```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score
```

Uploading & visualissation

```
#extracting the data file from directly sorce and storing as the panda datafram
df=pd.read_excel('https://archive.ics.uci.edu/ml/machine-learning-databases/003
df_copy=df
df_copy2=df
df
```

	MouseID	DYRK1A_N	ITSN1_N	BDNF_N	NR1_N	NR2A_N	pAKT_N	pBRAF_N
0	309_1	0.503644	0.747193	0.430175	2.816329	5.990152	0.218830	0.177565
1	309_2	0.514617	0.689064	0.411770	2.789514	5.685038	0.211636	0.172817
2	309_3	0.509183	0.730247	0.418309	2.687201	5.622059	0.209011	0.175722
3	309_4	0.442107	0.617076	0.358626	2.466947	4.979503	0.222886	0.176463
4	309_5	0.434940	0.617430	0.358802	2.365785	4.718679	0.213106	0.173627
...	...	...	...	...	...	...	...	...
1075	J3295_11	0.254860	0.463591	0.254860	2.092082	2.600035	0.211736	0.171262
1076	J3295_12	0.272198	0.474163	0.251638	2.161390	2.801492	0.251274	0.182496
1077	J3295_13	0.228700	0.395179	0.234118	1.733184	2.220852	0.220665	0.161435
1078	J3295_14	0.221242	0.412894	0.243974	1.876347	2.384088	0.208897	0.173623
1079	J3295_15	0.302626	0.461059	0.256564	2.092790	2.594348	0.251001	0.191811

1080 rows × 82 columns

```
display(df.info()) #checking of the type of data in each collumn
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1080 entries, 0 to 1079
Data columns (total 82 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   MouseID          1080 non-null   object
 1   DYRK1A_N         1077 non-null   float64
 2   ITSN1_N          1077 non-null   float64
 3   BDNF_N           1077 non-null   float64
 4   NR1_N            1077 non-null   float64
 5   NR2A_N           1077 non-null   float64
 6   pAKT_N           1077 non-null   float64
 7   pBRAF_N          1077 non-null   float64
 8   pCAMKII_N        1077 non-null   float64
 9   pCREB_N          1077 non-null   float64
 10  pELK_N           1077 non-null   float64
 11  pERK_N           1077 non-null   float64
 12  pJNK_N           1077 non-null   float64
 13  PKCA_N           1077 non-null   float64
 14  pMEK_N           1077 non-null   float64
 15  pNR1_N           1077 non-null   float64
 16  pNR2A_N          1077 non-null   float64
 17  pNR2B_N          1077 non-null   float64
 18  pPKCAB_N         1077 non-null   float64
 19  pRSK_N           1077 non-null   float64
 20  AKT_N            1077 non-null   float64
 21  BRAF_N           1077 non-null   float64
 22  CAMKII_N         1077 non-null   float64
 23  CREB_N           1077 non-null   float64
 24  ELK_N            1062 non-null   float64
 25  ERK_N            1077 non-null   float64
 26  GSK3B_N          1077 non-null   float64
 27  JNK_N            1077 non-null   float64
 28  MEK_N            1073 non-null   float64
 29  TRKA_N           1077 non-null   float64
 30  RSK_N            1077 non-null   float64
 31  APP_N            1077 non-null   float64
 32  Bcatenin_N       1062 non-null   float64
 33  SOD1_N           1077 non-null   float64
 34  MTOR_N           1077 non-null   float64
 35  P38_N            1077 non-null   float64
 36  pMTOR_N          1077 non-null   float64
 37  DSCR1_N          1077 non-null   float64
 38  AMPKA_N          1077 non-null   float64
 39  NR2B_N           1077 non-null   float64
 40  pNUMB_N          1077 non-null   float64
 41  RAPTOR_N         1077 non-null   float64
 42  TIAM1_N          1077 non-null   float64
 43  pP70S6_N         1077 non-null   float64
 44  NUMB_N           1080 non-null   float64
 45  P70S6_N          1080 non-null   float64
 46  pGSK3B_N         1080 non-null   float64
```

summary of data

total 82 columns,1080rows data type avilable are float and object out of which 77 has datatype float64, and 5 has dtype object

```
#checking the column name which having the data of object type variable for eacl
i,j=0,0
f=[]
for col in df.columns.values:
  if df[col].dtype=='float64':
    i=i+1
    f.append(col)
  else:
    j=j+1
    print('data type object collumn name is : ',col)

print('no. of variable having data type float64 are',f) #printing column name ha
```

```
data type object collumn name is :  MouseID
data type object collumn name is :  Genotype
data type object collumn name is :  Treatment
data type object collumn name is :  Behavior
data type object collumn name is :  class
no. of variable having data type float64 are ['DYRK1A_N', 'ITSN1_N', 'BDNF_N'
```

data type object collumn name is : MouseID

data type object collumn name is : Genotype

data type object collumn name is : Treatment

data type object collumn name is : Behavior

data type object collumn name is : class \n

Checking of total unique and null values

```
#dropping duplicates if any
df = df.drop_duplicates()

#unique and null values
for col in df.columns.values:
  list_vals = pd.unique(df[col]) #list of unique values
  print(col + ' has ' + str(len(list_vals))  +' unique values, ' + str(df[col].:
  if len(list_vals) < 10:
    list_str = ''
    for n in range(0, len(list_vals)):
      list_str = list_str + str(list_vals[n]) + ', '
    print(' These are: '+list_str[0:len(list_str) - 2])
```

```
MouseID has 1080 unique values, 0 null entries and datatype object
DYRK1A_N has 1078 unique values, 3 null entries and datatype float64
ITSN1_N has 1077 unique values, 3 null entries and datatype float64
BDNF_N has 1078 unique values, 3 null entries and datatype float64
NR1_N has 1078 unique values, 3 null entries and datatype float64
NR2A_N has 1078 unique values, 3 null entries and datatype float64
pAKT_N has 1077 unique values, 3 null entries and datatype float64
pBRAF_N has 1076 unique values, 3 null entries and datatype float64
pCAMKII_N has 1078 unique values, 3 null entries and datatype float64
```

```
pCREB_N has 1078 unique values, 3 null entries and datatype float64
pELK_N has 1078 unique values, 3 null entries and datatype float64
pERK_N has 1078 unique values, 3 null entries and datatype float64
pJNK_N has 1077 unique values, 3 null entries and datatype float64
PKCA_N has 1078 unique values, 3 null entries and datatype float64
pMEK_N has 1078 unique values, 3 null entries and datatype float64
pNR1_N has 1078 unique values, 3 null entries and datatype float64
pNR2A_N has 1078 unique values, 3 null entries and datatype float64
pNR2B_N has 1078 unique values, 3 null entries and datatype float64
pPKCAB_N has 1078 unique values, 3 null entries and datatype float64
pRSK_N has 1078 unique values, 3 null entries and datatype float64
AKT_N has 1078 unique values, 3 null entries and datatype float64
BRAF_N has 1078 unique values, 3 null entries and datatype float64
CAMKII_N has 1078 unique values, 3 null entries and datatype float64
CREB_N has 1074 unique values, 3 null entries and datatype float64
ELK_N has 1063 unique values, 18 null entries and datatype float64
ERK_N has 1078 unique values, 3 null entries and datatype float64
GSK3B_N has 1078 unique values, 3 null entries and datatype float64
JNK_N has 1078 unique values, 3 null entries and datatype float64
MEK_N has 1073 unique values, 7 null entries and datatype float64
TRKA_N has 1076 unique values, 3 null entries and datatype float64
RSK_N has 1075 unique values, 3 null entries and datatype float64
APP_N has 1078 unique values, 3 null entries and datatype float64
Bcatenin_N has 1063 unique values, 18 null entries and datatype float64
SOD1_N has 1078 unique values, 3 null entries and datatype float64
MTOR_N has 1078 unique values, 3 null entries and datatype float64
P38_N has 1076 unique values, 3 null entries and datatype float64
pMTOR_N has 1078 unique values, 3 null entries and datatype float64
DSCR1_N has 1078 unique values, 3 null entries and datatype float64
AMPKA_N has 1076 unique values, 3 null entries and datatype float64
NR2B_N has 1078 unique values, 3 null entries and datatype float64
pNUMB_N has 1078 unique values, 3 null entries and datatype float64
RAPTOR_N has 1078 unique values, 3 null entries and datatype float64
TIAM1_N has 1076 unique values, 3 null entries and datatype float64
pP70S6_N has 1077 unique values, 3 null entries and datatype float64
NUMB_N has 1080 unique values, 0 null entries and datatype float64
P70S6_N has 1080 unique values, 0 null entries and datatype float64
pGSK3B_N has 1080 unique values, 0 null entries and datatype float64
pPKCG_N has 1080 unique values, 0 null entries and datatype float64
CDK5_N has 1080 unique values, 0 null entries and datatype float64
S6_N has 1080 unique values, 0 null entries and datatype float64
ADARB1_N has 1080 unique values, 0 null entries and datatype float64
AcetylH3K9_N has 1080 unique values, 0 null entries and datatype float64
RRP1_N has 1080 unique values, 0 null entries and datatype float64
BAX_N has 1080 unique values, 0 null entries and datatype float64
ARC_N has 1080 unique values, 0 null entries and datatype float64
ERBB4_N has 1079 unique values, 0 null entries and datatype float64
nNOS_N has 1079 unique values, 0 null entries and datatype float64
Tau_N has 1080 unique values, 0 null entries and datatype float64
```

#THE DATA HAS FEW NULL ENTRIES AROUND SEVERAL COLUMNS\n

#as the collumns having large multiple values can be considered as the continous where as the collumn having genotype object data can be mapped as the discrete variable foe the classification basis.

the collumns with object data type

Genotype is of typeobject,has 2 unique values these are:**Control,Ts65D**

Treatment is of typeobject,has 2 unique values these are:**Memantine,Salin**

Behavior is of typeobject,has 2 unique values these are:**C/S,S/C**

class is of typeobject,has 8 unique values these are:**c-CS-m,c-SC-m,c-CS-s,c-SC-s,t-CS-m,t-SC-m,t-CS-s,t-SC-s**

as we need to predict problem is to either predict the genotype (binary) or the class using the gene expression variables from DYRK1A_N to CaNA_N, we will be droppng **"Treatment"** and **"behaviour"** and **"MouseID"**

```
#dropping Treatment,Behavior, MouseID columns
df=df.drop(['MouseID','Treatment','Behavior'],axis=1)
```

#Prediction of the Genotype dropping class \n

Heatplot will only plot float it wont plot genotype

Prearing the data

```
#Prediction of the Genotype dropping class
df=df.drop(['class'],axis=1)
map_genotype={'Control':0,'Ts65Dn':1}
df= df.replace({'Genotype': map_genotype})
```

```
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer

imp = IterativeImputer(max_iter=10, random_state=0)    # Imputing the values us
imp.fit(df)

data = imp.transform(df)
data
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/impute/_iterative.py:638: Conv
  " reached.", ConvergenceWarning)
array([[0.50364388, 0.74719322, 0.4301753 , ..., 0.1281856 , 1.67565235,
        0.        ],
       [0.51461708, 0.68906355, 0.41177034, ..., 0.1311187 , 1.74360965,
        0.        ],
       [0.50918309, 0.7302468 , 0.41830878, ..., 0.12743108, 1.92642659,
        0.        ],
       ...,
       [0.22869955, 0.39517937, 0.23411809, ..., 0.35521305, 1.43082502,
        1.        ],
       [0.22124241, 0.41289438, 0.24397413, ..., 0.36535319, 1.40403123,
        1.        ],
```

```
[0.30262572, 0.46105919, 0.25656431, ..., 0.36527803, 1.37099946,
 1.        ]])
```

Checking of null values after Imputation of the variable ⇓

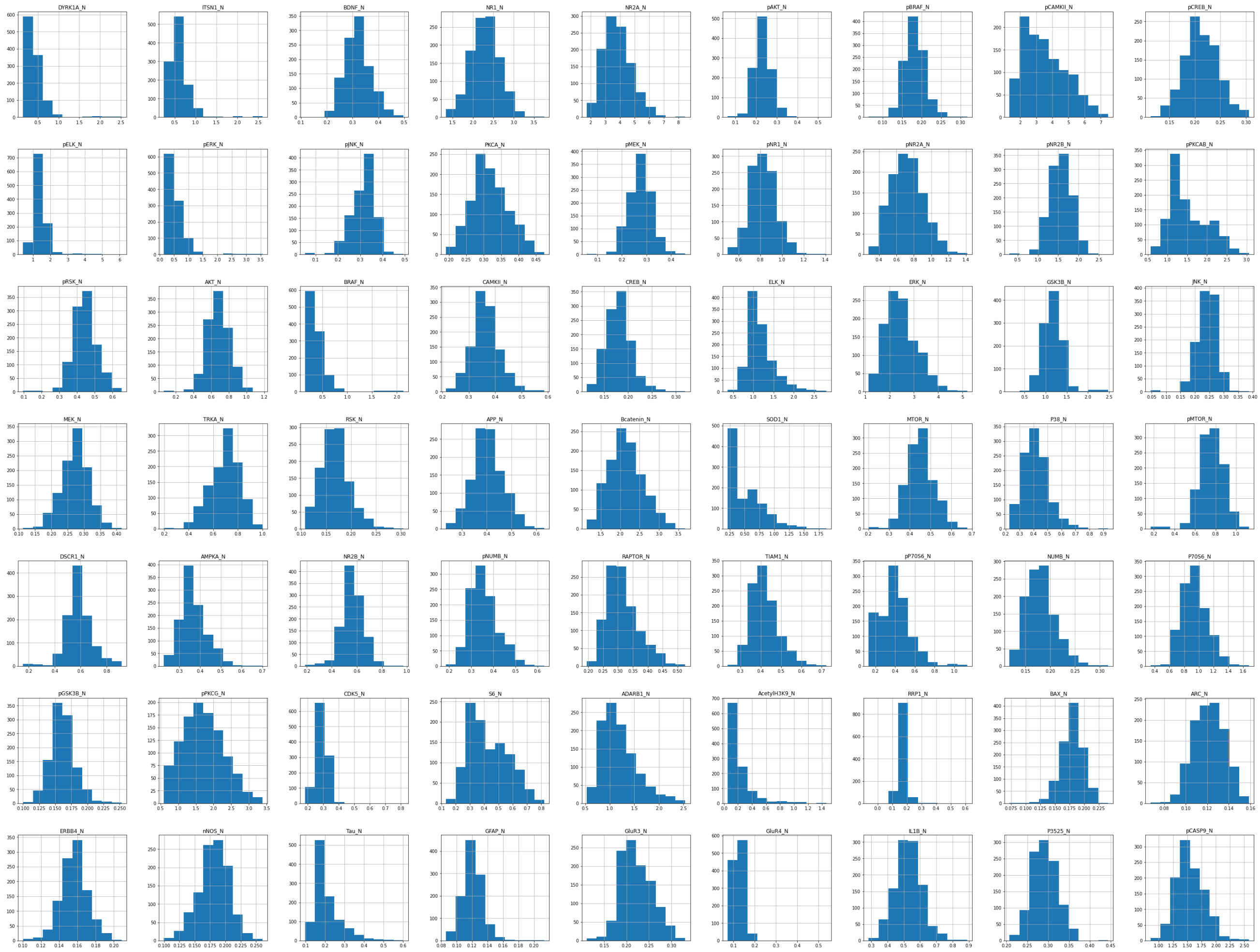
```
#forming new frame from iterated numpya array data
frame=pd.DataFrame(data,columns=df.columns) #now frame is a pdDataFrame

#again checking for null values in the new frame
for col in frame.columns.values:
  list_vals = pd.unique(frame[col]) #list of unique values
  print(col + ' has ' + str(len(list_vals))  +' unique values, ' + str(frame[co
  if len(list_vals) < 10:
    list_str = ''
    for n in range(0, len(list_vals)):
      list_str = list_str + str(list_vals[n]) + ', '
    print(' These are: '+list_str[0:len(list_str) - 2])
```

```
DYRK1A_N has 1080 unique values, 0 null entries and datatype float64
ITSN1_N has 1079 unique values, 0 null entries and datatype float64
BDNF_N has 1080 unique values, 0 null entries and datatype float64
NR1_N has 1080 unique values, 0 null entries and datatype float64
NR2A_N has 1080 unique values, 0 null entries and datatype float64
pAKT_N has 1079 unique values, 0 null entries and datatype float64
pBRAF_N has 1078 unique values, 0 null entries and datatype float64
pCAMKII_N has 1080 unique values, 0 null entries and datatype float64
pCREB_N has 1080 unique values, 0 null entries and datatype float64
pELK_N has 1080 unique values, 0 null entries and datatype float64
pERK_N has 1080 unique values, 0 null entries and datatype float64
pJNK_N has 1079 unique values, 0 null entries and datatype float64
PKCA_N has 1080 unique values, 0 null entries and datatype float64
pMEK_N has 1080 unique values, 0 null entries and datatype float64
pNR1_N has 1080 unique values, 0 null entries and datatype float64
pNR2A_N has 1080 unique values, 0 null entries and datatype float64
pNR2B_N has 1080 unique values, 0 null entries and datatype float64
pPKCAB_N has 1080 unique values, 0 null entries and datatype float64
pRSK_N has 1080 unique values, 0 null entries and datatype float64
AKT_N has 1080 unique values, 0 null entries and datatype float64
BRAF_N has 1080 unique values, 0 null entries and datatype float64
CAMKII_N has 1080 unique values, 0 null entries and datatype float64
CREB_N has 1076 unique values, 0 null entries and datatype float64
ELK_N has 1080 unique values, 0 null entries and datatype float64
ERK_N has 1080 unique values, 0 null entries and datatype float64
GSK3B_N has 1080 unique values, 0 null entries and datatype float64
JNK_N has 1080 unique values, 0 null entries and datatype float64
MEK_N has 1079 unique values, 0 null entries and datatype float64
TRKA_N has 1078 unique values, 0 null entries and datatype float64
RSK_N has 1077 unique values, 0 null entries and datatype float64
APP_N has 1080 unique values, 0 null entries and datatype float64
Bcatenin_N has 1080 unique values, 0 null entries and datatype float64
SOD1_N has 1080 unique values, 0 null entries and datatype float64
MTOR_N has 1080 unique values, 0 null entries and datatype float64
P38_N has 1078 unique values, 0 null entries and datatype float64
pMTOR_N has 1080 unique values, 0 null entries and datatype float64
```

```
DSCR1_N has 1080 unique values, 0 null entries and datatype float64
AMPKA_N has 1078 unique values, 0 null entries and datatype float64
NR2B_N has 1080 unique values, 0 null entries and datatype float64
pNUMB_N has 1080 unique values, 0 null entries and datatype float64
RAPTOR_N has 1080 unique values, 0 null entries and datatype float64
TIAM1_N has 1078 unique values, 0 null entries and datatype float64
pP70S6_N has 1079 unique values, 0 null entries and datatype float64
NUMB_N has 1080 unique values, 0 null entries and datatype float64
P70S6_N has 1080 unique values, 0 null entries and datatype float64
pGSK3B_N has 1080 unique values, 0 null entries and datatype float64
pPKCG_N has 1080 unique values, 0 null entries and datatype float64
CDK5_N has 1080 unique values, 0 null entries and datatype float64
S6_N has 1080 unique values, 0 null entries and datatype float64
ADARB1_N has 1080 unique values, 0 null entries and datatype float64
AcetylH3K9_N has 1080 unique values, 0 null entries and datatype float64
RRP1_N has 1080 unique values, 0 null entries and datatype float64
BAX_N has 1080 unique values, 0 null entries and datatype float64
ARC_N has 1080 unique values, 0 null entries and datatype float64
ERBB4_N has 1079 unique values, 0 null entries and datatype float64
nNOS_N has 1079 unique values, 0 null entries and datatype float64
Tau_N has 1080 unique values, 0 null entries and datatype float64
GFAP_N has 1079 unique values, 0 null entries and datatype float64
```

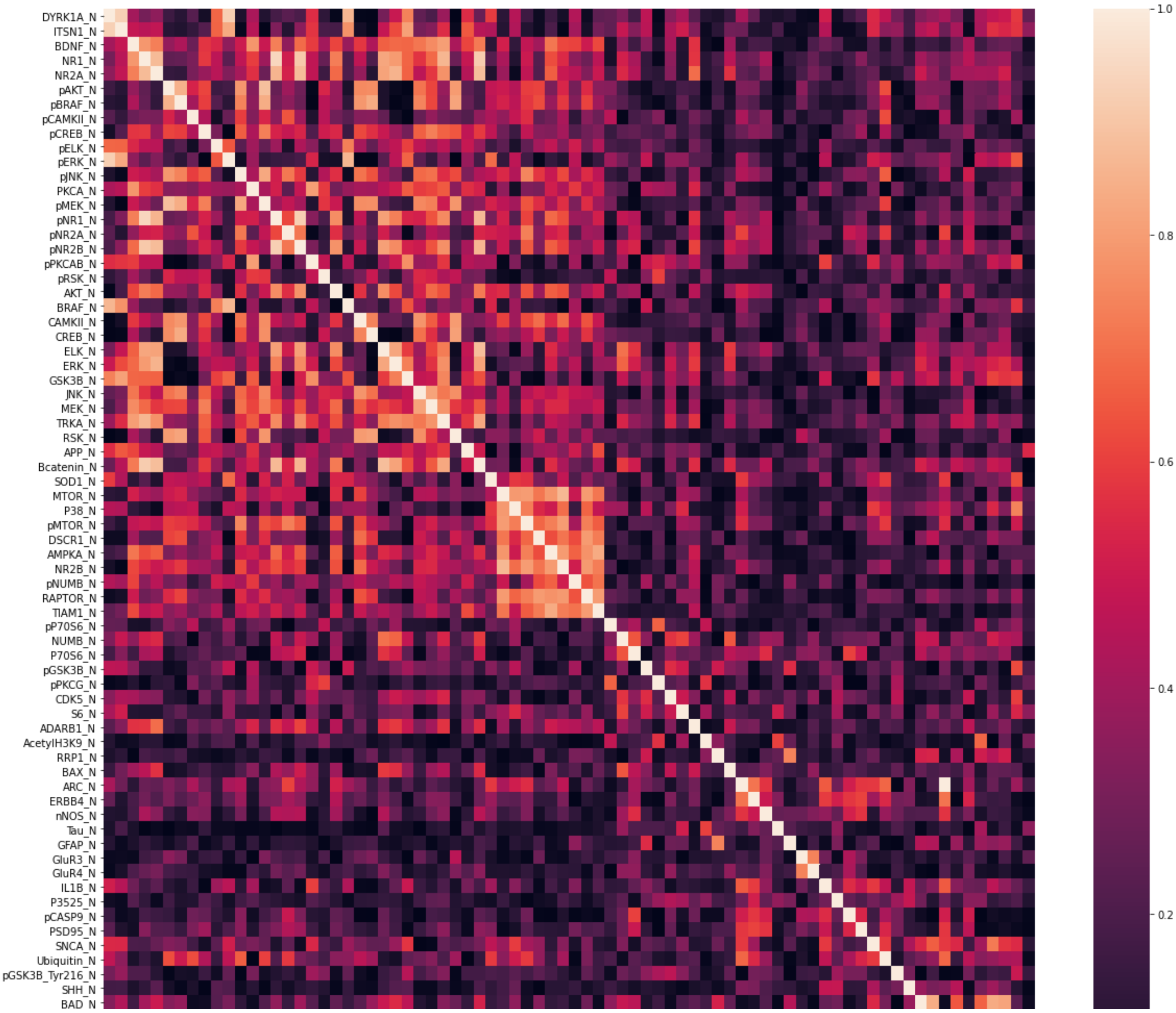
Visualisation of the data using histogram plots

```
frame.hist(bins=10, figsize=(50,50))
plt.show()
```

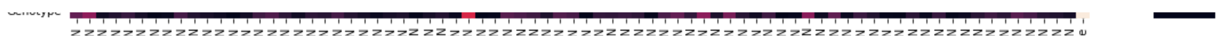



```
1 #checking spearman correlation map after plotting absolute value of the float va
2 corrmat=frame.corr(method="spearman")
3 fig,ax=plt.subplots(figsize=(20,20))
4 sns.heatmap(abs(corrmat),annot=False)
5 plt.show
```

```
<function matplotlib.pyplot.show>
```



#most of the variable are seems to be uncorrelated s visualisation by heatmap \n except DYRK1A_N, ITSN1_N having correlation index of range 0.7 to 0.85 one of these can be dropped



```
#storing of variable in the variable X and Y
X=frame.drop(['Genotype'],axis=1) #test_x
Y=frame['Genotype'] #test_y
```

```
#splitting of test data on train data and test data in 70/30 ratio
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3)
```

```
#scaling and data/Normalising data using sklearn.preproseccing.standardscaler
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.fit_transform(X_test)
```

LASSO regularized logistic regression

Training of model and Testing

```
from sklearn.linear_model import Lasso
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
from sklearn import metrics
#Cross validation search for L1 lasso model
L1_reg =Lasso()
hyperparameters = {'alpha':[0.001, 0.01, 0.1, 1, 10, 100, 1000]} # defining hyp
clf_lasso = GridSearchCV(L1_reg, hyperparameters, scoring = 'neg_mean_squared_e
clf_lasso.fit(X_train, Y_train)
print('Best hyperparameters are ', clf_lasso.best_params_)
print('the poly coefficients are',clf_lasso.best_estimator_.coef_)
```

```
Best hyperparameters are  {'alpha': 0.001}
the poly coefficients are [-0.02417414  0.36165681  0.         -0.12645418  0
  0.02651541  0.          0.05326986 -0.01440851 -0.00826299 -0.03902996
  0.04878808 -0.02436991 -0.02190462 -0.05523344  0.14775138 -0.13119197
 -0.03933637  0.05537856 -0.26920548 -0.08377507 -0.         -0.09061921
 -0.18346977 -0.00814901  0.          0.0252315   0.15301244 -0.06897449
  0.24706197 -0.03102878  0.02867998 -0.06550118 -0.03496024  0.
  0.09202686 -0.13866439 -0.0706255  -0.04525697 -0.01984101  0.11351007
  0.03778653  0.05166026  0.00187472  0.01266935 -0.00517649  0.
 -0.          0.01506778  0.          0.0179347   0.0007612  -0.03012157
  0.0853575   0.00175185  0.02922081  0.00525772 -0.10039406  0.
 -0.07695528  0.00472886  0.04566186  0.          0.02750224 -0.
 -0.         -0.03831532  0.03375193  0.         -0.00134057 -0.00753411
 -0.09136477 -0.03333992 -0.          0.02874995  0.01682856]
```

Testing for the L1 logistic classification

```
#Testing of the model
lasso_final_model=clf_lasso.best_estimator_
prediction=lasso_final_model.predict(X_test)

#Considering 0.5 as descision boundary , if prediction less than 0.5 itll be 0
for i in range(0,len(prediction)):
  if prediction[i]>= 0.5:
    prediction[i] = 1
  else:
    prediction[i] = 0
print('testing parameters for Logistic Classification')
print('\nAccuracy score:',metrics.accuracy_score(Y_test,prediction))
print('\nf1 score      :', metrics.f1_score(Y_test, prediction))
print('\nroc_auc_score : ', metrics.roc_auc_score(Y_test, prediction))
```

```
testing parameters for Logistic Classification

Accuracy score: 0.9753086419753086

f1 score      : 0.9733333333333333

roc_auc_score :  0.9747731883780576
```

#2Support Vector Classifier

```
from sklearn.svm import SVC
hyperparameters = {'kernel':('rbf','linear','poly'), 'C':[.1, 1, 5, 10], 'degre
svc=SVC()
clf_svc = GridSearchCV(svc, hyperparameters, scoring= 'f1')
clf_svc.fit(np.array(X_train), np.squeeze(Y_train))
svc_final_model=clf_svc.best_estimator_
prediction=svc_final_model.predict(X_test) #fitting model to the best data

print('best estimator parameters are :',clf_svc.best_params_)
print('testing parameters for SV Classification')
print('\nAccuracy score:',metrics.accuracy_score(Y_test,prediction))
print('\nf1 score      :', metrics.f1_score(Y_test, prediction))
print('\nroc_auc_score : ', metrics.roc_auc_score(Y_test, prediction))
```

```
best estimator parameters are : {'C': 5, 'degree': 3, 'kernel': 'rbf'}
testing parameters for SV Classification

Accuracy score: 0.9969135802469136

f1 score      : 0.9966996699669968

roc_auc_score :  0.9971098265895953
```

#3Random aforest classifier

```
from sklearn.ensemble import RandomForestClassifier
hyperparameters = {'max_depth':[2,5,10,20],'n_estimators':[10,30,100]}
scoring='f1'
rfc=RandomForestClassifier()
clf_rfc = GridSearchCV(rfc, hyperparameters, scoring=scoring)
clf_rfc.fit(np.array(X_train), np.squeeze(Y_train))
print('Best parameters are :',clf_rfc.best_params_)
y_test_true=np.squeeze(Y_test)
y_predicted=clf_rfc.best_estimator_.predict(X_test)

print('testing parameters for Random Forest Classification')
print('\nAccuracy score:',metrics.accuracy_score(y_test_true,y_predicted))
print('\n f1 score      :',metrics.f1_score(y_test_true,y_predicted))
print('\nroc_auc_score : ', metrics.roc_auc_score(y_test_true,y_predicted))
Y_train.shape
```

```
Best parameters are : {'max_depth': 20, 'n_estimators': 100}
testing parameters for Random Forest Classification

Accuracy score: 0.9814814814814815

 f1 score      : 0.9801324503311258
```

```
roc_auc_score :  0.981395704934349
(756,)
```

#

#Recursive feature elimination

It uses the model accuracy to identify which attributes (and combination of attributes) contribute the most to predicting the target attribute.

```
#Recursive feature elimination for RFC
from sklearn.feature_selection import RFECV
from sklearn.model_selection import StratifiedKFold
rfc = RandomForestClassifier(random_state=101)
rfecv = RFECV(estimator=rfc, step=1, cv=StratifiedKFold(10), scoring='accuracy'
model=rfecv.fit(X_train, Y_train)
Y_predicted=model.predict(X_test)

print('testing parameters for Recursive feature elimination with cross validati
print('\nAccuracy score:',metrics.accuracy_score(Y_test,Y_predicted))
print('\n f1 score      :',metrics.f1_score(Y_test,Y_predicted))
print('\nroc_auc_score : ', metrics.roc_auc_score(Y_test,Y_predicted))
```

```
testing parameters for Recursive feature elimination with cross validation fo

Accuracy score: 0.9814814814814815

 f1 score      : 0.98

roc_auc_score :  0.9809746200666078
```

Observation REFCV takes more time to execute than other classification method.

```
#Recursive feature elimination for SVC
#REFCV model for SVC #code curtesy stack excahange
from sklearn.feature_selection import RFECV
from sklearn.svm import SVC

# create classifier to use with recursive feature elimination
svc = SVC(kernel="linear", class_weight = 'balanced')
# run recursive feature elimination with cross-validation
rfecv = RFECV(estimator=svc, step=1, cv=3,scoring = 'roc_auc') # pick features 
newTrain = rfecv.fit(X_train, Y_train)

# test model
```

```
y_predict=newTrain.predict(X_test)
print('testing parameters for Recursive feature elimination with SVC')
print('\nAccuracy score:',metrics.accuracy_score(Y_test,y_predict))
print('\n f1 score      :',metrics.f1_score(Y_test,y_predict))
print('\nroc_auc_score : ', metrics.roc_auc_score(Y_test,y_predict))
```

```
testing parameters for Recursive feature elimination with SVC

Accuracy score: 0.9475308641975309

 f1 score      : 0.9435215946843853

roc_auc_score :  0.9470772882134517
```

CONCLUSION:

#1 after application of RFECV ON RandomForest Classifier and Support Vector Classificaation both accuracyscore and f1 score decreased.

#2performance scores were better beforhand application of Recurcive elimination