In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [3]:
```
data = pd.read_csv("C:/Users/SANDEEP/OneDrive/Desktop/NASA datasers/pc1_csv.csv")
```

In [4]:
```
for i in data.columns:
    data[i]=data[i].astype(int)
data
```

Out[4]:

	loc	v(g)	ev(g)	iv(G)	N	V	L	D	I	E	...	lOCode	lOComment	locC
0	1	1	1	1	1	1	1	1	1	1	...	2	2	
1	1	1	1	1	1	1	1	1	1	1	...	1	1	
2	91	9	3	2	318	2089	0	27	75	57833	...	80	44	
3	109	21	5	18	381	2547	0	28	89	72282	...	97	41	
4	505	106	41	82	2339	20696	0	75	272	1571506	...	457	71	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
1104	6	4	4	1	26	96	0	13	7	1282	...	6	0	
1105	10	5	5	1	43	182	0	21	8	3835	...	10	0	
1106	5	3	3	1	17	62	0	4	13	301	...	5	0	
1107	18	8	5	5	111	613	0	22	26	14050	...	18	0	
1108	26	18	13	6	228	1335	0	35	37	47834	...	26	0	

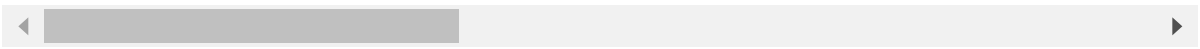
1109 rows × 22 columns

In [5]:
```
corr=data.corr()
corr
```

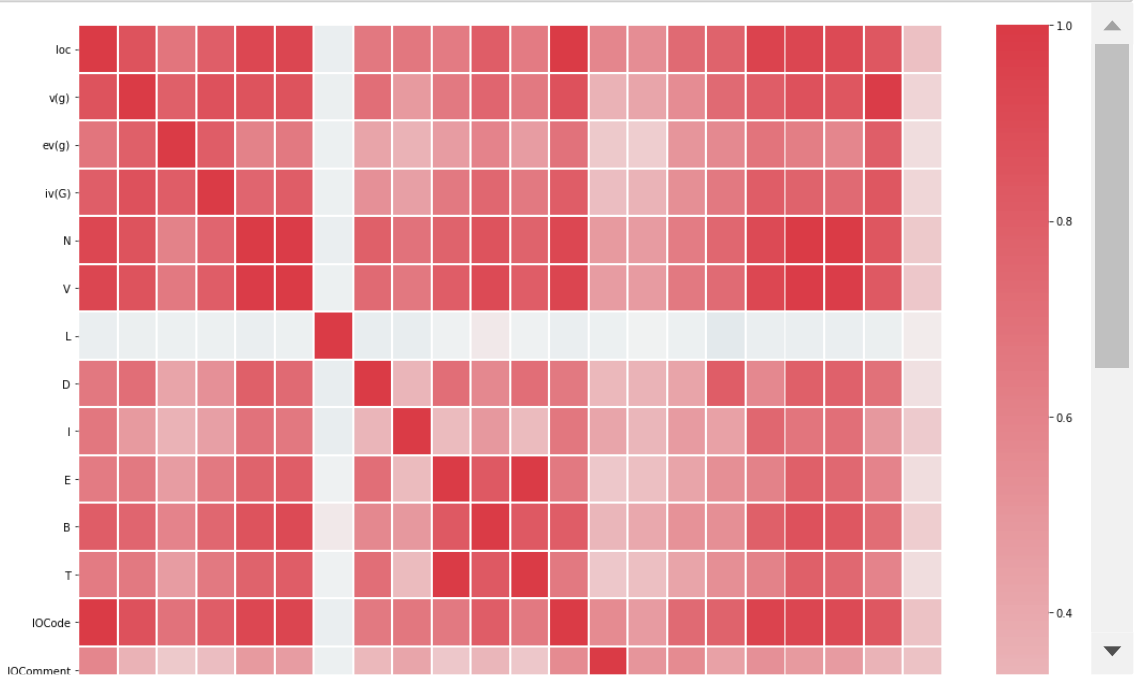
Out[5]:

	loc	v(g)	ev(g)	iv(G)	N	V	L
loc	1.000000	0.864420	0.675679	0.797270	0.923664	0.936537	-0.028365
v(g)	0.864420	1.000000	0.793415	0.868095	0.860175	0.861484	-0.024728
ev(g)	0.675679	0.793415	1.000000	0.803969	0.609799	0.654714	-0.015583
iv(G)	0.797270	0.868095	0.803969	1.000000	0.759722	0.805140	-0.017807
N	0.923664	0.860175	0.609799	0.759722	1.000000	0.987272	-0.028846
V	0.936537	0.861484	0.654714	0.805140	0.987272	1.000000	-0.022704
L	-0.028365	-0.024728	-0.015583	-0.017807	-0.028846	-0.022704	1.000000
D	0.656516	0.712694	0.423175	0.524299	0.791267	0.731993	-0.043738
I	0.664536	0.482435	0.346839	0.453230	0.691596	0.662762	-0.041540
E	0.643407	0.650214	0.463020	0.649317	0.774191	0.803556	-0.008296
B	0.803769	0.757663	0.597687	0.749576	0.858304	0.909768	0.050503
T	0.643405	0.650213	0.463021	0.649316	0.774190	0.803555	-0.008291
lOCode	0.996509	0.866482	0.689939	0.804101	0.923509	0.937492	-0.028066
lOComment	0.578843	0.348095	0.221211	0.283693	0.475649	0.462923	-0.018427
locCodeAndComment	0.545351	0.420476	0.201804	0.338457	0.472906	0.468479	-0.002863
lOBlank	0.735256	0.551908	0.502349	0.532557	0.635768	0.650029	-0.021944
uniq_Op	0.778583	0.738266	0.567062	0.650649	0.753042	0.729357	-0.073879
uniq_Opnd	0.946538	0.809928	0.681466	0.805286	0.907747	0.928627	-0.033633
total_Op	0.929811	0.867681	0.626903	0.775931	0.997363	0.990269	-0.028798
total_Opnd	0.908472	0.843721	0.582886	0.732795	0.995579	0.975662	-0.028399
branchCount	0.838833	0.991572	0.796042	0.839564	0.841362	0.835876	-0.025467
defects	0.267560	0.157544	0.113450	0.154821	0.221494	0.228565	0.034876

22 rows × 22 columns



In [6]:
```
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 [7]:
```
dat=data.drop(['E','L','ev(g)','iv(G)','v(g)','branchCount','loc','lOBlank'],axis=1)
dat.shape
```

Out[7]: (1109, 14)

In [8]:
```
lis= ['v(g)','ev(g)','T','V','D','I','lOCode','uniq_Op','uniq_Opnd','total_Op','total_Opnd']
fea=data[lis]
fea
```

Out[8]:

	v(g)	ev(g)	T	V	D	I	lOCode	uniq_Op	uniq_Opnd	total_Op	total_Opnd
0	1	1	1	1	1	1	2	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
2	9	3	3212	2089	27	75	80	29	66	192	126
3	21	5	4015	2547	28	89	97	28	75	229	152
4	106	41	87305	20696	75	272	457	64	397	1397	942
...	...	...	...	...	...	...	...	...	...	...	...
1104	4	4	71	96	13	7	6	10	3	18	8
1105	5	5	213	182	21	8	10	14	5	28	15
1106	3	3	16	62	4	13	5	8	5	11	6
1107	8	5	780	613	22	26	18	22	24	61	50
1108	18	13	2657	1335	35	37	26	23	35	119	109

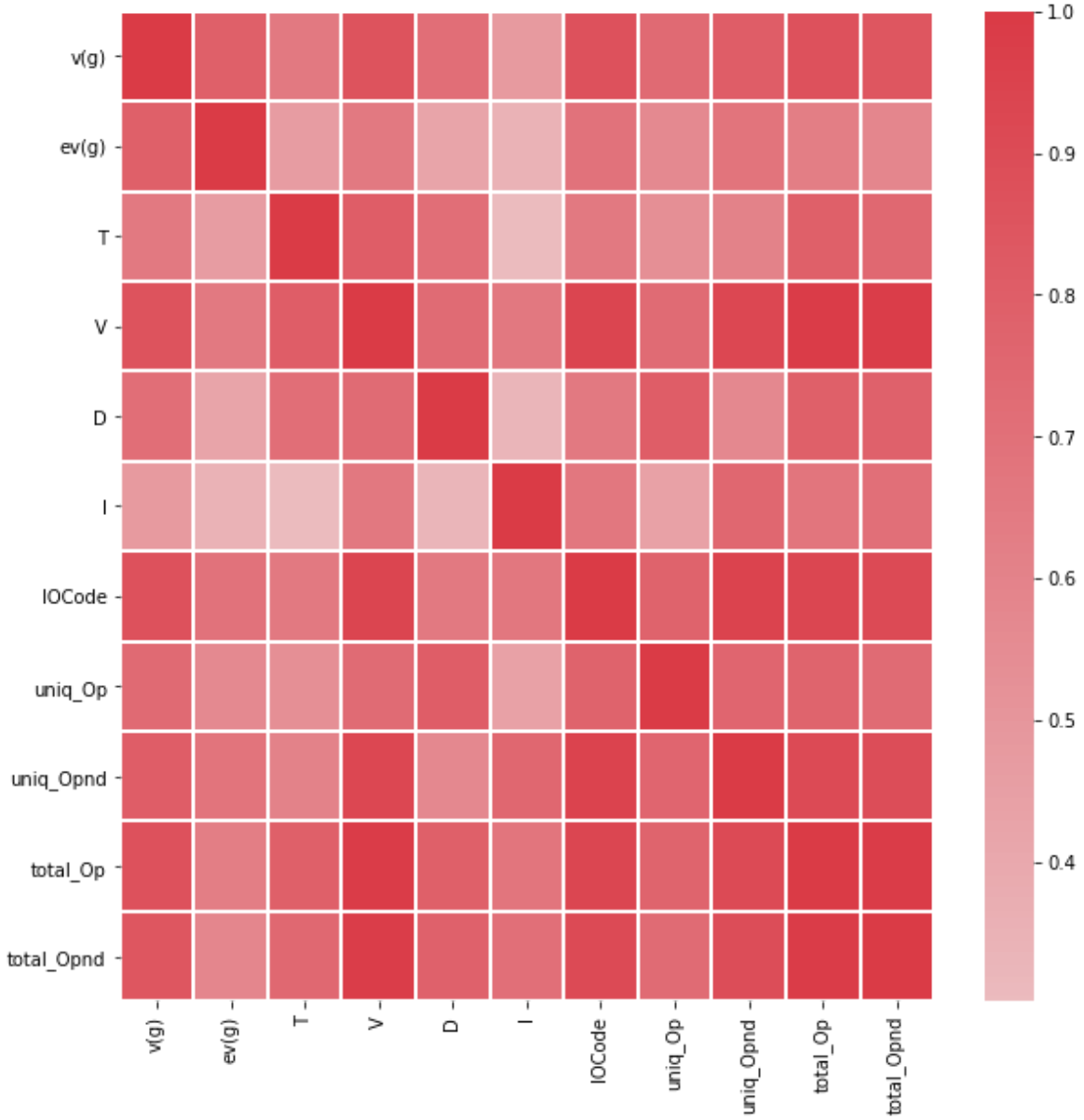
1109 rows × 11 columns

In [9]:
```
corr2=fea.corr()
corr2
```

Out[9]:

	v(g)	ev(g)	T	V	D	I	IOCode	uniq_Op	un
v(g)	1.000000	0.793415	0.650213	0.861484	0.712694	0.482435	0.866482	0.738266	(
ev(g)	0.793415	1.000000	0.463021	0.654714	0.423175	0.346839	0.689939	0.567062	(
T	0.650213	0.463021	1.000000	0.803555	0.714123	0.301476	0.648433	0.533072	(
V	0.861484	0.654714	0.803555	1.000000	0.731993	0.662762	0.937492	0.729357	(
D	0.712694	0.423175	0.714123	0.731993	1.000000	0.331189	0.655442	0.806737	(
I	0.482435	0.346839	0.301476	0.662762	0.331189	1.000000	0.666192	0.438137	(
IOCode	0.866482	0.689939	0.648433	0.937492	0.655442	0.666192	1.000000	0.779182	(
uniq_Op	0.738266	0.567062	0.533072	0.729357	0.806737	0.438137	0.779182	1.000000	(
uniq_Opnd	0.809928	0.681466	0.607232	0.928627	0.575535	0.755229	0.949123	0.763300	
total_Op	0.867681	0.626903	0.792128	0.990269	0.792524	0.673725	0.929557	0.767891	(
total_Opnd	0.843721	0.582886	0.744915	0.975662	0.783433	0.709298	0.908446	0.727902	(

In [10]:
```
f,ax=plt.subplots(figsize=(10,10))
cmap=sns.diverging_palette(220,10,as_cmap=True)
heatmap=sns.heatmap(corr2,cmap=cmap,center=0.0,vmax=1,linewidths=1,ax=ax)
plt.show()
```



In [11]:
```
#KNN
```

In [13]:
```
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from imblearn.under_sampling import RandomUnderSampler
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
```

In [14]:
```
X = fea
y = data['defects']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)
```

In [15]:
```
smote = SMOTE(random_state=1)
X_train_resampled,y_train_resampled = smote.fit_resample(X_train,y_train)
```

In [16]:
```
knn = KNeighborsClassifier(n_neighbors=3)
```

In [17]:
```
knn.fit(X_train, y_train)
```

Out[17]:
```
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 [18]:
```
y_pred = knn.predict(X_test)
```

In [19]:
```
a=metrics.accuracy_score(y_test,y_pred)
p=metrics.precision_score(y_test,y_pred)
r=metrics.recall_score(y_test,y_pred)
f1=2*(p*r)/(p+r)
print("Accuracy:",a,"  Precision:",p,"  Recall:",r,"  F1Score:",f1)
```

```
Accuracy: 0.9129129129129129  Precision: 0.4117647058823529  Recall: 0.2692307692307692  F1Score: 0.3255813953488372
```

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

In [20]:
```
Xr = fea
yr = data['defects']
Xr_train, Xr_test, yr_train, yr_test = train_test_split(Xr, yr, test_size=0.3, random_state=1)
```

In [21]:
```
smote = SMOTE(random_state=42)
X_train_resampled,y_train_resampled = smote.fit_resample(Xr_train,yr_train)
```

In [22]:
```
rus = RandomUnderSampler(random_state=42)
X_train_rus, y_train_rus = rus.fit_resample(Xr_train, yr_train)
```

In [24]:
```
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[24]:
```
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 [25]:
```
yr_pred = lr.predict(Xr_test)
```

In [26]:
```
a=metrics.accuracy_score(yr_test,yr_pred)
p=metrics.precision_score(yr_test,yr_pred)
r=metrics.recall_score(yr_test,yr_pred)
f1=2*(p*r)/(p+r)
print("Accuracy:",a,"   Precision:",p," Recall:",r,"   F1Score:",f1)
```

```
Accuracy: 0.7177177177177178    Precision: 0.13829787234042554    Recall: 0.5    F1Score:
0.21666666666666667
```

In [27]:
```
#XGBOOST
```

In [28]:
```
import xgboost as xgb
```

In [29]:
```
Xx = data[lis]
yx = data["defects"]
Xx_train, Xx_test, yx_train, yx_test = train_test_split(Xx, yx, test_size=0.3, random_state=1)
```

In [30]:
```
smote = SMOTE(random_state=1)
X_train_resampled,y_train_resampled = smote.fit_resample(Xx_train,yx_train)
```

In [31]:
```
params = {'objective': 'binary:logistic', 'eval_metric': 'logloss'}
```

In [32]:
```
dtrain = xgb.DMatrix(Xx_train, label=yx_train)
dtest = xgb.DMatrix(Xx_test, label=yx_test)
```

In [33]:
```
model = xgb.train(params, dtrain, num_boost_round=100)
```

In [34]:
```
yx_pred = model.predict(dtest)
```

In [35]:
```
yx_pred = [1 if p >= 0.5 else 0 for p in yx_pred]
```

In [36]:
```
a=metrics.accuracy_score(yx_test,yx_pred)
p=metrics.precision_score(yx_test,yx_pred)
r=metrics.recall_score(yx_test,yx_pred)
f1=2*(p*r)/(p+r)
print("Accuracy:",a,"  Precision:",p,"  Recall:",r,"  F1Score:",f1)
```

```
Accuracy: 0.9099099099099099  Precision: 0.4  Recall: 0.3076923076923077  F1Score:
0.34782608695652173
```

In [ ]:
```
#Voting Ensemble
```

In [38]:
```
from sklearn.ensemble import VotingClassifier
from xgboost import XGBClassifier
```

In [39]:
```
X = fea
y = data['defects']
```

In [40]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

In [41]:
```
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 [42]:
```
voting_clf = VotingClassifier(estimators=[('xgb', model1), ('rus', model2),('knn', model3)], voting=
```

In [43]:
```
voting_clf.fit(X_train, y_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[43]:
```
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 [44]:
```
y_predic = voting_clf.predict(X_test)
```

In [45]:
```
a=metrics.accuracy_score(y_test,y_predic)
p=metrics.precision_score(y_test,y_predic)
r=metrics.recall_score(y_test,y_predic)
f1=metrics.f1_score(y_test,y_predic)
print("Accuracy:",a,"Precision: ",p,"Recall: ",r,"F1score: ",f1)
```

```
Accuracy: 0.918918918918919 Precision:  0.5 Recall:  0.1111111111111111 F1score:  0.181818
1818181818
```