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/Eclipsedata1.csv")
data.shape
```

Out[2]: (6729, 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	NBD
0	1	0	6	29	54	9	32	74	
1	1	0	12	13	25	16	18	32	
2	0	0	5	10	16	12	29	38	
3	2	0	7	16	88	9	28	116	
4	2	4	6	27	118	9	55	188	
...	...	...	...	...	...	...	...	...	
6724	0	0	5	14	15	14	30	43	
6725	1	0	2	5	5	5	9	11	
6726	0	0	3	7	21	4	7	29	
6727	0	0	4	16	103	9	27	203	
6728	0	0	1	2	14	7	10	71	

6729 rows × 200 columns

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

In [5]:
```
data = data.loc[:, ~zero_cols]
data
```

Out[5]:

	pre	ACD	FOUT_avg	FOUT_max	FOUT_sum	MLOC_avg	MLOC_max	MLOC_sum	NBD
0	1	0	6	29	54	9	32	74	
1	1	0	12	13	25	16	18	32	
2	0	0	5	10	16	12	29	38	
3	2	0	7	16	88	9	28	116	
4	2	4	6	27	118	9	55	188	
...	...	...	...	...	...	...	...	...	
6724	0	0	5	14	15	14	30	43	
6725	1	0	2	5	5	5	9	11	
6726	0	0	3	7	21	4	7	29	
6727	0	0	4	16	103	9	27	203	
6728	0	0	1	2	14	7	10	71	

6729 rows × 96 columns

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

Out[6]:

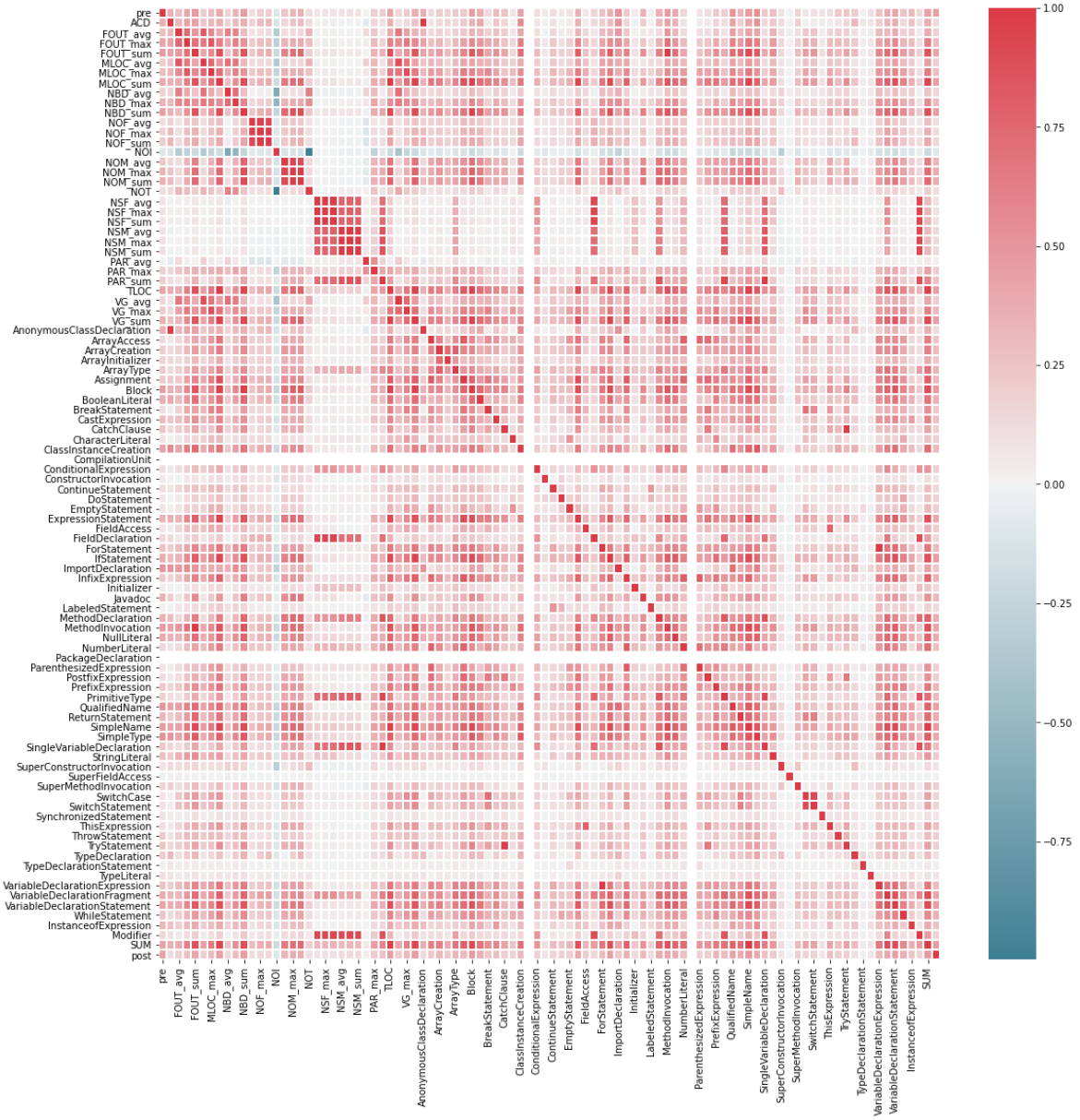
	pre	ACD	FOUT_avg	FOUT_max	FOUT_sum	MLOC_avg	MLO
pre	1.000000	0.360430	0.274687	0.407330	0.471807	0.206010	0.2
ACD	0.360430	1.000000	0.320713	0.443755	0.438841	0.256226	0.3
FOUT_avg	0.274687	0.320713	1.000000	0.744481	0.495558	0.835760	0.5
FOUT_max	0.407330	0.443755	0.744481	1.000000	0.698143	0.670054	0.8
FOUT_sum	0.471807	0.438841	0.495558	0.698143	1.000000	0.417283	0.5
...	...	...	...	...	...	...	
WhileStatement	0.250002	0.125348	0.182384	0.322650	0.472348	0.298575	0.4
InstanceofExpression	0.325988	0.269866	0.195501	0.310556	0.380633	0.211123	0.2
Modifier	0.153219	0.097331	0.040545	0.111295	0.188929	0.047577	0.
SUM	0.438837	0.374507	0.387679	0.617531	0.864129	0.449573	0.6
post	0.443016	0.119252	0.175573	0.284341	0.419170	0.196604	0.2

96 rows × 96 columns

In [7]:
```
corr.shape
```

Out[7]: (96, 96)

In [8]:
```
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	Block	SimpleName	SUM	TLOC	NBD_sum	VG_sum	IfStatement	VariableDeclarat
0	0	22	458	1011	128	14	23	12	
1	0	7	197	437	55	4	8	2	
2	0	12	217	478	70	9	13	9	
3	0	42	512	1136	174	25	34	16	
4	0	52	725	1698	277	34	46	16	
...	...	...	...	...	...	...	...	...	
6724	0	4	161	341	68	4	5	2	
6725	0	5	48	115	22	4	4	0	
6726	0	10	136	341	54	10	11	4	
6727	0	50	517	1331	259	45	73	31	
6728	0	28	234	541	130	17	17	0	

6729 rows × 96 columns

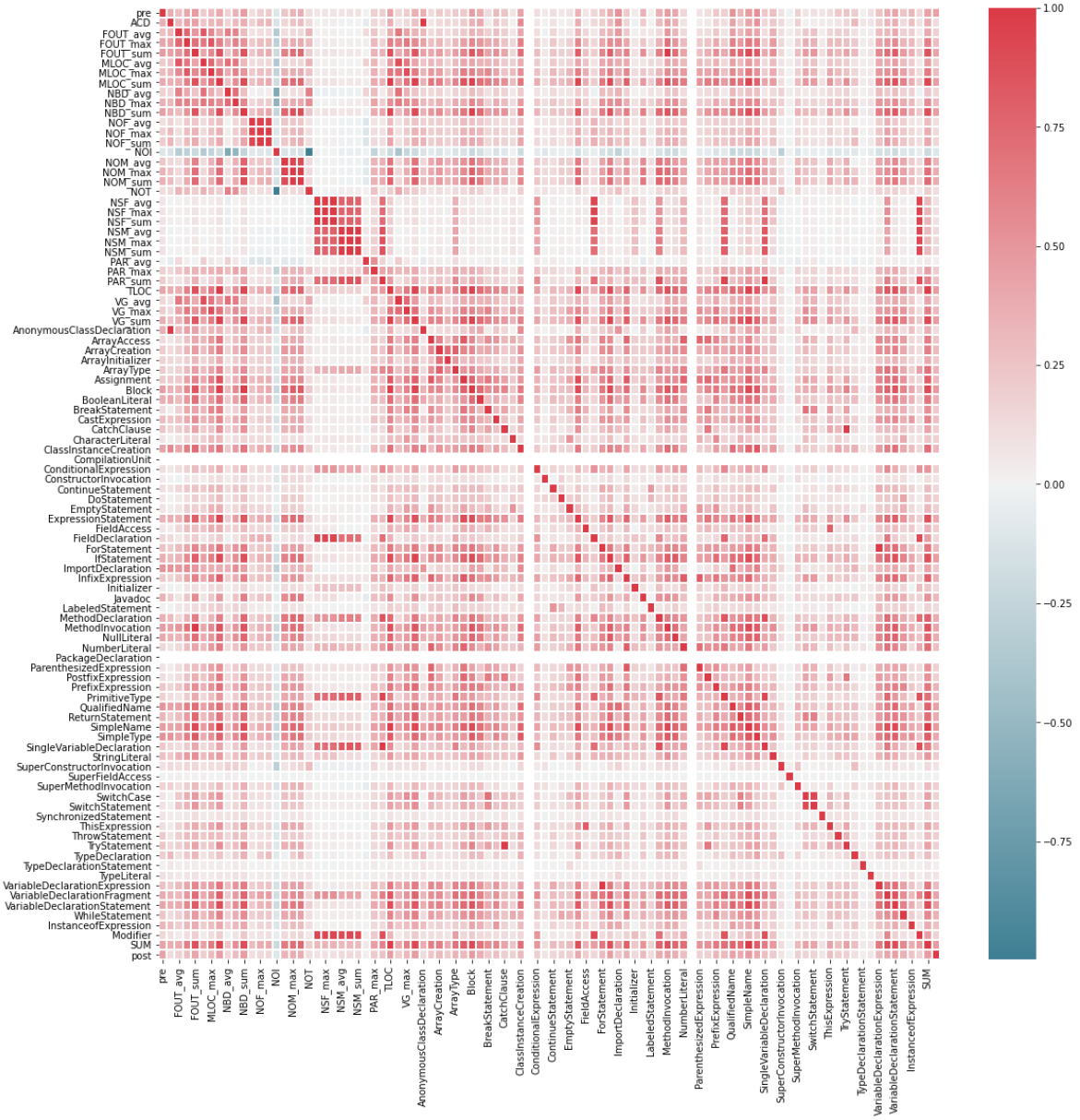
In [11]:
```
corr2=selected_data.corr()
corr2
```

Out[11]:

	post	Block	SimpleName	SUM	TLOC	NBD_sum	V
post	1.000000	0.477872	0.470122	0.467638	0.460576	0.458973	0
Block	0.477872	1.000000	0.911565	0.903672	0.958016	0.965443	0
SimpleName	0.470122	0.911565	1.000000	0.985329	0.964736	0.893706	0
SUM	0.467638	0.903672	0.985329	1.000000	0.970832	0.880548	0
TLOC	0.460576	0.958016	0.964736	0.970832	1.000000	0.923550	0
...	...	...	...	...	...	...	
TypeDeclarationStatement	0.018132	0.031242	0.048882	0.050949	0.038080	0.033692	0
PAR_avg	0.016561	0.012002	0.049167	0.048541	0.027357	-0.021865	0
SuperFieldAccess	-0.006652	-0.000962	-0.000460	-0.001131	0.000035	-0.001058	-0
CompilationUnit	NaN	NaN	NaN	NaN	NaN	NaN	
PackageDeclaration	NaN	NaN	NaN	NaN	NaN	NaN	

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]:

	Block	SimpleName	SUM	TLOC	NBD_sum	VG_sum	IfStatement	VariableDeclarationSta
0	22	458	1011	128	14	23	12	
1	7	197	437	55	4	8	2	
2	12	217	478	70	9	13	9	
3	42	512	1136	174	25	34	16	
4	52	725	1698	277	34	46	16	
...	...	...	...	...	...	...	...	
6724	4	161	341	68	4	5	2	
6725	5	48	115	22	4	4	0	
6726	10	136	341	54	10	11	4	
6727	50	517	1331	259	45	73	31	
6728	28	234	541	130	17	17	0	

6729 rows × 95 columns

In [14]:
```
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn import metrics
```

In [15]:
```
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 [16]:
```
params = {'objective': 'binary:hinge', 'eval_metric': 'logloss'}
```

In [17]:
```
dtrain = xgb.DMatrix(X_train, label=y_train)
dtest = xgb.DMatrix(X_test, label=y_test)
```

In [18]:
```
model = xgb.train(params, dtrain, num_boost_round=100)
```

In [19]:
```
y_pred = model.predict(dtest)
```

In [20]:
```
y_pred = [1 if p >= 0.5 else 0 for p in y_pred]
```

In [21]:
```
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: 0.8164933135215453 Precision:  0.7810296563020843 Recall:  0.81649331352154
53 F1score:  0.7979638252822547
```

In [ ]:
```
#RUS
```

In [22]:
```
from imblearn.under_sampling import RandomUnderSampler
from sklearn.linear_model import LogisticRegression
from imblearn.over_sampling import SMOTE
```

In [23]:
```
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 [24]:
```
rus = RandomUnderSampler(random_state=42)
X_train_rus, y_train_rus = rus.fit_resample(Xr_train, yr_train)
```

In [25]:
```
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[25]:
```
LogisticRegression(random_state=42)
```

In [26]:
```
yr_pred = lr.predict(Xr_test)
```

In [27]:
```
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: 0.5418524021792966 Precision:  0.7667700081643148 Recall:  0.5418524021792966 F1score:  0.7979638252822547
```

In [162]:
```
#KNN
```

In [28]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [29]:
```
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 [30]:
```
knn = KNeighborsClassifier(n_neighbors=3)
```

In [31]:
```
knn.fit(Xn_train, yn_train)
```

Out[31]:

KNeighborsClassifier

KNeighborsClassifier(n_neighbors=3)

In [32]:
```
yn_pred = knn.predict(Xn_test)
```

In [33]:
```
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: 0.8405151064883606 Precision:  0.7824634558085621 Recall:  0.8405151064883606 F1score:  0.7979638252822547
```

In [177]:
```
#Voting ensemble
```

In [34]:
```
from sklearn.ensemble import VotingClassifier
from xgboost import XGBClassifier
```

In [35]:
```
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 [36]:
```
model1 = KNeighborsClassifier(n_neighbors=5)
model3 = XGBClassifier(n_estimators=100, learning_rate=0.1,objective='binary:logistic',eval_me
model2 = LogisticRegression(random_state=42)
```

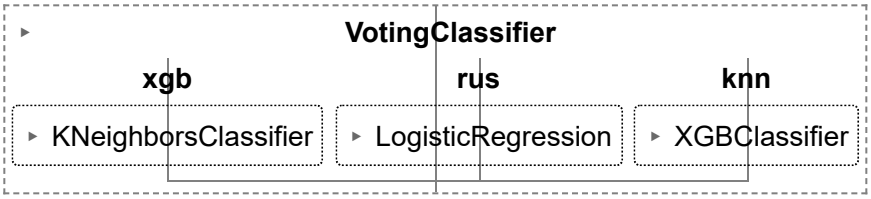
In [37]:
```
voting_clf = VotingClassifier(estimators=[('xgb', model1), ('rus', model2),('knn', model3)], voting=
```

In [38]:
```
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[38]:



In [39]:
```
yv_pred = voting_clf.predict(Xv_test)
```

In [40]:
```
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: 0.849925705794948 Precision:  0.7896247280586621 Recall:  0.849925705794948
F1score:  0.7979638252822547
```