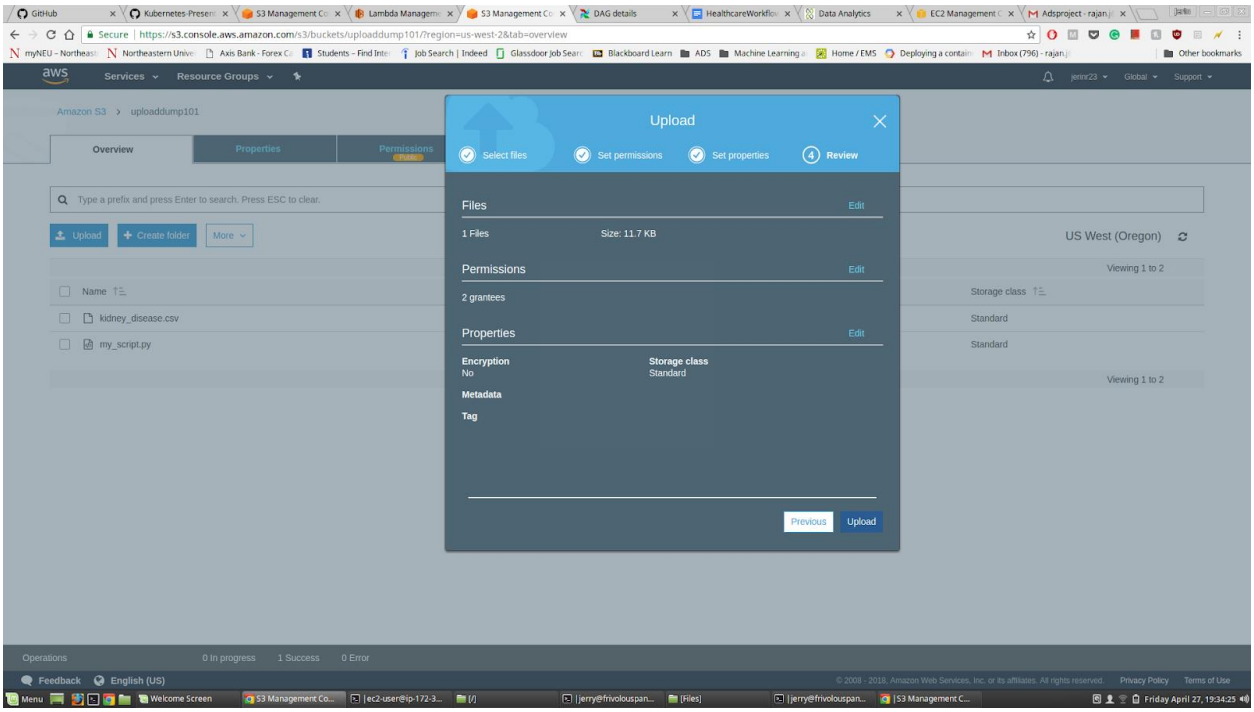


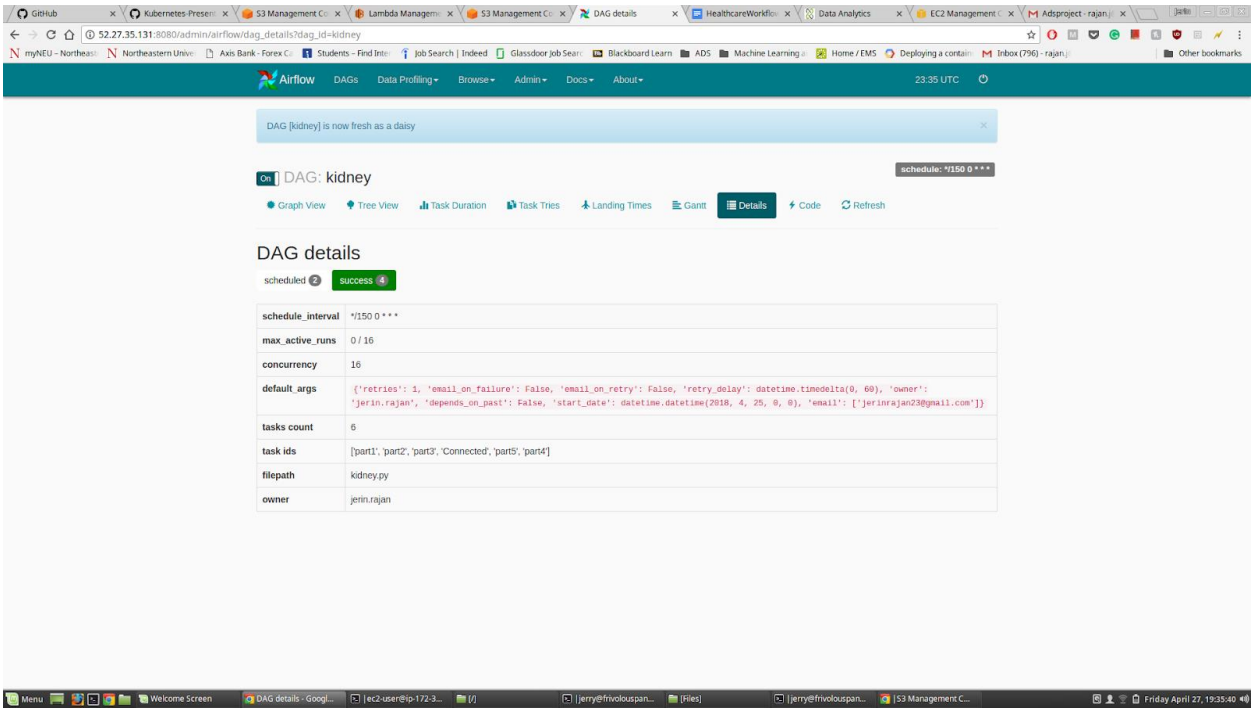


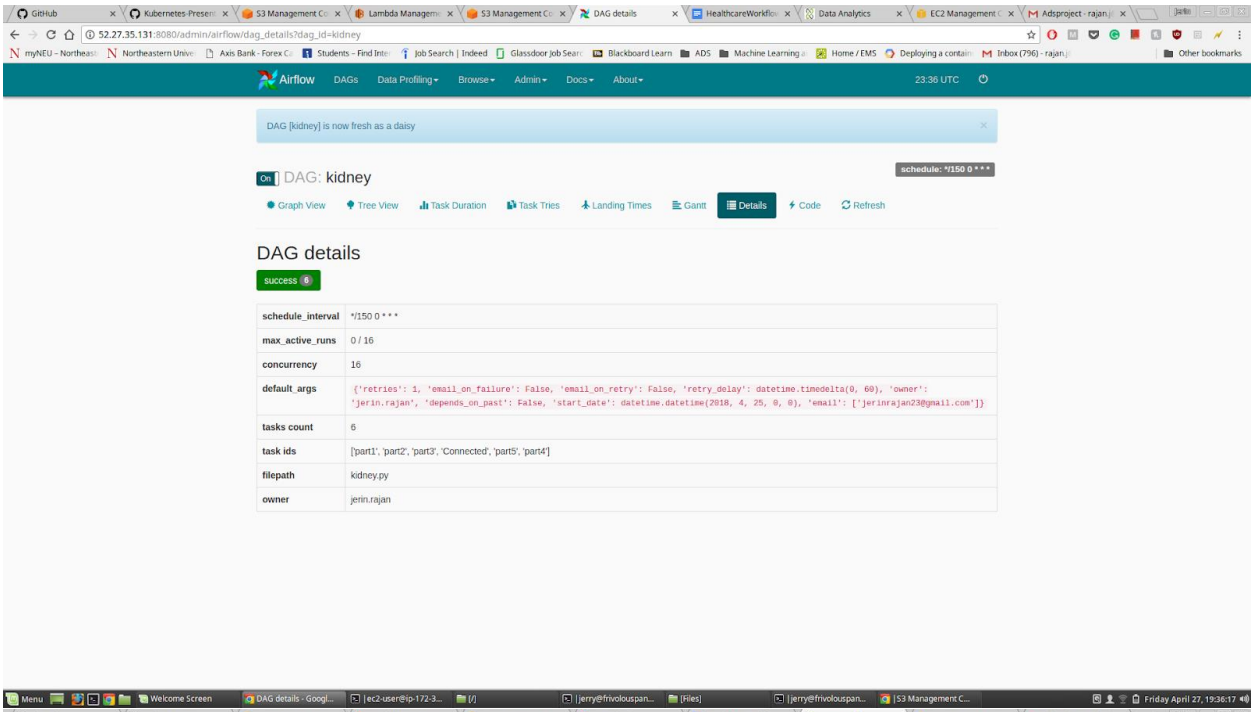
	Unnamed: 0	Model_Name	Dataset	F1_score	Precision_score	Recall_score	Accuracy_score	True_Positive	False_Positive	True_Negative	False_Negative
0	0	Random Forest Classifier	Training	1.000000	1.000000	1.000000	1.000000	106	0	174	0
1	1	Random Forest Classifier	Testing	1.000000	1.000000	1.000000	1.000000	44	0	76	0
2	2	Stochastic Gradient Decent	Training	0.988571	0.982955	0.994253	0.985714	103	3	173	1
3	3	Stochastic Gradient Decent	Testing	0.993377	1.000000	0.986842	0.991667	44	0	75	1
4	4	k nearest Neighbour	Training	0.988372	1.000000	0.977011	0.985714	106	0	170	4
5	5	k nearest Neighbour	Testing	0.993377	1.000000	0.986842	0.991667	44	0	75	1
6	6	Logistic Regression	Training	0.988439	0.994186	0.982759	0.985714	105	1	171	3
7	7	Logistc Regression	Testing	0.993377	1.000000	0.986842	0.991667	44	0	75	1
8	8	Support Vector Classifier	Training	0.985507	0.994152	0.977011	0.982143	105	1	170	4
9	9	Support Vector Classifier	Testing	0.993377	1.000000	0.986842	0.991667	44	0	75	1
10	10	Bernoulli Naive Bayes	Training	0.979943	0.977143	0.982759	0.975000	102	4	171	3
11	11	Bernoulli Naive Bayes	Testing	0.986667	1.000000	0.973684	0.983333	44	0	74	2
12	12	Gradient Naive Bayes	Training	0.945455	1.000000	0.896552	0.935714	106	0	156	18
13	13	Gradient Naive Bayes	Testing	0.775510	0.633333	1.000000	0.633333	0	44	76	0

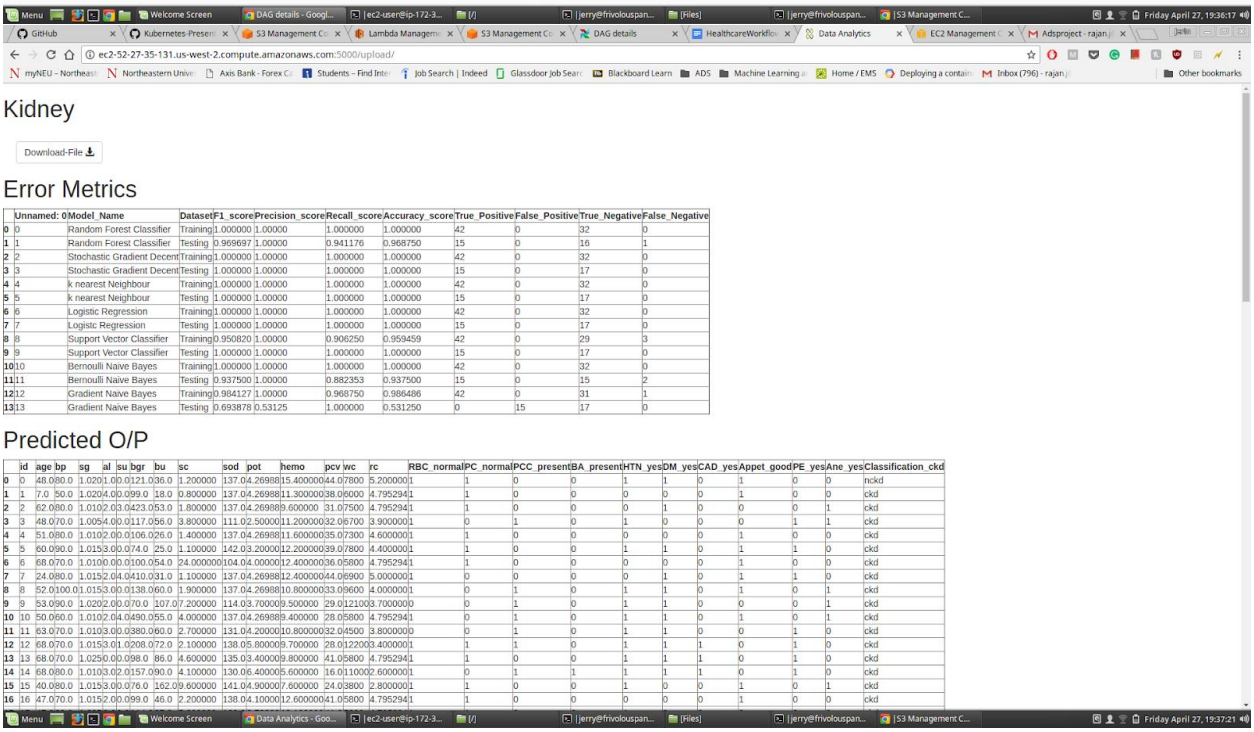
	id	age	bp	sg	al	su	bgr	bu	sc	sod	pot	hemo	pcv	wc	rc	RBC_normal	PC_normal	PCC_present	BA_present	HTN_yes	DM_yes	CAD_yes	Appet_good	PE_yes	Ane_yes	Classification_ckd
0	0	48.0	80.0	1.020	1.0	0.0	121.0	36.0	1.200000	137.0	4.26988	15.400000	44.0	7800	5.200000	1	1	0	0	1	1	0	1	0	0	ckd
1	1	7.0	50.0	1.020	4.0	0.0	99.0	18.0	0.800000	137.0	4.26988	11.300000	38.0	6000	4.795294	1	1	0	0	0	0	0	1	0	0	ckd
2	2	62.0	80.0	1.010	2.0	3.0	423.0	53.0	1.800000	137.0	4.26988	9.600000	31.0	7500	4.795294	1	1	0	0	0	1	0	0	0	1	ckd
3	3	48.0	70.0	1.005	4.0	0.0	117.0	56.0	3.800000	111.0	2.50000	11.200000	32.0	6700	3.900000	1	0	1	0	1	0	0	0	1	1	ckd
4	4	51.0	80.0	1.010	2.0	0.0	106.0	26.0	1.400000	137.0	4.26988	11.600000	35.0	7300	4.600000	1	1	0	0	0	0	0	1	0	0	ckd
5	5	60.0	90.0	1.015	3.0	0.0	74.0	25.0	1.100000	142.0	3.20000	12.200000	39.0	7800	4.400000	1	1	0	0	1	1	0	1	1	0	ckd
6	6	68.0	70.0	1.010	0.0	0.0	100.0	54.0	24.000000	104.0	4.00000	12.400000	36.0	5800	4.795294	1	1	0	0	0	0	0	1	0	0	ckd
7	7	24.0	80.0	1.015	2.0	4.0	410.0	31.0	1.100000	137.0	4.26988	12.400000	44.0	6900	5.000000	1	0	0	0	0	1	0	1	1	0	ckd
8	8	52.0	100.0	1.015	3.0	0.0	138.0	60.0	1.900000	137.0	4.26988	10.800000	33.0	9600	4.000000	1	0	1	0	1	1	0	1	0	1	ckd
9	9	53.0	90.0	1.020	2.0	0.0	70.0	107.0	7.200000	114.0	3.70000	9.500000	29.0	12100	3.700000	0	0	1	0	1	1	0	0	0	1	ckd
10	10	50.0	60.0	1.010	2.0	4.0	490.0	55.0	4.000000	137.0	4.26988	9.400000	28.0	5800	4.795294	1	0	1	0	1	1	0	1	0	1	ckd
11	11	63.0	70.0	1.010	3.0	0.0	380.0	60.0	2.700000	131.0	4.20000	10.800000	32.0	4500	3.800000	0	0	1	0	1	1	0	0	1	0	ckd
12	12	68.0	70.0	1.015	3.0	1.0	208.0	72.0	2.100000	138.0	5.80000	9.700000	28.0	12200	3.400000	1	1	0	1	1	1	0	1	0	0	ckd
13	13	68.0	70.0	1.025	0.0	0.0	98.0	86.0	4.600000	135.0	3.40000	9.800000	41.0	5800	4.795294	1	1	0	0	1	1	1	0	1	0	ckd
14	14	68.0	80.0	1.010	3.0	2.0	157.0	90.0	4.100000	130.0	6.40000	5.600000	16.0	11000	2.600000	1	0	1	1	1	1	1	0	1	0	ckd
15	15	40.0	80.0	1.015	3.0	0.0	76.0	162.0	9.600000	141.0	4.90000	7.600000	24.0	3800	2.800000	1	1	0	0	1	0	0	1	0	1	ckd
16	16	47.0	70.0	1.015	2.0	0.0	99.0	46.0	2.200000	138.0	4.10000	12.600000	41.0	5800	4.795294	1	1	0	0	0	0	0	1	0	0	ckd


Upload
Select files
Set permissions
Set properties
4 Review
Files
Edit
1 Files
Size: 42.2 KB
Permissions
Edit
2 grantees
Properties
Edit
Encryption
No
Storage class
Standard
Metadata
Tag
Previous
Upload
Amazon S3 > uploaddump101
Overview
Properties
Permissions
Upload
Create folder
More
US West (Oregon)
Name
Storage class
kidney_disease.csv
Standard
my_script.py
Standard
Feedback
English (US)


Upload
Select files
Set permissions
Set properties
4 Review
Files
Edit
1 Files
Size: 11.7 KB
Permissions
Edit
2 grantees
Properties
Edit
Encryption
No
Storage class
Standard
Metadata
Tag
Previous
Upload
Amazon S3 > uploaddump101
Overview
Properties
Permissions
Upload
Create folder
More
US West (Oregon)
Viewing 1 to 2
Name
Storage class
kidney_disease.csv
Standard
my_script.py
Standard
Viewing 1 to 2
Operations
0 In progress
1 Success
0 Error
Feedback
English (US)

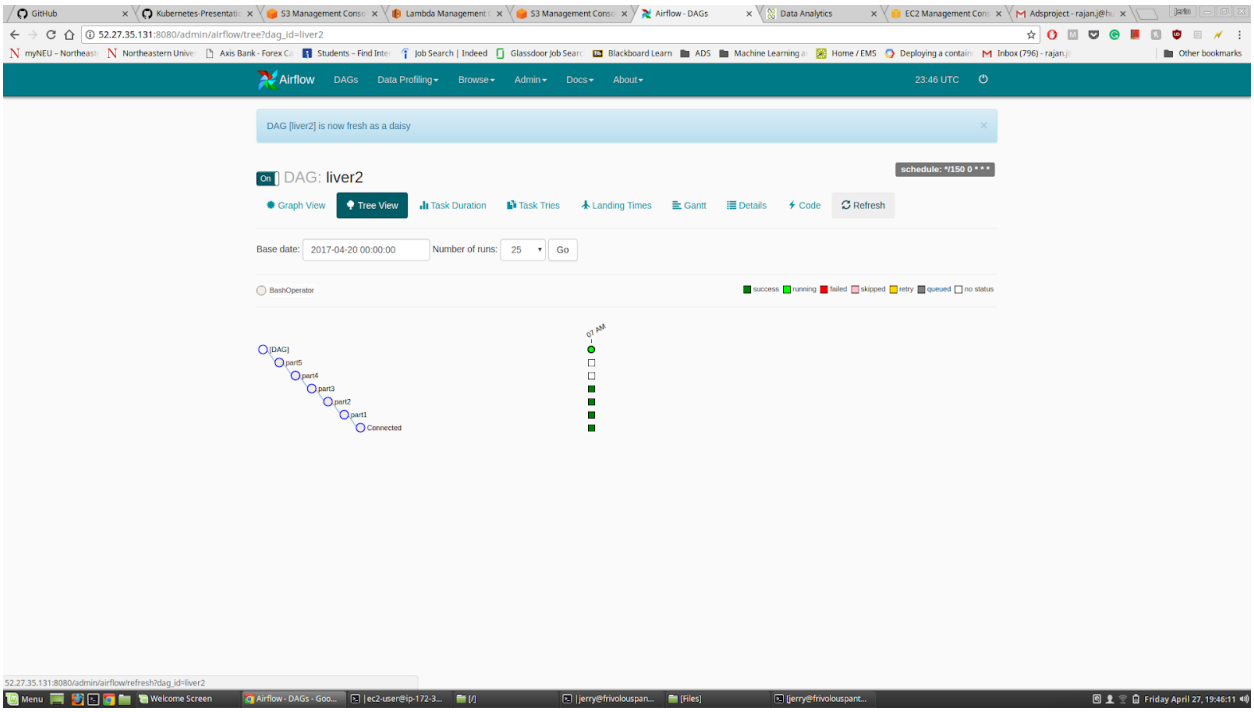

Airflow
DAGs
Data Profiling
Browse
Admin
Docs
About
23:35 UTC
DAG [kidney] is now fresh as a daisy
On
DAG: kidney
schedule: */150 0 * * *
Graph View
Tree View
Task Duration
Task Tries
Landing Times
Gantt
Details
Code
Refresh
DAG details
scheduled 2
success 4
schedule_interval
*/150 0 * * *
max_active_runs
0 / 16
concurrency
16
default_args
{'retries': 1, 'email_on_failure': False, 'email_on_retry': False, 'retry_delay': datetime.timedelta(0, 60), 'owner': 'jerin.rajan', 'depends_on_past': False, 'start_date': datetime.datetime(2018, 4, 25, 0, 0), 'email': ['jerinrajan23@gmail.com']}
tasks count
6
task ids
['part1', 'part2', 'part3', 'Connected', 'part5', 'part4']
filepath
kidney.py
owner
jerin.rajan

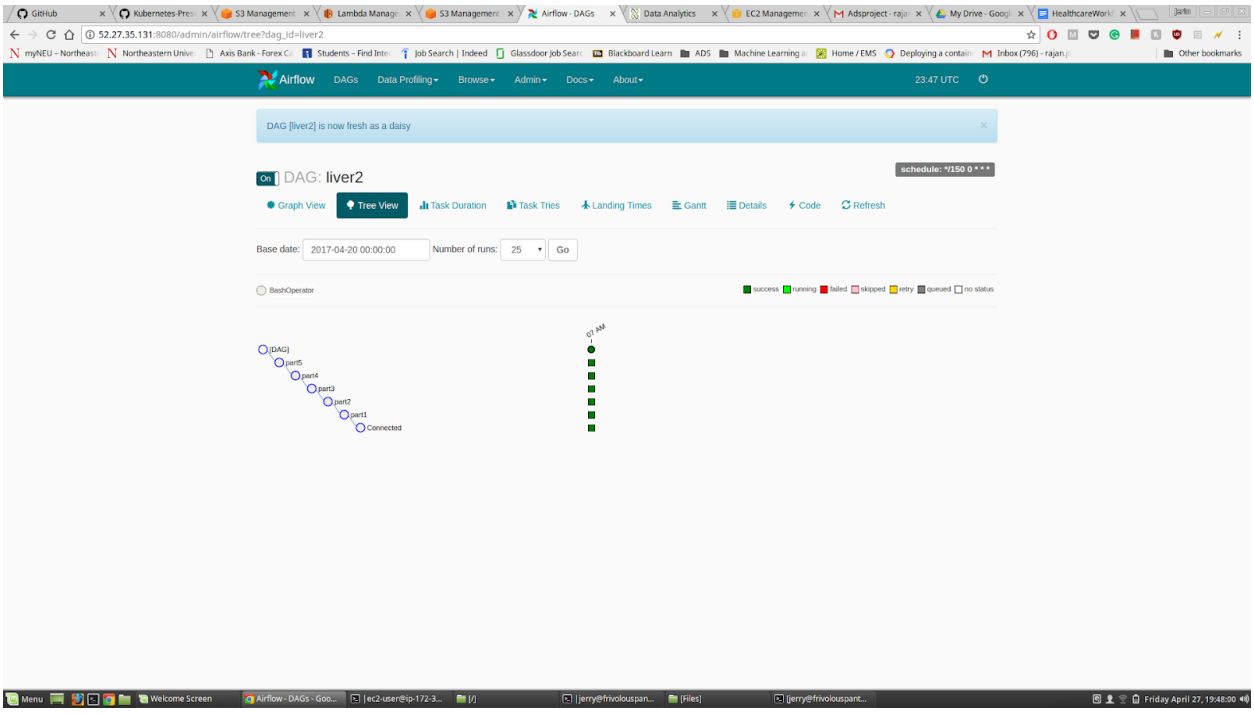


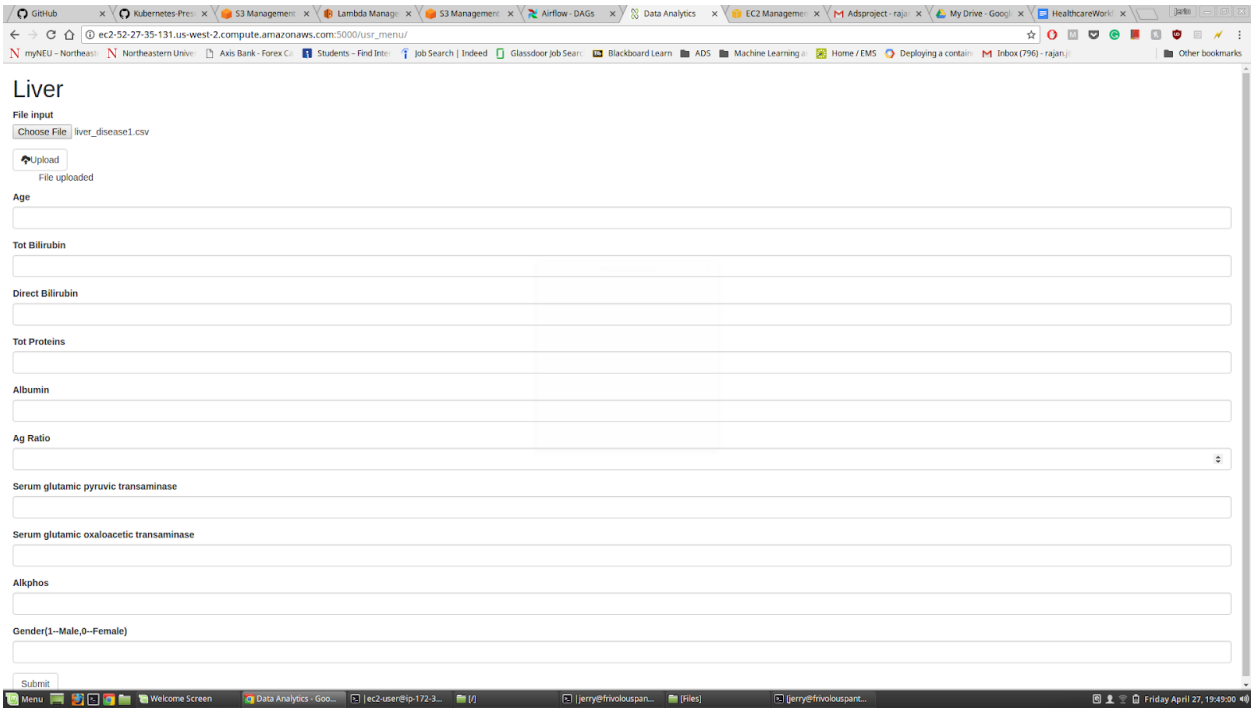


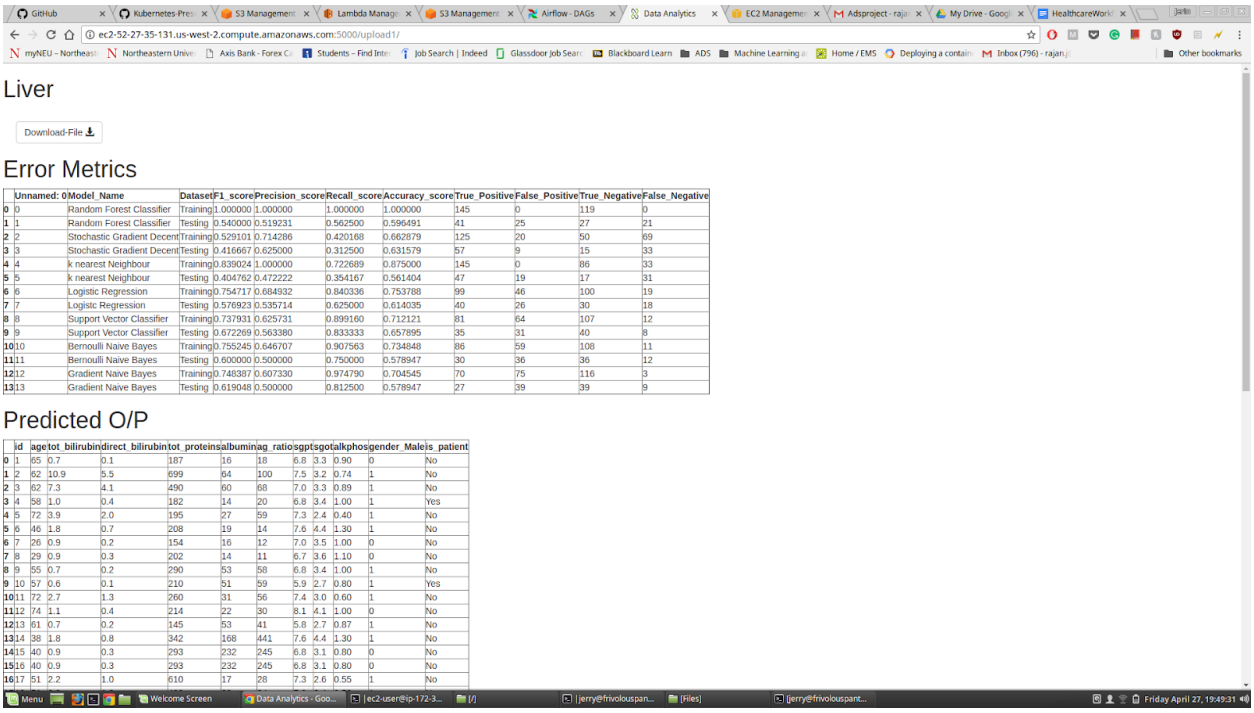
	Unnamed: 0	Model_Name	Dataset	F1_score	Precision_score	Recall_score	Accuracy_score	True_Positive	False_Positive	True_Negative	False_Negative
0	0	Random Forest Classifier	Training	1.000000	1.00000	1.000000	1.000000	42	0	32	0
1	1	Random Forest Classifier	Testing	0.969697	1.00000	0.941176	0.968750	15	0	16	1
2	2	Stochastic Gradient Decent	Training	1.000000	1.00000	1.000000	1.000000	42	0	32	0
3	3	Stochastic Gradient Decent	Testing	1.000000	1.00000	1.000000	1.000000	15	0	17	0
4	4	k nearest Neighbour	Training	1.000000	1.00000	1.000000	1.000000	42	0	32	0
5	5	k nearest Neighbour	Testing	1.000000	1.00000	1.000000	1.000000	15	0	17	0
6	6	Logistic Regression	Training	1.000000	1.00000	1.000000	1.000000	42	0	32	0
7	7	Logistc Regression	Testing	1.000000	1.00000	1.000000	1.000000	15	0	17	0
8	8	Support Vector Classifier	Training	0.950820	1.00000	0.906250	0.959459	42	0	29	3
9	9	Support Vector Classifier	Testing	1.000000	1.00000	1.000000	1.000000	15	0	17	0
10	10	Bernoulli Naive Bayes	Training	1.000000	1.00000	1.000000	1.000000	42	0	32	0
11	11	Bernoulli Naive Bayes	Testing	0.937500	1.00000	0.882353	0.937500	15	0	15	2
12	12	Gradient Naive Bayes	Training	0.984127	1.00000	0.968750	0.986486	42	0	31	1
13	13	Gradient Naive Bayes	Testing	0.693878	0.53125	1.000000	0.531250	0	15	17	0

	id	age	bp	sg	al	su	bgr	bu	sc	sod	pot	hemo	pcv	wc	rc	RBC_normal	PC_normal	PCC_present	BA_present	HTN_yes	DM_yes	CAD_yes	Appet_good	PE_yes	Ane_yes	Classification_ckd
0	0	48.0	80.0	1.020	1.0	0.0	121.0	36.0	1.200000	137.0	4.26988	15.400000	44.0	7800	5.200000	1	1	0	0	1	1	0	1	0	0	nckd
1	1	7.0	50.0	1.020	4.0	0.0	99.0	18.0	0.800000	137.0	4.26988	11.300000	38.0	6000	4.795294	1	1	0	0	0	0	0	1	0	0	ckd
2	2	62.0	80.0	1.010	2.0	3.0	423.0	53.0	1.800000	137.0	4.26988	9.600000	31.0	7500	4.795294	1	1	0	0	0	1	0	0	0	1	ckd
3	3	48.0	70.0	1.005	4.0	0.0	117.0	56.0	3.800000	111.0	2.50000	11.200000	32.0	6700	3.900000	1	0	1	0	1	0	0	0	1	1	ckd
4	4	51.0	80.0	1.010	2.0	0.0	106.0	26.0	1.400000	137.0	4.26988	11.600000	35.0	7300	4.600000	1	1	0	0	0	0	0	1	0	0	ckd
5	5	60.0	90.0	1.015	3.0	0.0	74.0	25.0	1.100000	142.0	3.20000	12.200000	39.0	7800	4.400000	1	1	0	0	1	1	0	1	1	0	ckd
6	6	68.0	70.0	1.010	0.0	0.0	100.0	54.0	24.000000	104.0	4.00000	12.400000	36.0	5800	4.795294	1	1	0	0	0	0	0	1	0	0	ckd
7	7	24.0	80.0	1.015	2.0	4.0	410.0	31.0	1.100000	137.0	4.26988	12.400000	44.0	6900	5.000000	1	0	0	0	0	1	0	1	1	0	ckd
8	8	52.0	100.0	1.015	3.0	0.0	138.0	60.0	1.900000	137.0	4.26988	10.800000	33.0	9600	4.000000	1	0	1	0	1	1	0	1	0	1	ckd
9	9	53.0	90.0	1.020	2.0	0.0	70.0	107.0	7.200000	114.0	3.70000	9.500000	29.0	12100	3.700000	0	0	1	0	1	1	0	0	0	1	ckd
10	10	50.0	60.0	1.010	2.0	4.0	490.0	55.0	4.000000	137.0	4.26988	9.400000	28.0	5800	4.795294	1	0	1	0	1	1	0	1	0	1	ckd
11	11	63.0	70.0	1.010	3.0	0.0	380.0	60.0	2.700000	131.0	4.20000	10.800000	32.0	4500	3.800000	0	0	1	0	1	1	0	0	1	0	ckd
12	12	68.0	70.0	1.015	3.0	1.0	208.0	72.0	2.100000	138.0	5.80000	9.700000	28.0	12200	3.400000	1	1	1	0	1	1	1	0	1	0	ckd
13	13	68.0	70.0	1.025	0.0	0.0	98.0	86.0	4.600000	135.0	3.40000	9.800000	41.0	5800	4.795294	1	1	0	0	1	1	1	0	1	0	ckd
14	14	68.0	80.0	1.010	3.0	2.0	157.0	90.0	4.100000	130.0	6.40000	5.600000	16.0	11000	2.600000	1	0	1	1	1	1	0	1	0	1	ckd
15	15	40.0	80.0	1.015	3.0	0.0	76.0	162.0	9.600000	141.0	4.90000	7.600000	24.0	3800	2.800000	1	1	0	0	1	0	0	1	0	1	ckd
16	16	47.0	70.0	1.015	2.0	0.0	99.0	46.0	2.200000	138.0	4.10000	12.600000	41.0	5800	4.795294	1	1	0	0	0	0	0	1	0	0	ckd


52.27.35.131:8080/admin/airflow/tree?dag_id=liver2
Airflow
DAGs
Data Profiling
Browse
Admin
Docs
About
23:46 UTC
DAG [liver2] is now fresh as a daisy
On
DAG: liver2
schedule: */150 0 * * *
Graph View
Tree View
Task Duration
Task Tries
Landing Times
Gantt
Details
Code
Refresh
Base date: 2017-04-20 00:00:00
Number of runs: 25
Go
BashOperator
success
running
failed
skipped
retry
queued
no status
07 AM
[DAG]
part5
part4
part3
part2
part1
Connected
52.27.35.131:8080/admin/airflow/refresh?dag_id=liver2
Friday April 27, 19:46:11

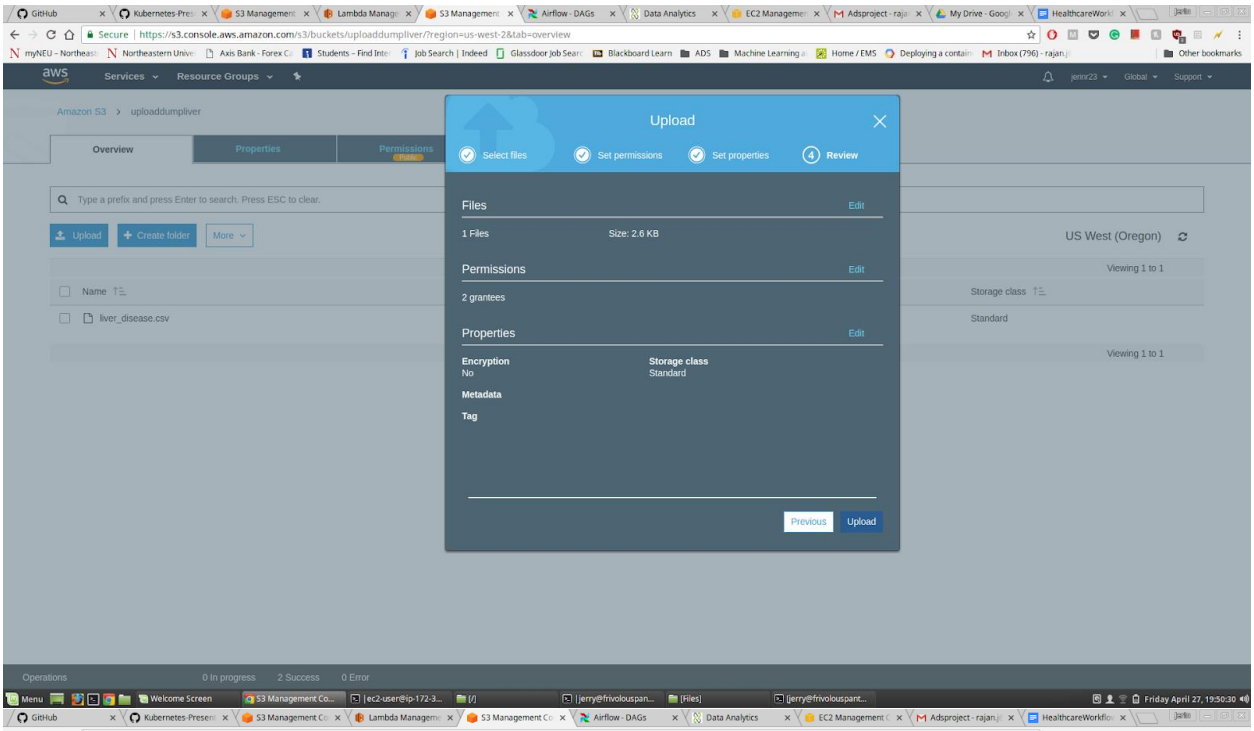

52.27.35.131:8080/admin/airflow/tree?dag_id=liver2
Airflow
DAGs
Data Profiling
Browse
Admin
Docs
About
23:47 UTC
DAG [liver2] is now fresh as a daisy
On
DAG: liver2
schedule: */150 0 * * *
Graph View
Tree View
Task Duration
Task Tries
Landing Times
Gantt
Details
Code
Refresh
Base date: 2017-04-20 00:00:00
Number of runs: 25
Go
BashOperator
success
running
failed
skipped
retry
queued
no status
07 AM
[DAG]
part5
part4
part3
part2
part1
Connected
Friday April 27, 19:48:00

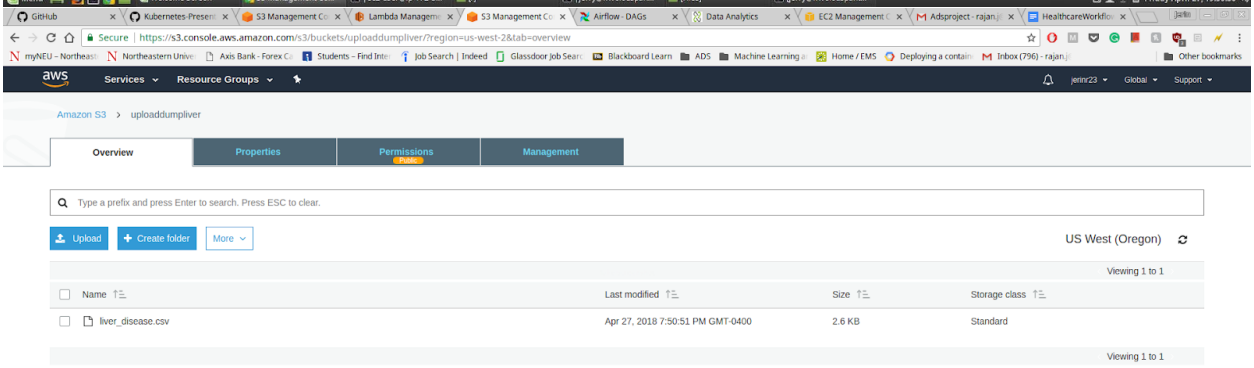


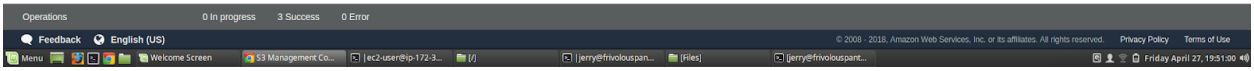


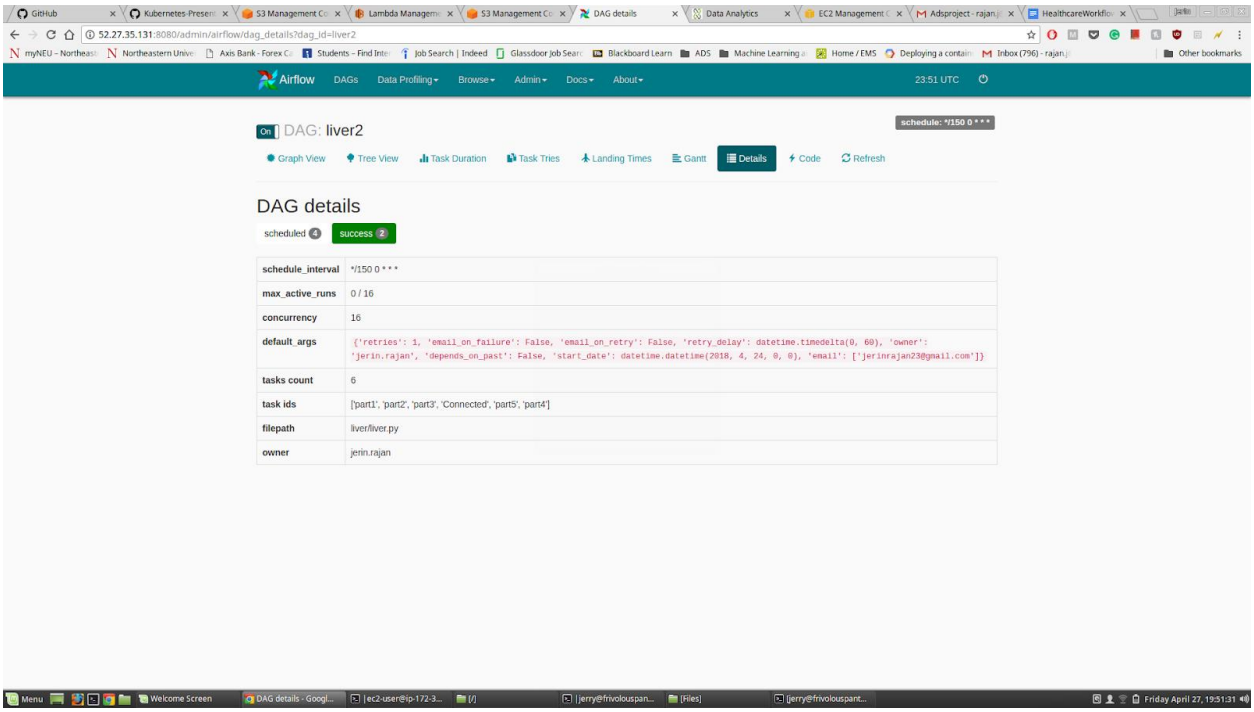
	Unnamed: 0	Model_Name	Dataset	F1_score	Precision_score	Recall_score	Accuracy_score	True_Positive	False_Positive	True_Negative	False_Negative
0	0	Random Forest Classifier	Training	1.000000	1.000000	1.000000	1.000000	145	0	119	0
1	1	Random Forest Classifier	Testing	0.540000	0.519231	0.562500	0.596491	41	25	27	21
2	2	Stochastic Gradient Decent	Training	0.529101	0.714286	0.420168	0.662879	125	20	50	69
3	3	Stochastic Gradient Decent	Testing	0.416667	0.625000	0.312500	0.631579	57	9	15	33
4	4	k nearest Neighbour	Training	0.839024	1.000000	0.722689	0.875000	145	0	86	33
5	5	k nearest Neighbour	Testing	0.404762	0.472222	0.354167	0.561404	47	19	17	31
6	6	Logistic Regression	Training	0.754717	0.684932	0.840336	0.753788	99	46	100	19
7	7	Logistic Regression	Testing	0.576923	0.535714	0.625000	0.614035	40	26	30	18
8	8	Support Vector Classifier	Training	0.737931	0.625731	0.899160	0.712121	81	64	107	12
9	9	Support Vector Classifier	Testing	0.672269	0.563380	0.833333	0.657895	35	31	40	8
10	10	Bernoulli Naive Bayes	Training	0.755245	0.646707	0.907563	0.734848	86	59	108	11
11	11	Bernoulli Naive Bayes	Testing	0.600000	0.500000	0.750000	0.578947	30	36	36	12
12	12	Gradient Naive Bayes	Training	0.748387	0.607330	0.974790	0.704545	70	75	116	3
13	13	Gradient Naive Bayes	Testing	0.619048	0.500000	0.812500	0.578947	27	39	39	9

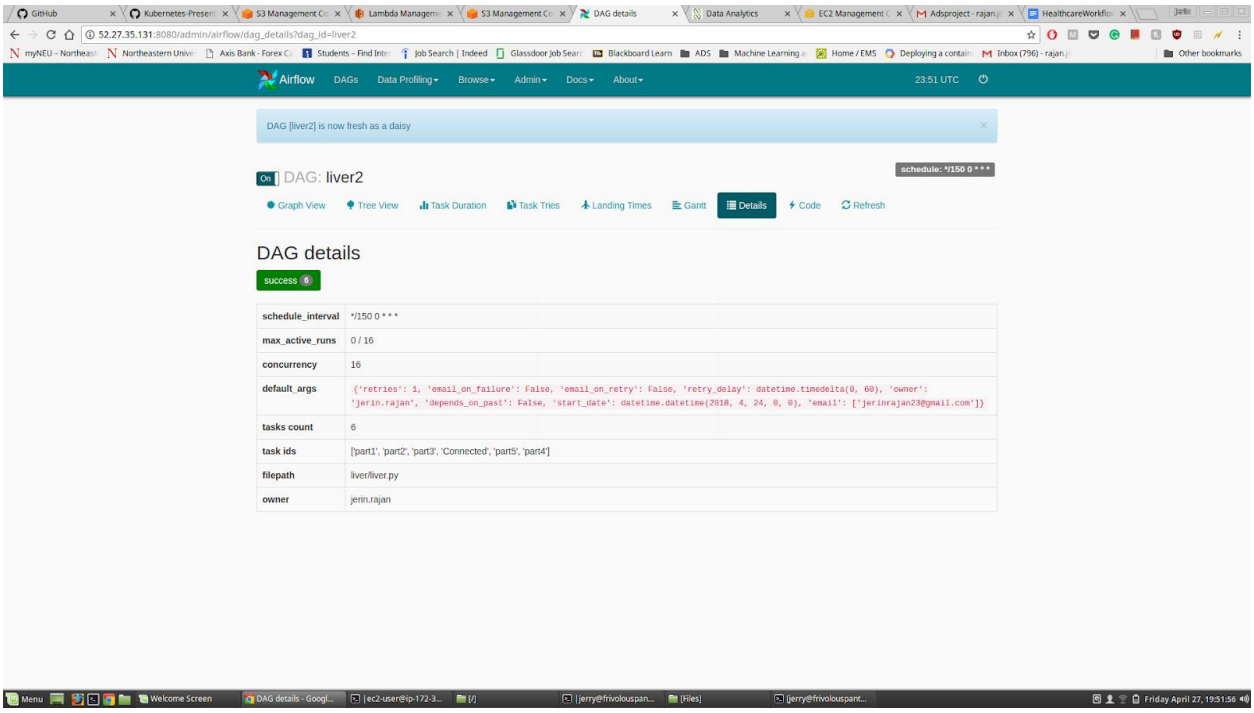
	id	age	tot_bilirubin	direct_bilirubin	tot_proteins	albumin	ag_ratio	sgpt	sgot	alkphos	gender_Male	is_patient
0	1	65	0.7	0.1	187	16	18	6.8	3.3	0.90	0	No
1	2	62	10.9	5.5	699	64	100	7.5	3.2	0.74	1	No
2	3	62	7.3	4.1	490	60	68	7.0	3.3	0.89	1	No
3	4	58	1.0	0.4	182	14	20	6.8	3.4	1.00	1	Yes
4	5	72	3.9	2.0	195	27	59	7.3	2.4	0.40	1	No
5	6	46	1.8	0.7	208	19	14	7.6	4.4	1.30	1	No
6	7	26	0.9	0.2	154	16	12	7.0	3.5	1.00	0	No
7	8	29	0.9	0.3	202	14	11	6.7	3.6	1.10	0	No
8	9	55	0.7	0.2	290	53	58	6.8	3.4	1.00	1	No
9	10	57	0.6	0.1	210	51	59	5.9	2.7	0.80	1	Yes
10	11	72	2.7	1.3	260	31	56	7.4	3.0	0.60	1	No
11	12	74	1.1	0.4	214	22	30	8.1	4.1	1.00	0	No
12	13	61	0.7	0.2	145	53	41	5.8	2.7	0.87	1	No
13	14	38	1.8	0.8	342	168	441	7.6	4.4	1.30	1	No
14	15	40	0.9	0.3	293	232	245	6.8	3.1	0.80	0	No
15	16	40	0.9	0.3	293	232	245	6.8	3.1	0.80	0	No
16	17	51	2.2	1.0	610	17	28	7.3	2.6	0.55	1	No

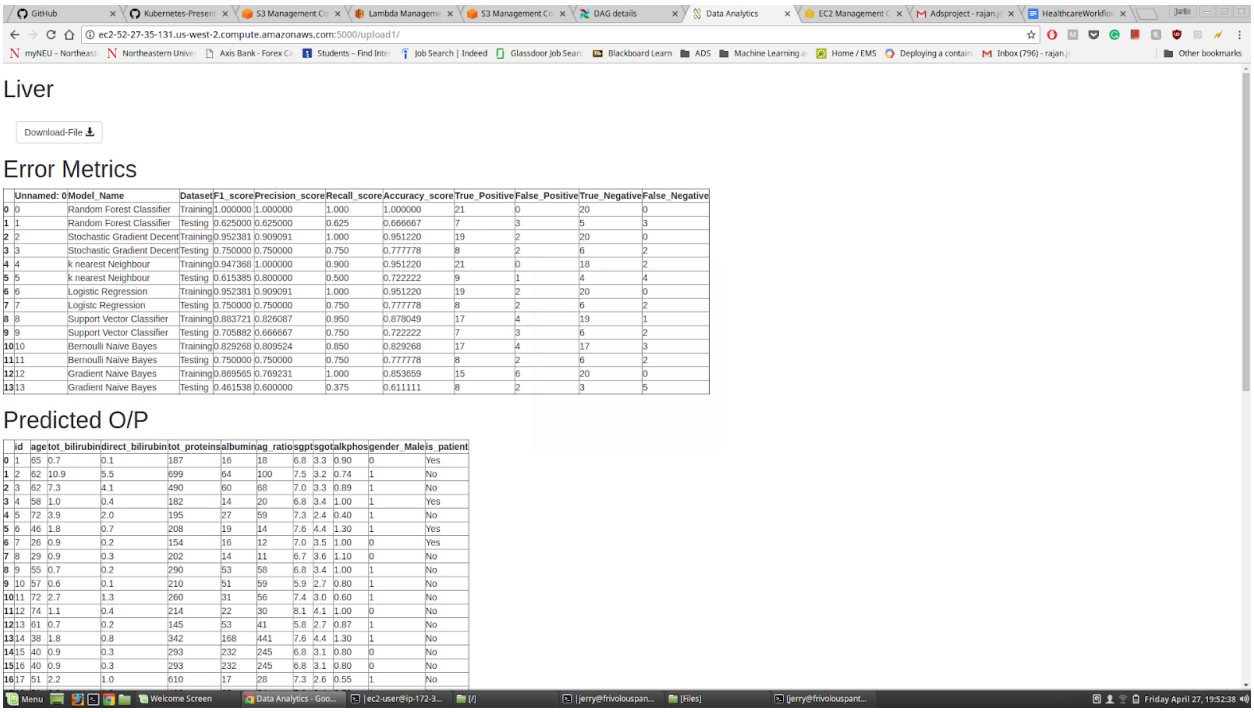

Upload
Select files
Set permissions
Set properties
4 Review
Files
Edit
1 Files
Size: 2.6 KB
Permissions
Edit
2 grantees
Properties
Edit
Encryption
No
Storage class
Standard
Metadata
Tag
Previous
Upload
Amazon S3 > uploaddumpliver
Overview
Properties
Permissions
Type a prefix and press Enter to search. Press ESC to clear.
Upload
Create folder
More
US West (Oregon)
Viewing 1 to 1
Name
Storage class
liver_disease.csv
Standard
Operations
0 In progress
2 Success
0 Error


Amazon S3 > uploaddumpliver
Overview
Properties
Permissions
Management
Type a prefix and press Enter to search. Press ESC to clear.
Upload
Create folder
More
US West (Oregon)
Viewing 1 to 1
Name
Last modified
Size
Storage class
liver_disease.csv
Apr 27, 2018 7:50:51 PM GMT-0400
2.6 KB
Standard
Viewing 1 to 1


Operations
0 In progress
3 Success
0 Error
Feedback
English (US)
© 2008 - 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy
Terms of Use


52.27.35.131:8080/admin/airflow/dag_details?dag_id=liver2
Airflow
DAGs
Data Profiling
Browse
Admin
Docs
About
23:51 UTC
On
DAG: liver2
schedule: */150 0 * * *
Graph View
Tree View
Task Duration
Task Tries
Landing Times
Gantt
Details
Code
Refresh
DAG details
scheduled 4
success 2
schedule_interval
*/150 0 * * *
max_active_runs
0 / 16
concurrency
16
default_args
{'retries': 1, 'email_on_failure': False, 'email_on_retry': False, 'retry_delay': datetime.timedelta(0, 60), 'owner': 'jerin.rajan', 'depends_on_past': False, 'start_date': datetime.datetime(2018, 4, 24, 0, 0), 'email': ['jerinrajan23@gmail.com']}
tasks count
6
task ids
['part1', 'part2', 'part3', 'Connected', 'part5', 'part4']
filepath
liver/liver.py
owner
jerin.rajan

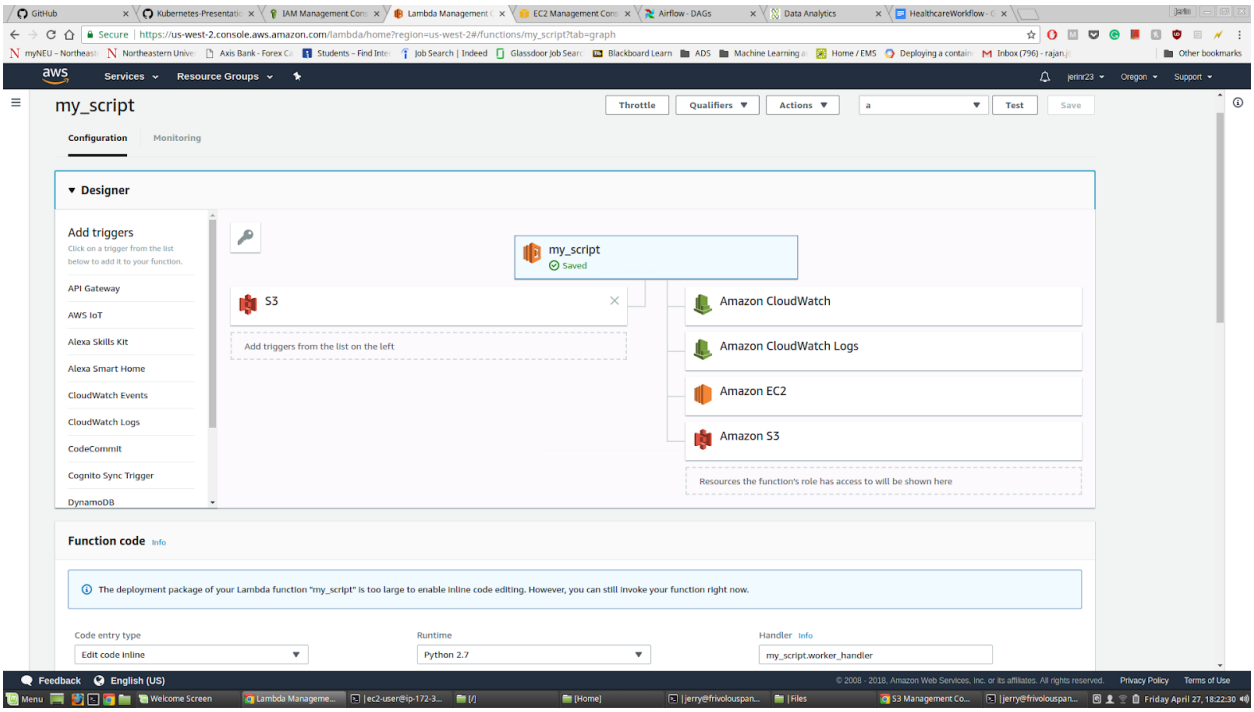

52.27.35.131:8080/admin/airflow/dag_details?dag_id=liver2
Airflow
DAGs
Data Profiling
Browse
Admin
Docs
About
23:51 UTC
DAG [liver2] is now fresh as a daisy
On
DAG: liver2
schedule: */150 0 * * *
Graph View
Tree View
Task Duration
Task Tries
Landing Times
Gantt
Details
Code
Refresh
DAG details
success 6
schedule_interval
*/150 0 * * *
max_active_runs
0 / 16
concurrency
16
default_args
{'retries': 1, 'email_on_failure': False, 'email_on_retry': False, 'retry_delay': datetime.timedelta(0, 60), 'owner': 'jerin.rajan', 'depends_on_past': False, 'start_date': datetime.datetime(2018, 4, 24, 0, 0), 'email': ['jerinrajan23@gmail.com']}
tasks count
6
task ids
['part1', 'part2', 'part3', 'Connected', 'part5', 'part4']
filepath
liver/liver.py
owner
jerin.rajan



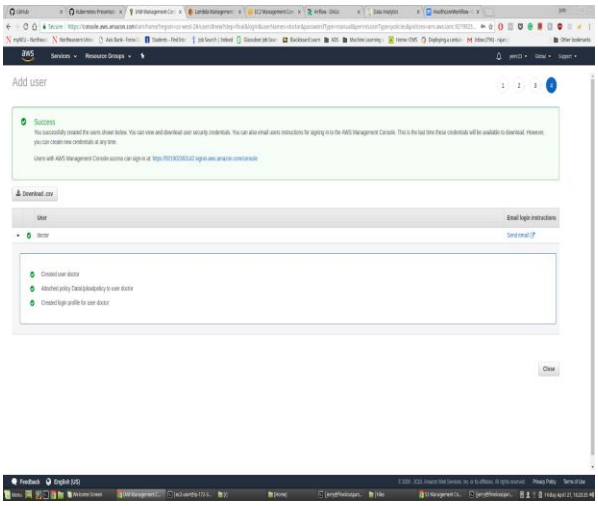
	Unnamed: 0	Model_Name	Dataset	F1_score	Precision_score	Recall_score	Accuracy_score	True_Positive	False_Positive	True_Negative	False_Negative
0	0	Random Forest Classifier	Training	1.000000	1.000000	1.000	1.000000	21	0	20	0
1	1	Random Forest Classifier	Testing	0.625000	0.625000	0.625	0.666667	7	3	5	3
2	2	Stochastic Gradient Decent	Training	0.952381	0.909091	1.000	0.951220	19	2	20	0
3	3	Stochastic Gradient Decent	Testing	0.750000	0.750000	0.750	0.777778	8	2	6	2
4	4	k nearest Neighbour	Training	0.947368	1.000000	0.900	0.951220	21	0	18	2
5	5	k nearest Neighbour	Testing	0.615385	0.800000	0.500	0.722222	9	1	4	4
6	6	Logistic Regression	Training	0.952381	0.909091	1.000	0.951220	19	2	20	0
7	7	Logistc Regression	Testing	0.750000	0.750000	0.750	0.777778	8	2	6	2
8	8	Support Vector Classifier	Training	0.883721	0.826087	0.950	0.878049	17	4	19	1
9	9	Support Vector Classifier	Testing	0.705882	0.666667	0.750	0.722222	7	3	6	2
10	10	Bernoulli Naive Bayes	Training	0.829268	0.809524	0.850	0.829268	17	4	17	3
11	11	Bernoulli Naive Bayes	Testing	0.750000	0.750000	0.750	0.777778	8	2	6	2
12	12	Gradient Naive Bayes	Training	0.869565	0.769231	1.000	0.853659	15	6	20	0
13	13	Gradient Naive Bayes	Testing	0.461538	0.600000	0.375	0.611111	8	2	3	5

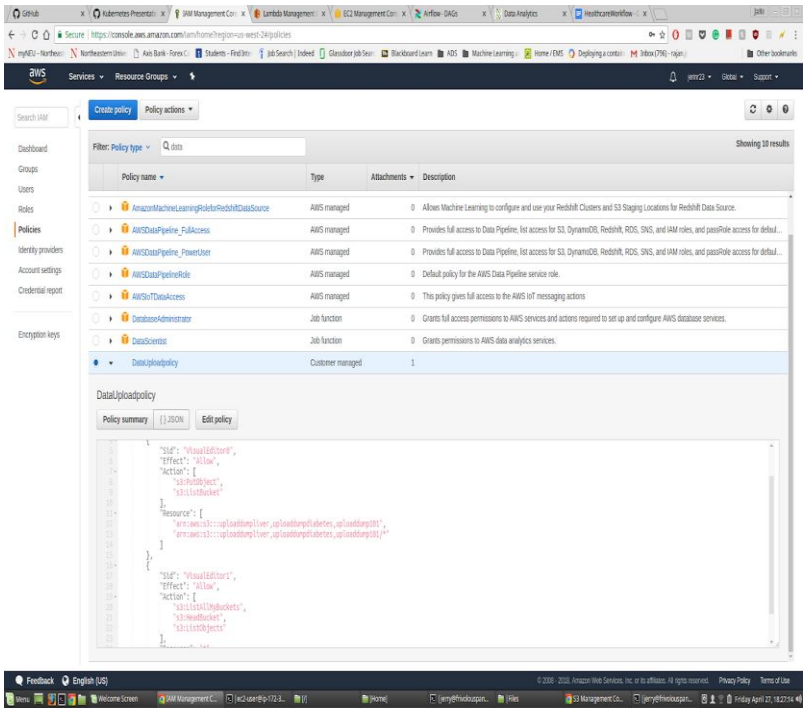
	id	age	tot_bilirubin	direct_bilirubin	tot_proteins	albumin	ag_ratio	sgpt	sgot	alkphos	gender_Male	is_patient
0	1	65	0.7	0.1	187	16	18	6.8	3.3	0.90	0	Yes
1	2	62	10.9	5.5	699	64	100	7.5	3.2	0.74	1	No
2	3	62	7.3	4.1	490	60	68	7.0	3.3	0.89	1	No
3	4	58	1.0	0.4	182	14	20	6.8	3.4	1.00	1	Yes
4	5	72	3.9	2.0	195	27	59	7.3	2.4	0.40	1	No
5	6	46	1.8	0.7	208	19	14	7.6	4.4	1.30	1	Yes
6	7	26	0.9	0.2	154	16	12	7.0	3.5	1.00	0	Yes
7	8	29	0.9	0.3	202	14	11	6.7	3.6	1.10	0	No
8	9	55	0.7	0.2	290	53	58	6.8	3.4	1.00	1	No
9	10	57	0.6	0.1	210	51	59	5.9	2.7	0.80	1	No
10	11	72	2.7	1.3	260	31	56	7.4	3.0	0.60	1	No
11	12	74	1.1	0.4	214	22	30	8.1	4.1	1.00	0	No
12	13	61	0.7	0.2	145	53	41	5.8	2.7	0.87	1	No
13	14	38	1.8	0.8	342	168	441	7.6	4.4	1.30	1	No
14	15	40	0.9	0.3	293	232	245	6.8	3.1	0.80	0	No
15	16	40	0.9	0.3	293	232	245	6.8	3.1	0.80	0	No
16	17	51	2.2	1.0	610	17	28	7.3	2.6	0.55	1	No

Lambda Designer



IAM User + Policy Created




Create policy
Policy actions
Filter: Policy type
Showing 18 results
Policy name
Type
Attachments
Description
AmazonMachineLearningRoleforRedshiftDataSource
AWS managed
Allows Machine Learning to configure and use your Redshift Clusters and S3 Staging Locations for Redshift Data Source.
AWSDataPipeline_FullAccess
AWS managed
Provides full access to Data Pipeline, list access for S3, DynamoDB, Redshift, RDS, SNS, and IAM roles, and passRole access for defaul...
AWSDataPipeline_PowerUser
AWS managed
Provides full access to Data Pipeline, list access for S3, DynamoDB, Redshift, RDS, SNS, and IAM roles, and passRole access for defaul...
AWSDataPipelineRole
AWS managed
Default policy for the AWS Data Pipeline service role.
AWSIoTDataAccess
AWS managed
This policy gives full access to the AWS IoT messaging actions
DatabaseAdministrator
Job function
Grants full access permissions to AWS services and actions required to set up and configure AWS database services.
DataScientist
Job function
Grants permissions to AWS data analytics services.
DataUploadpolicy
Customer managed
1
DataUploadpolicy
Policy summary
{} JSON
Edit policy

Future Scope

We have done for 3 dataset or it can be said that we have done for three different type of diseases. As we proceed forward, we can make this application even bigger if we get good dataset of other diseases as well.