In [1]:

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

In [2]:

```
data = pd.read_csv("C:/Users/SANDEEP/OneDrive/Desktop/Eclipse/Eclipsedata3.csv")
data.shape
```

Out[2]:

```
(10593, 200)
```

In [3]:

```
for i in data.columns:
    data[i]=data[i].astype(int)
data
```

Out[3]:

```
       pre  ACD  FOUT_avg  FOUT_max  FOUT_sum  MLOC_avg  MLOC_max  MLOC_sum  NB
    0    0    0         1         1         1         7         7         7
    1    4    0         5        21        50         7        19        68
    2    2    0        21        30        63        30        38        90
    3    0    0         5        10        16        12        29        38
    4    4    0         7        18       110        12        29       183
  ...  ...  ...       ...       ...       ...       ...       ...       ...
10588    0    0         1         4        14         4         8        40
10589    0    1         2        11        46         4        17        96
10590    6    3         3         7        62         6        21       120
10591    0    0         4        12        42         8        21        78
10592    0    0         0         1         1         0         1         2

10593 rows × 200 columns
```

In [4]:
```
zero_cols = (data == 0).all()
data = data.loc[:, ~zero_cols]
data
```

Out[4]:

	pre	ACD	FOUT_avg	FOUT_max	FOUT_sum	MLOC_avg	MLOC_max	MLOC_sum	NB
0	0	0	1	1	1	7	7	7	
1	4	0	5	21	50	7	19	68	
2	2	0	21	30	63	30	38	90	
3	0	0	5	10	16	12	29	38	
4	4	0	7	18	110	12	29	183	
...	...	...	...	...	...	...	...	...	
10588	0	0	1	4	14	4	8	40	
10589	0	1	2	11	46	4	17	96	
10590	6	3	3	7	62	6	21	120	
10591	0	0	4	12	42	8	21	78	
10592	0	0	0	1	1	0	1	2	

10593 rows × 96 columns

In [6]:
```
corr=data.corr()
corr
```

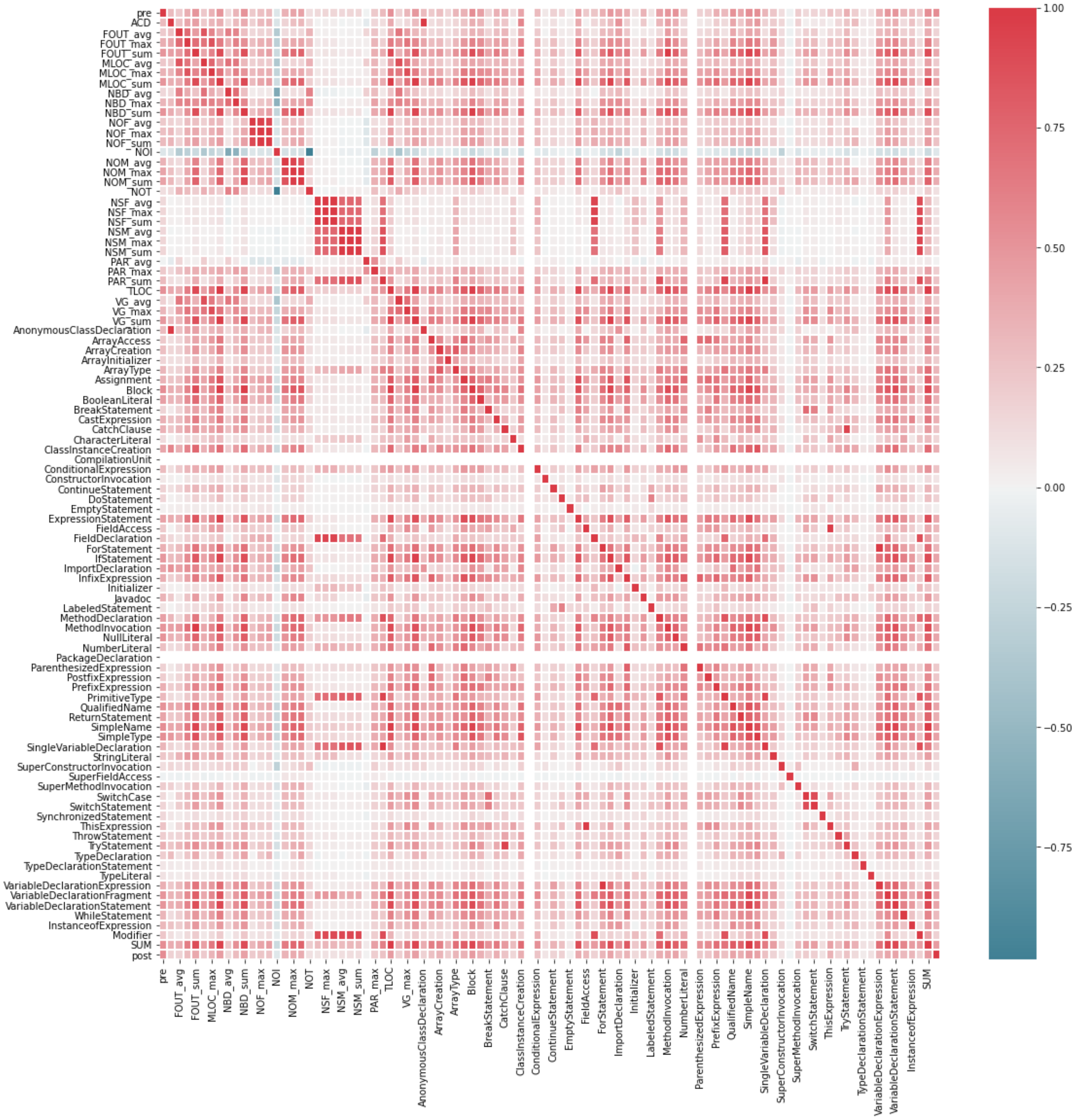
Out[6]:

	pre	ACD	FOUT_avg	FOUT_max	FOUT_sum	MLOC_avg	MLO
pre	1.000000	0.299642	0.211375	0.365085	0.501480	0.228496	0.
ACD	0.299642	1.000000	0.268620	0.414117	0.443453	0.220870	0.
FOUT_avg	0.211375	0.268620	1.000000	0.735744	0.469585	0.842860	0.
FOUT_max	0.365085	0.414117	0.735744	1.000000	0.705375	0.675737	0.
FOUT_sum	0.501480	0.443453	0.469585	0.705375	1.000000	0.411938	0.
...	...	...	...	...	...	...	
WhileStatement	0.328385	0.122946	0.185394	0.343531	0.488488	0.302098	0.
InstanceofExpression	0.316048	0.255800	0.204580	0.319602	0.485287	0.224644	0.
Modifier	0.239428	0.101620	0.051564	0.131366	0.201849	0.059347	0.
SUM	0.536728	0.377426	0.375488	0.626226	0.888933	0.435001	0.
post	0.547864	0.171447	0.167172	0.295234	0.399859	0.215783	0.

96 rows × 96 columns

In [7]:

```
f,ax=plt.subplots(figsize=(18,18))
cmap=sns.diverging_palette(220,10,as_cmap=True)
heatmap=sns.heatmap(corr,cmap=cmap,center=0.0,vmax=1,linewidths=1,ax=ax)
plt.show()
```



In [9]:

```
target_corr = corr['post']
top_features = target_corr.abs().sort_values(ascending=False)[:96].index
```

In [10]:

```
selected_data = data[top_features]
selected_data
```

Out[10]:

	post	pre	TLOC	Block	SUM	MLOC_sum	SimpleName	VG_sum	SimpleType	NBD_su
0	0	0	13	1	76	7	31	3	7	
1	0	4	122	19	979	68	443	19	66	
2	0	2	120	25	852	90	375	18	51	
3	0	0	64	12	418	38	187	13	26	
4	0	4	248	49	1386	183	624	46	81	
...	...	...	...	...	...	...	...	...	...	
10588	0	0	65	16	395	40	161	16	27	
10589	1	0	155	41	886	96	362	45	74	
10590	0	6	190	37	1000	120	418	36	66	
10591	2	0	109	27	646	78	266	27	40	
10592	0	0	13	3	64	2	25	3	3	

10593 rows × 96 columns

In [11]:

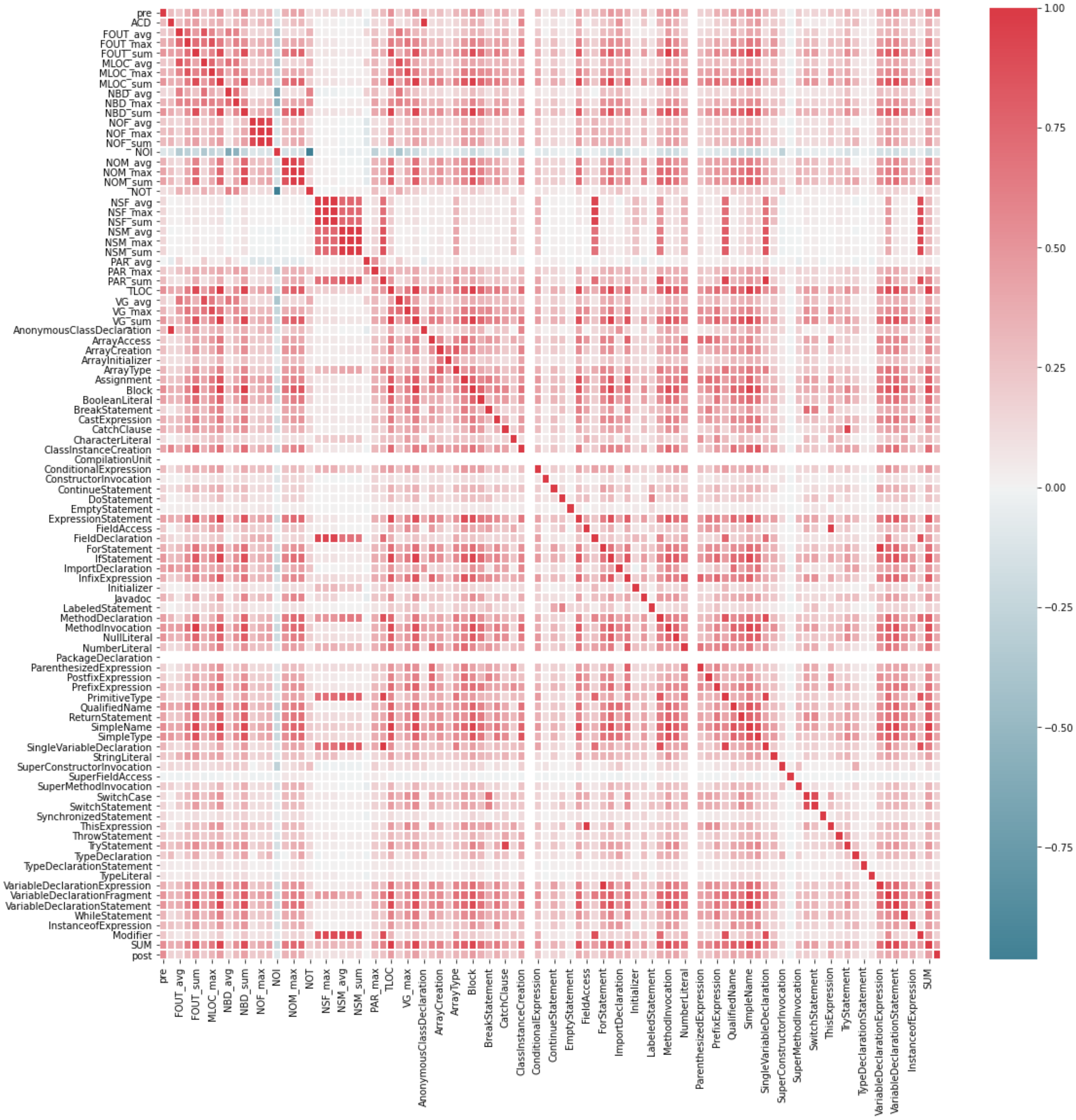
```
corr2=selected_data.corr()
corr2
```

Out[11]:

	post	pre	TLOC	Block	SUM	MLOC_sum	SimpleNan
post	1.000000	0.547864	0.472131	0.466297	0.465648	0.460866	0.4606
pre	0.547864	1.000000	0.525151	0.534935	0.536728	0.499238	0.5397
TLOC	0.472131	0.525151	1.000000	0.961094	0.978093	0.982800	0.9739
Block	0.466297	0.534935	0.961094	1.000000	0.921820	0.956723	0.9326
SUM	0.465648	0.536728	0.978093	0.921820	1.000000	0.944784	0.9856
...	...	...	...	...	...	...	
EmptyStatement	0.014660	0.014466	0.058003	0.053716	0.052484	0.057172	0.0512
SuperFieldAccess	-0.013429	-0.013032	-0.002449	0.008266	0.000016	-0.010722	-0.0026
PAR_avg	0.003907	-0.001759	0.030887	0.018436	0.053069	0.038187	0.0552
CompilationUnit	NaN	NaN	NaN	NaN	NaN	NaN	Na
PackageDeclaration	NaN	NaN	NaN	NaN	NaN	NaN	Na

96 rows × 96 columns

In [12]:
```
f,ax=plt.subplots(figsize=(18,18))
cmap=sns.diverging_palette(220,10,as_cmap=True)
heatmap=sns.heatmap(corr,cmap=cmap,center=0.0,vmax=1,linewidths=1,ax=ax)
plt.show()
```



In [13]:
```
selected_data=selected_data.drop(['post'],axis=1)
selected_data
```

Out[13]:

	pre	TLOC	Block	SUM	MLOC_sum	SimpleName	VG_sum	SimpleType	NBD_sum	IfS
0	0	13	1	76	7	31	3	7	1	
1	4	122	19	979	68	443	19	66	13	
2	2	120	25	852	90	375	18	51	11	
3	0	64	12	418	38	187	13	26	9	
4	4	248	49	1386	183	624	46	81	31	
...	...	...	...	...	...	...	...	...	...	
10588	0	65	16	395	40	161	16	27	16	
10589	0	155	41	886	96	362	45	74	37	
10590	6	190	37	1000	120	418	36	66	34	
10591	0	109	27	646	78	266	27	40	22	
10592	0	13	3	64	2	25	3	3	3	

10593 rows × 95 columns

In [ ]:
```
#XGBOOST
```

In [15]:
```
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn import metrics
```

In [16]:
```
X = selected_data
y = data['post']
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2, random_state=1)
```

In [17]:
```
params = {'objective': 'binary:hinge', 'eval_metric': 'logloss'}
```

In [18]:
```
dtrain = xgb.DMatrix(X_train, label=y_train)
dtest = xgb.DMatrix(X_test, label=y_test)
```

In [19]:
```
model = xgb.train(params, dtrain, num_boost_round=100)
```

In [20]:
```
y_pred = model.predict(dtest)
```

In [21]:
```
y_pred = [1 if p >= 0.5 else 0 for p in y_pred]
```

In [22]:

```
a=metrics.accuracy_score(y_test,y_pred)
p=metrics.precision_score(y_test,y_pred,average='weighted',zero_division=0)
r=metrics.recall_score(y_test,y_pred,average='weighted',zero_division=0)
f1=metrics.f1_score(y_test,y_pred,average='weighted',zero_division=0)
print("Accuracy:",a,"Precision: ",p,"Recall: ",r,"F1score: ",f1)
```

```
Accuracy: 0.8329400660689005 Precision:  0.7929653597384768 Recall:  0.83294006606890
05 F1score:  0.8124520447302954
```

In [24]:

```
#RUS
```

In [26]:

```
from imblearn.under_sampling import RandomUnderSampler
from sklearn.linear_model import LogisticRegression
```

In [27]:

```
Xr = selected_data
yr = data['post']
Xr_train, Xr_test, yr_train, yr_test = train_test_split(Xr, yr, test_size=0.3, random_state=1)
```

In [28]:

```
rus = RandomUnderSampler(random_state=42)
X_train_rus, y_train_rus = rus.fit_resample(Xr_train, yr_train)
```

In [29]:

```
lr = LogisticRegression(random_state=42)
lr.fit(X_train_rus, y_train_rus)
```

```
C:\Users\SANDEEP\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:458: Conv
ergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/mod
ules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-lea
rn.org/stable/modules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
```

Out[29]:

```
LogisticRegression(random_state=42)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [30]:

```
yr_pred = lr.predict(Xr_test)
```

In [33]:

```
a=metrics.accuracy_score(yr_test,yr_pred)
p=metrics.precision_score(yr_test,yr_pred,average='weighted',zero_division=0)
r=metrics.recall_score(yr_test,yr_pred,average='weighted',zero_division=0)
f1=metrics.f1_score(y_test,y_pred,average='weighted',zero_division=0)
print("Accuracy:",a,"Precision: ",p,"Recall: ",r,"F1score: ",f1)
```

```
Accuracy: 0.06261799874134676 Precision:  0.7929084212030609 Recall:  0.0626179987413
4676 F1score:  0.8124520447302954
```

In [34]:

```
#KNN
```

In [35]:

```
from sklearn.neighbors import KNeighborsClassifier
```

In [36]:

```
Xn= selected_data
yn = data['post']
Xn_train, Xn_test, yn_train, yn_test = train_test_split(Xn, yn, test_size=0.3, random_state=1)
```

In [37]:

```
knn = KNeighborsClassifier(n_neighbors=3)
```

In [38]:

```
knn.fit(Xn_train, yn_train)
```

Out[38]:

```
KNeighborsClassifier(n_neighbors=3)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [39]:

```
yn_pred = knn.predict(Xn_test)
```

In [40]:

```
a=metrics.accuracy_score(yn_test,yn_pred)
p=metrics.precision_score(yn_test,yn_pred,average='weighted',zero_division=0)
r=metrics.recall_score(yn_test,yn_pred,average='weighted',zero_division=0)
f1=metrics.f1_score(y_test,y_pred,average='weighted',zero_division=0)
print("Accuracy:",a,"Precision: ",p,"Recall: ",r,"F1score: ",f1)
```

```
Accuracy: 0.8335431088735054 Precision:  0.7661762498654886 Recall:  0.83354310887350
54 F1score:  0.8124520447302954
```

In [41]:

```
#Voting
```

In [42]:

```
from sklearn.ensemble import VotingClassifier
from xgboost import XGBClassifier
```

In [43]:

```
Xv= selected_data
yv = data['post']
Xv_train, Xv_test, yv_train, yv_test = train_test_split(Xv, yv, test_size=0.3, random_state=1)
```

In [44]:

```
model1 = KNeighborsClassifier(n_neighbors=5)
model3 = XGBClassifier(n_estimators=100, learning_rate=0.1,objective='binary:logistic',eval_me
model2 = LogisticRegression(random_state=42)
```

In [45]:

```
voting_clf = VotingClassifier(estimators=[('xgb', model1), ('rus', model2),('knn', model3)], voting=
```

In [46]:

```
voting_clf.fit(Xv_train, yv_train)
```

```
C:\Users\SANDEEP\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:458: Conv
ergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/mod
ules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-lea
rn.org/stable/modules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
```

Out[46]:

```
VotingClassifier(estimators=[('xgb', KNeighborsClassifier()),
                             ('rus', LogisticRegression(random_state=42)),
                             ('knn',
                              XGBClassifier(base_score=None, booster=None,
                                            callbacks=None,
                                            colsample_bylevel=None,
                                            colsample_bynode=None,
                                            colsample_bytree=None,
                                            early_stopping_rounds=None,
                                            enable_categorical=False,
                                            eval_metric='logloss',
                                            feature_types=None, gamma=None,
                                            gpu_id=None, grow_policy=None,
                                            importance_type=None,
                                            interaction_constraints=None,
                                            learning_rate=0.1, max_bin=None,
                                            max_cat_threshold=None,
                                            max_cat_to_onehot=None,
                                            max_delta_step=None, max_depth=None,
                                            max_leaves=None,
                                            min_child_weight=None, missing=nan,
                                            monotone_constraints=None,
                                            n_estimators=100, n_jobs=None,
                                            num_parallel_tree=None,
                                            predictor=None, random_state=None, ...))])
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [47]:

```
yv_pred = voting_clf.predict(Xv_test)
```

In [48]:

```
a=metrics.accuracy_score(yv_test,yv_pred)
p=metrics.precision_score(yv_test,yv_pred,average='weighted',zero_division=0)
r=metrics.recall_score(yv_test,yv_pred,average='weighted',zero_division=0)
f1=metrics.f1_score(y_test,y_pred,average='weighted',zero_division=0)
print("Accuracy:",a,"Precision: ",p,"Recall: ",r,"F1score: ",f1)
```

```
Accuracy: 0.8568281938325991 Precision:  0.8003623577306566 Recall:  0.85682819383259
91 F1score:  0.8124520447302954
```

In [ ]:

```

```