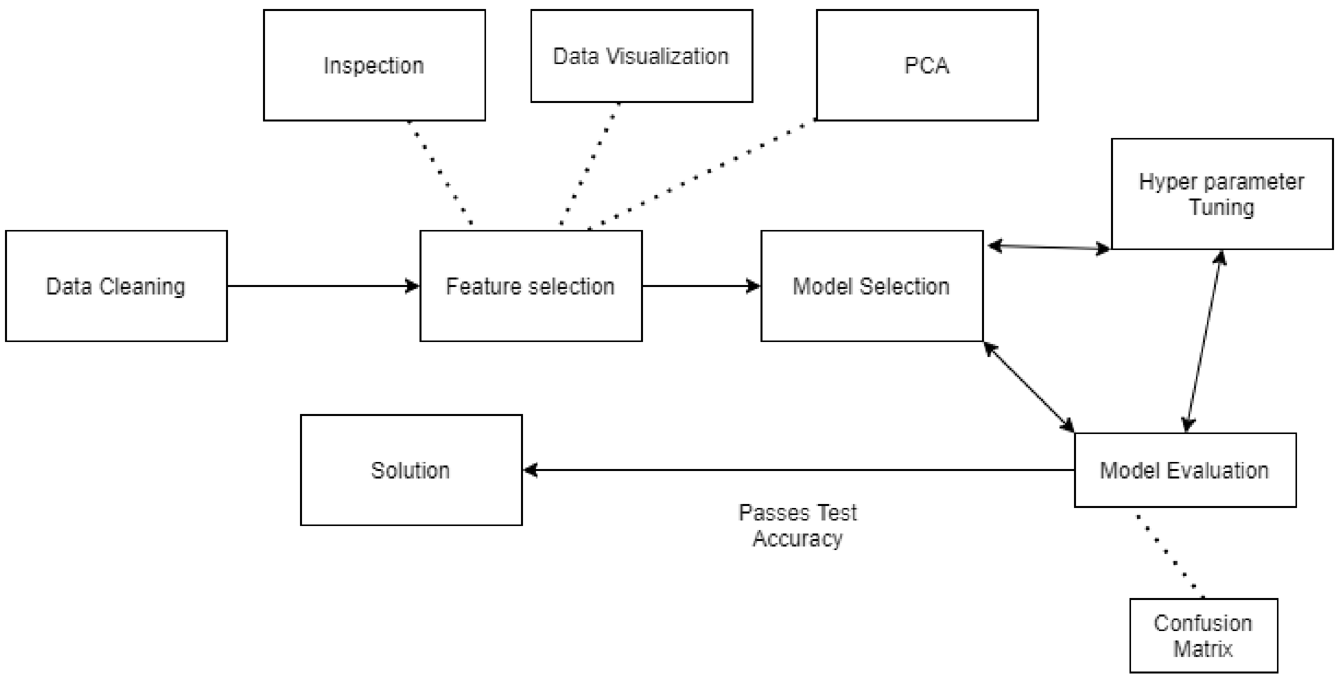
Methodology

- ###### **Data Cleaning:** Checking for null values and based on their number either droping them or replacing with mean, median, mode based on the type and description of data. Droping decscrete and catagorical variables that have highly skewed histograms.
- ###### **Data Visualization:** This step helps understand the understand the data in a visually. We can understand normality of the data as well. This helps us to decide whether to normalize the data. In case of catagorical variables it also helps in feature selection.
- ###### **Feature Selection:** Based on the Pearson correlation between the labeled column and rest of the features. In general, a very great correlation should have an absolute value greater than 0.75. When the labeled column is depended on multiple columns, the correlation with one column may be less. But combined features may have higher effect.
- ###### **Train Test Split:** We split the data into 80:20 ratio for tarining testing respectively.
- ###### **Model Selection:** Based on the data visualization and data correlation, we need to select a model that would best suit..
- ###### **Evalution:** In this case we are using F1 Score to determine the accuracy of the predicting model.



Importing important libraries

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

importing data

In [2]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount
("/content/drive", force_remount=True).
```

In [3]:

```
train = pd.read_csv("drive/My Drive/Lab 8/train.csv",names=list(range(188)))
```

```
test = pd.read_csv("drive/My Drive/Lab 8/test.csv",names=list(range(188)))
```

checking for null values

In [4]:

```
Null=[]
for i in train:
    Null.append((i,train[i].isna().mean()*100))
Null=pd.DataFrame(Null,columns=['class','per'])
Null
```

Out[4]:

	class	per
0	0	0.0
1	1	0.0
2	2	0.0
3	3	0.0
4	4	0.0
...	...	...
183	183	0.0
184	184	0.0
185	185	0.0
186	186	0.0
187	187	0.0

188 rows × 2 columns

Checking whether any class is having null values

In [5]:

```
l=Null[Null['per']!=0]['class']
Null[Null['per']!=0]
```

Out[5]:

class	per

The data is clean

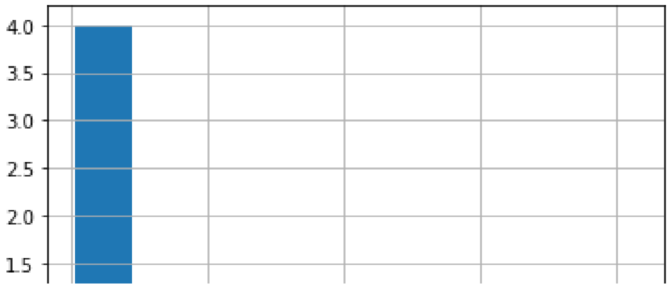
Cheching the distribution of class variable

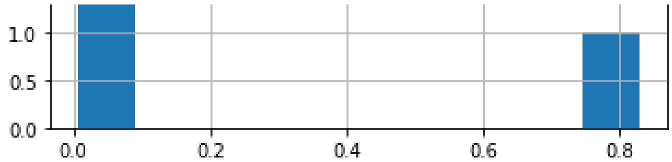
In [6]:

```
(train[187].value_counts()/len(train.values)).hist()
```

Out[6]:

<matplotlib.axes._subplots.AxesSubplot at 0x7f9cee752940>





its skewe towards only one class

Undersampling

In [7]:

```
cols = [col for col in train.columns if col not in [187]]
X = train[cols]
```

In [8]:

```
y=train[187]
from imblearn.under_sampling import RandomUnderSampler
rus = RandomUnderSampler(random_state=0)
rus.fit(X, y)
X, y = rus.fit_sample(X, y)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/externals/six.py:31: FutureWarning: The mo
dule is deprecated in version 0.21 and will be removed in version 0.23 since we've droppe
d support for Python 2.7. Please rely on the official version of six (https://pypi.org/pr
oject/six/).
  "(https://pypi.org/project/six/).", FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:144: FutureWarning: T
he sklearn.neighbors.base module is  deprecated in version 0.22 and will be removed in ve
rsion 0.24. The corresponding classes / functions should instead be imported from sklearn
.neighbors. Anything that cannot be imported from sklearn.neighbors is now part of the pr
ivate API.
  warnings.warn(message, FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Fu
nction safe_indexing is deprecated; safe_indexing is deprecated in version 0.22 and will
be removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)
```

Converting into a df

In [9]:

```
df = pd.DataFrame(data=X)
df
```

Out[9]:

	0	1	2	3	4	5	6	7	8	9	10	11
0	0.988889	0.704444	0.015556	0.000000	0.093333	0.102222	0.095556	0.086667	0.073333	0.082222	0.073333	0.080000
1	0.844059	0.601485	0.037129	0.000000	0.004950	0.009901	0.014851	0.004950	0.022277	0.019802	0.034653	0.037129
2	1.000000	0.807396	0.215716	0.224961	0.229584	0.257319	0.263482	0.255778	0.260401	0.257319	0.268105	0.263482
3	1.000000	0.880531	0.477876	0.269911	0.212389	0.247788	0.230089	0.247788	0.256637	0.283186	0.269911	0.292035
4	1.000000	0.939314	0.688654	0.248021	0.113456	0.168865	0.121372	0.065963	0.044855	0.044855	0.036939	0.068602
...	...	...	...	...	...	...	...	...	...	...	...	...
3200	0.619898	0.558673	0.530612	0.530612	0.517857	0.502551	0.451531	0.382653	0.260204	0.176020	0.079082	0.030612
3201	0.926829	0.470383	0.501742	0.501742	0.512195	0.505226	0.491289	0.435540	0.324042	0.181185	0.000000	0.006969
3202	0.907749	0.487085	0.542435	0.564576	0.583026	0.568266	0.571956	0.516605	0.376384	0.258303	0.066421	0.092251
3203	1.000000	0.515244	0.524390	0.496951	0.484756	0.460366	0.420732	0.399390	0.332317	0.280488	0.155488	0.091463
3204	1.000000	0.576271	0.596610	0.596610	0.593220	0.532203	0.569492	0.461017	0.362712	0.186441	0.000000	0.071186

3205 rows × 187 columns

Checking the frequencies of values

Noting the columns with 80% of values with same value are contained in a list

In [10]:

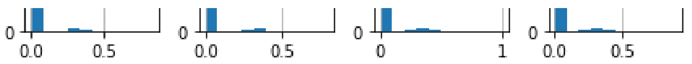
```
l=[]
for i in range(187):
  if ((df[i].value_counts()/len(df.values)).sort_values(ascending=False))[0] > 0.80:
    l.append(i)
```

displaying the columns with skewed data

In [11]:

```
fig=plt.figure(figsize=(15,15))
ax=fig.gca()
df[l].hist(ax=ax)
plt.show()
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:3: UserWarning: To output mu
ltiple subplots, the figure containing the passed axes is being cleared
  This is separate from the ipykernel package so we can avoid doing imports until
```



Removing these labels will not affect the decision making

In [12]:

```
df1=df.drop(1,axis=1)
df1
```

Out[12]:

	0	1	2	3	4	5	6	7	8	9	10	11
0	0.988889	0.704444	0.015556	0.000000	0.093333	0.102222	0.095556	0.086667	0.073333	0.082222	0.073333	0.080000
1	0.844059	0.601485	0.037129	0.000000	0.004950	0.009901	0.014851	0.004950	0.022277	0.019802	0.034653	0.037129
2	1.000000	0.807396	0.215716	0.224961	0.229584	0.257319	0.263482	0.255778	0.260401	0.257319	0.268105	0.263482
3	1.000000	0.880531	0.477876	0.269911	0.212389	0.247788	0.230089	0.247788	0.256637	0.283186	0.269911	0.292035
4	1.000000	0.939314	0.688654	0.248021	0.113456	0.168865	0.121372	0.065963	0.044855	0.044855	0.036939	0.068602
...	...	...	...	...	...	...	...	...	...	...	...	...
3200	0.619898	0.558673	0.530612	0.530612	0.517857	0.502551	0.451531	0.382653	0.260204	0.176020	0.079082	0.030612
3201	0.926829	0.470383	0.501742	0.501742	0.512195	0.505226	0.491289	0.435540	0.324042	0.181185	0.000000	0.006969
3202	0.907749	0.487085	0.542435	0.564576	0.583026	0.568266	0.571956	0.516605	0.376384	0.258303	0.066421	0.092251
3203	1.000000	0.515244	0.524390	0.496951	0.484756	0.460366	0.420732	0.399390	0.332317	0.280488	0.155488	0.091463
3204	1.000000	0.576271	0.596610	0.596610	0.593220	0.532203	0.569492	0.461017	0.362712	0.186441	0.000000	0.071186

3205 rows × 127 columns

storing data with and without these columns help us understand which data is giving better accuracy or contains valuable information. As the domain data is not present

Normalizing train data

In [13]:

```
col = list(df)
col1 = list(df1)
```

In [14]:

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
df[col] = scaler.fit_transform(df[col])
df1[col1]= scaler.fit_transform(df1[col1])
```

In [15]:

```
y1 = pd.DataFrame(data=y)
y1
```

Out[15]:

	0
0	0.0
1	0.0
2	0.0
3	0.0
4	0.0

...	.0
3200	4.0
3201	4.0
3202	4.0
3203	4.0
3204	4.0

3205 rows × 1 columns

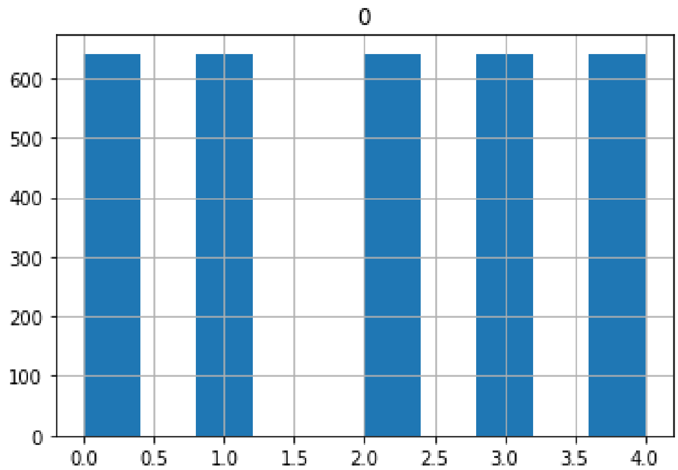
Checking the distribution of the label variable after RandomUnderSampling

In [16]:

```
y1.hist()
```

Out[16]:

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f9cde79ab00>]],
      dtype=object)
```



In [17]:

```
X_test = test[col]
y_test = test[187]
X_test1 = test[col1]
y_test1 = test[187]
X_test[col] = scaler.fit_transform(X_test[col])
X_test1[col1]= scaler.fit_transform(X_test1[col1])
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  """
/usr/local/lib/python3.6/dist-packages/pandas/core/frame.py:3072: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  self.iloc._setitem_with_indexer((slice(None), indexer), value)
/usr/local/lib/python3.6/dist-packages/pandas/core/frame.py:3037: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  self._setitem_array(key, value)
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:6: SettingWithCopyWarning:
```

```
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy

/usr/local/lib/python3.6/dist-packages/pandas/core/frame.py:3072: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  self.iloc._setitem_with_indexer((slice(None), indexer), value)
/usr/local/lib/python3.6/dist-packages/pandas/core/frame.py:3037: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  self._setitem_array(key, value)
```

Models

clf, clf1 fits complete data and fits only data with selected columns respectively.

KNN on both the data sets

In [18]:

```
from sklearn.neighbors import KNeighborsClassifier
knn =  KNeighborsClassifier(weights="distance",)
knn1 =  KNeighborsClassifier()
knn.fit(df, y1)
knn1.fit(df1, y1)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:4: DataConversionWarning: A
column-vector y was passed when a 1d array was expected. Please change the shape of y to
(n_samples, ), for example using ravel().
  after removing the cwd from sys.path.
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:5: DataConversionWarning: A
column-vector y was passed when a 1d array was expected. Please change the shape of y to
(n_samples, ), for example using ravel().
  """
```

Out[18]:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

In [19]:

```
from sklearn.metrics import f1_score
tr_error=f1_score(y1,knn.predict(df),average='weighted')
te_error=f1_score(y_test,knn.predict(X_test),average='weighted')
tr_error1=f1_score(y1,knn1.predict(df1),average='weighted')
te_error1=f1_score(y_test1,knn1.predict(X_test1),average='weighted')
error = {}
error["data"] = ["full","selected"]
error["train_F1_score"]=[tr_error,tr_error1]
error["test_F1_score"]=[te_error,te_error1]
error=pd.DataFrame(error)
error
```

Out[19]:

	data	train_F1_score	test_F1_score
0	full	1.000000	0.818056
1	selected	0.901986	0.808445

	data	train_F1_score	test_F1_score
1	selected	0.901986	0.828445

The selected data had a little higher accuracy on test data. There is overfit in the first model as the difference in acuuracy is over 19%.

SVM

In [20]:

```
from sklearn import svm
svc = svm.LinearSVC()
svc1 = svm.LinearSVC()
svc.fit(df, y1)
svc1.fit(df1, y1)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/utils/validation.py:760: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.6/dist-packages/sklearn/svm/_base.py:947: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/validation.py:760: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.6/dist-packages/sklearn/svm/_base.py:947: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
```

Out[20]:

```
LinearSVC(C=1.0, class_weight=None, dual=True, fit_intercept=True,
          intercept_scaling=1, loss='squared_hinge', max_iter=1000,
          multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
          verbose=0)
```

In [21]:

```
tr_error=f1_score(y1,svc.predict(df),average='weighted')
te_error=f1_score(y_test,svc.predict(X_test),average='weighted')
tr_error1=f1_score(y1,svc1.predict(df1),average='weighted')
te_error1=f1_score(y_test1,svc1.predict(X_test1),average='weighted')
error = {}
error["data"] = ["full","selected"]
error["train_F1_score"]=[tr_error,tr_error1]
error["test_F1_score"]=[te_error,te_error1]
error=pd.DataFrame(error)
error
```

Out[21]:

	data	train_F1_score	test_F1_score
0	full	0.801014	0.694573
1	selected	0.786734	0.689804

Simmilar results as KNN. The model which used full data had higher score that selected but there is no over fit in the in selceted one.

This shows that the plane seperating the features is not doing qa better job. so we need to move to non-linear models

Decision tree

In [22]:

```
from sklearn import tree
dt = tree.DecisionTreeClassifier()
dt1 = tree.DecisionTreeClassifier()
dt.fit(df, y1)
dt1.fit(df1, y1)
```

Out[22]:

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, 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, presort='deprecated',
                       random_state=None, splitter='best')
```

In [23]:

```
tr_error=f1_score(y1,dt.predict(df),average='weighted')
te_error=f1_score(y_test,dt.predict(X_test),average='weighted')
tr_error1=f1_score(y1,dt1.predict(df1),average='weighted')
te_error1=f1_score(y_test1,dt1.predict(X_test1),average='weighted')
error = {}
error["data"] = ["full","selected"]
error["train_F1_score"]=[tr_error,tr_error1]
error["test_F1_score"]=[te_error,te_error1]
error=pd.DataFrame(error)
error
```

Out[23]:

	data	train_F1_score	test_F1_score
0	full	1.0	0.773745
1	selected	1.0	0.776096

there is overfit in both the cases. Lets try random forest so that there may be a better result

Random Forest

In [24]:

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier()
rf1 = RandomForestClassifier()
rf.fit(df, y1)
rf1.fit(df1, y1)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:4: DataConversionWarning: A
column-vector y was passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().
  after removing the cwd from sys.path.
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:5: DataConversionWarning: A
column-vector y was passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().
  """
```

Out[24]:

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=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=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [25]:

```
tr_error=f1_score(y1,rf.predict(df),average='weighted')
```

```
te_error=f1_score(y_test,rf.predict(X_test),average='weighted')
tr_error1=f1_score(y1,rf1.predict(df1),average='weighted')
te_error1=f1_score(y_test1,rf1.predict(X_test1),average='weighted')
error = {}
error["data"] = ["full","selected"]
error["train_F1_score"]=[tr_error,tr_error1]
error["test_F1_score"]=[te_error,te_error1]
error=pd.DataFrame(error)
error
```

Out[25]:

	data	train_F1_score	test_F1_score
0	full	1.0	0.889222
1	selected	1.0	0.887856

The accuracies are same for both the

of all the models Random forest gives better results. This could due to the non-linearity nature helps understand this complex data. And as a combination of multiple trees helps to capture the patterns in multiple ways.

In [26]:

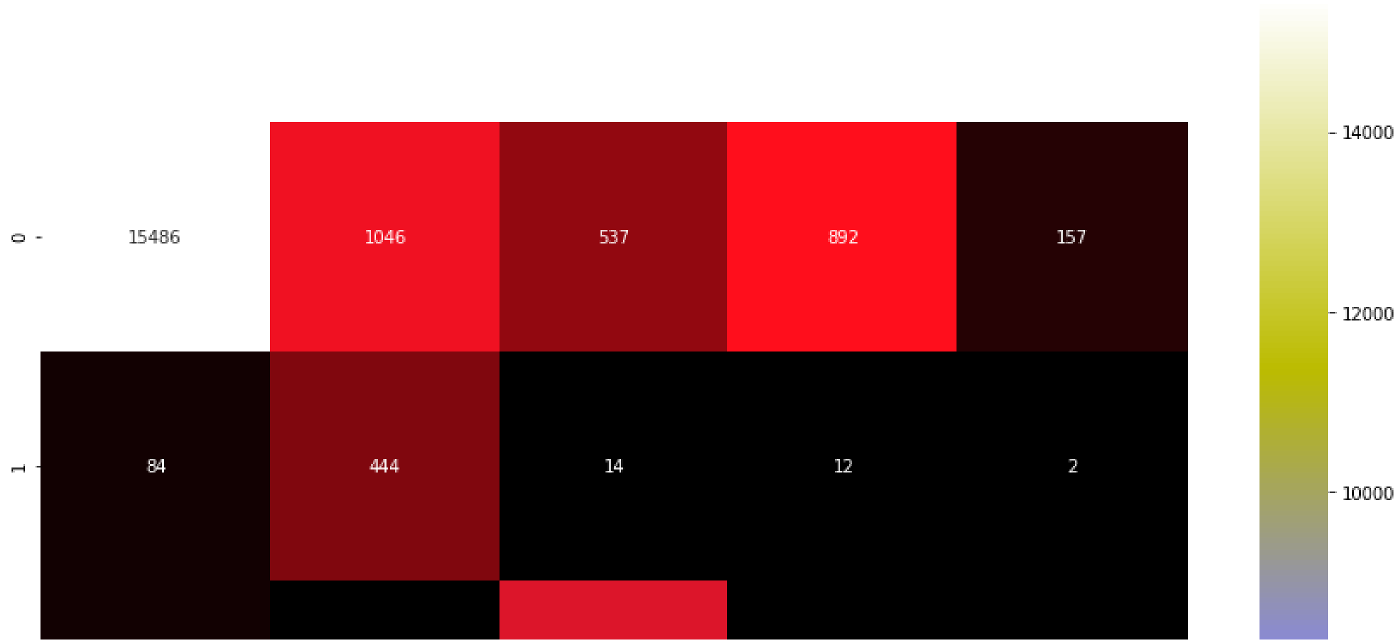
```
import numpy as np
p=rf.predict(X_test)
pred=[]
act=[]
for i in range(len(X_test)):
    act.append(y_test[i])
    pred.append(p[i])
```

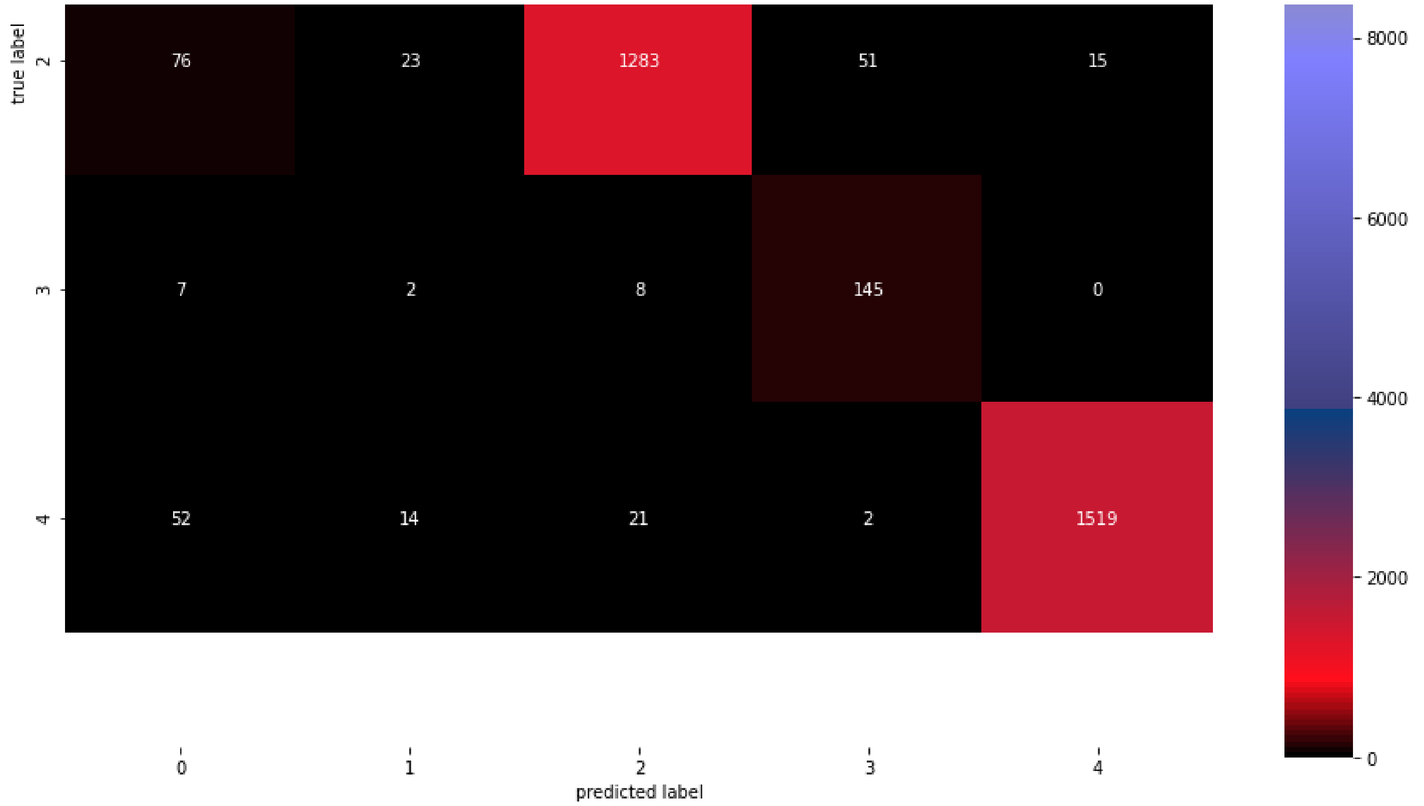
In [27]:

```
from sklearn.metrics import confusion_matrix
confm=confusion_matrix(act, pred, labels=list(range(5)))
import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(15,15))
ax=sns.heatmap(confm,annot=True,fmt='d',cbar=True,square=True,cmap="gist_stern")
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
plt.xlabel("predicted label")
plt.ylabel("true label")
```

Out[27]:

```
Text(114.0, 0.5, 'true label')
```





In [28]:

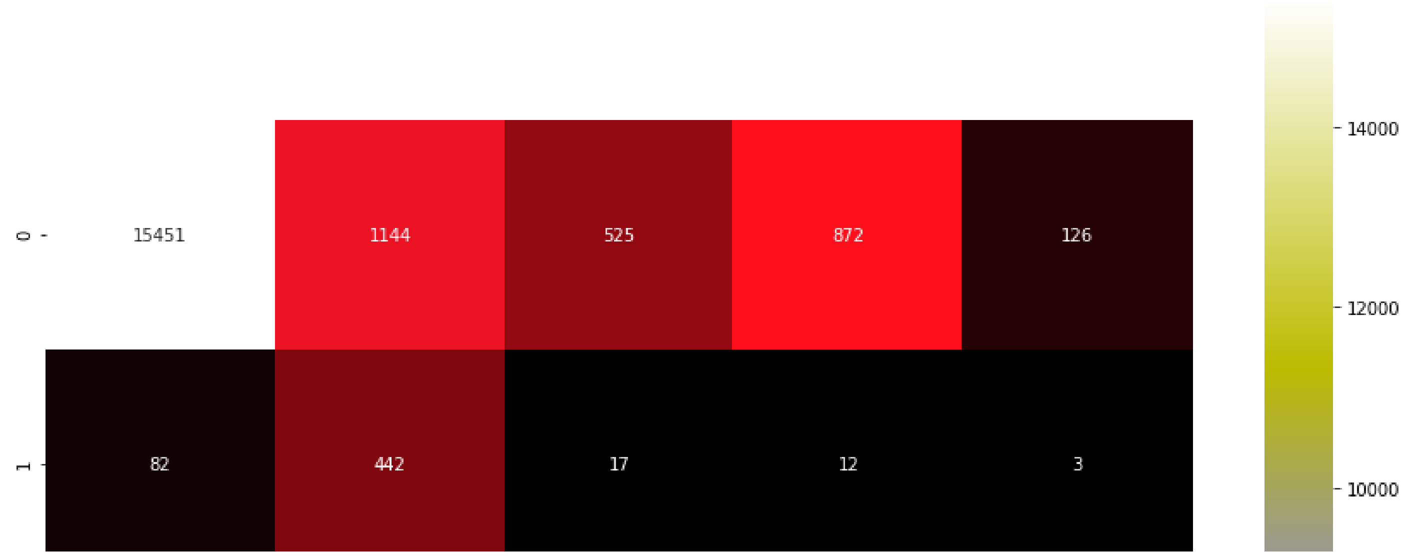
```python
import numpy as np
p=rf1.predict(X_test1)
pred=[]
act=[]
for i in range(len(X_test)):
    act.append(y_test[i])
    pred.append(p[i])
```

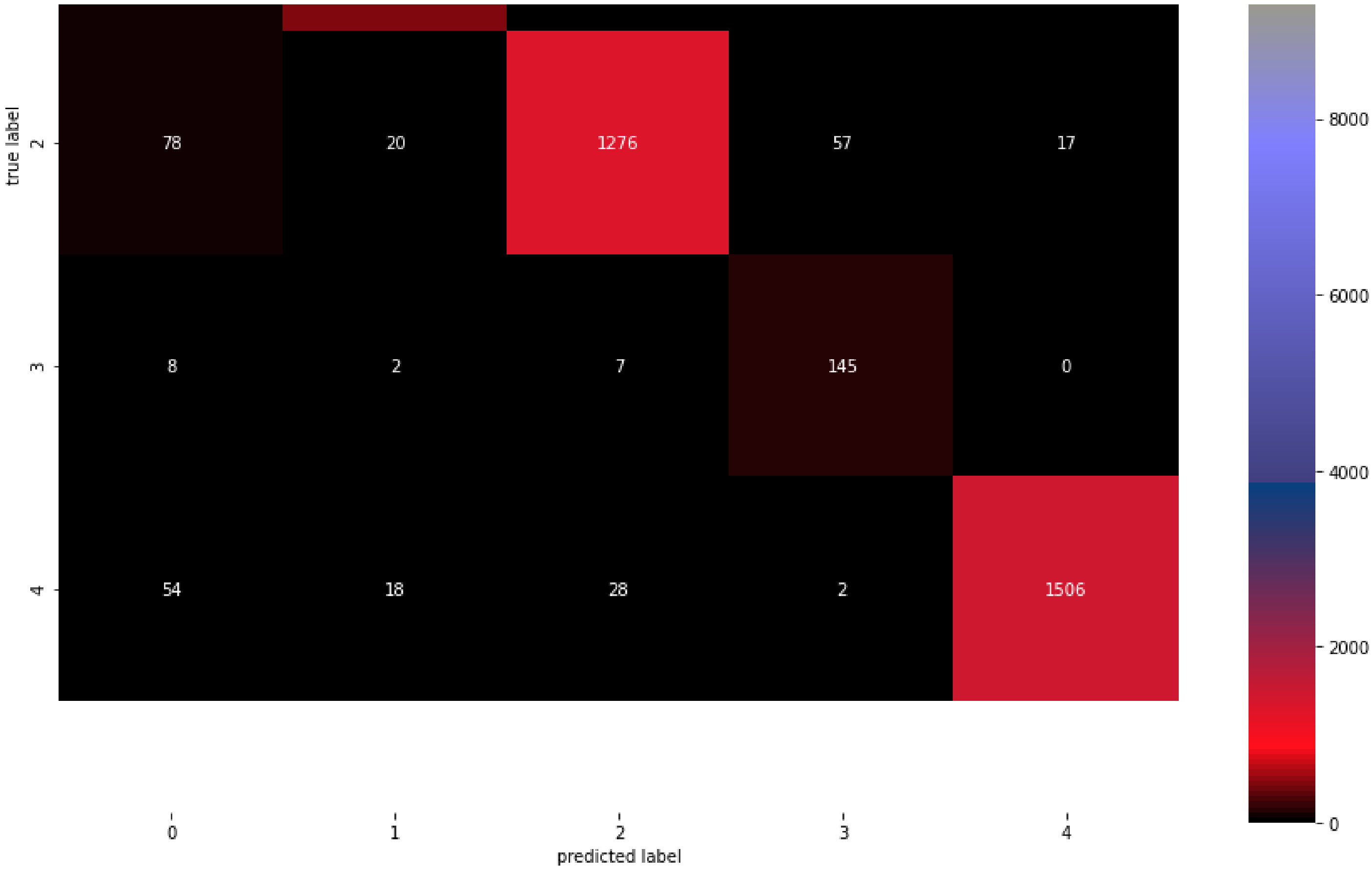
In [29]:

```python
from sklearn.metrics import confusion_matrix
confm=confusion_matrix(act, pred, labels=list(range(5)))
import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(15,15))
ax=sns.heatmap(confm,annot=True,fmt='d',cbar=True,square=True,cmap="gist_stern")
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
plt.xlabel("predicted label")
plt.ylabel("true label")
```

Out[29]:

```
Text(114.0, 0.5, 'true label')
```





In both confusion matrices the there are so many false positives in the class disease. The boundary for disease and not is mot clear. there is a chance of improvement in this area.