In [65]:
```python
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

from sklearn.preprocessing import StandardScaler,RobustScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV,KFold,StratifiedKFold,train_test_spli
from sklearn.metrics import f1_score,accuracy_score,roc_auc_score,auc
from xgboost import XGBClassifier

import matplotlib.pyplot as plt
import seaborn as sns
```

In [58]:
```python
path="creditcard.csv (6).zip"
data=pd.read_csv(path)
```

In [59]:
```python
usage=data.memory_usage(index=False)
TO_MB=(1024)**2
print(f"All Columns Has Same Size Of {round(usage[0]/TO_MB)} MB \n The Sum Of Total Me
```

```
All Columns Has Same Size Of 2 MB
 The Sum Of Total Memory Usage Is 67 MB
```

In [60]:
```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64  
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

In [61]:
```python
data.isnull().sum()
```

Out[61]:
```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

In [62]:
```python
data.shape # before removing null values
```

Out[62]:
```
(284807, 31)
```

In [30]:
```python
data.head(5)   # before preprocessing
```

Out[30]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	149.62	0
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724	2.69	0
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	378.66	0
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.061458	123.50	0
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.215153	69.99	0

5 rows × 31 columns

In [31]:
```python
data.dropna(how='any',inplace=True)
print(data)
```

```
            Time         V1         V2        V3        V4        V5  \
0            0.0  -1.359807  -0.072781  2.536347  1.378155 -0.338321   
1            0.0   1.191857   0.266151  0.166480  0.448154  0.060018   
2            1.0  -1.358354  -1.340163  1.773209  0.379780 -0.503198   
3            1.0  -0.966272  -0.185226  1.792993 -0.863291 -0.010309   
4            2.0  -1.158233   0.877737  1.548718  0.403034 -0.407193   
...          ...        ...        ...       ...       ...       ...   
284802  172786.0 -11.881118  10.071785 -9.834783 -2.066656 -5.364473   
284803  172787.0  -0.732789  -0.055080  2.035030 -0.738589  0.868229   
284804  172788.0   1.919565  -0.301254 -3.249640 -0.557828  2.630515   
284805  172788.0  -0.240440   0.530483  0.702510  0.689799 -0.377961   
284806  172792.0  -0.533413  -0.189733  0.703337 -0.506271 -0.012546   

              V6        V7        V8        V9  ...       V21       V22  \
0       0.462388  0.239599  0.098698  0.363787  ... -0.018307  0.277838   
1      -0.082361 -0.078803  0.085102 -0.255425  ... -0.225775 -0.638672   
2       1.800499  0.791461  0.247676 -1.514654  ...  0.247998  0.771679   
3       1.247203  0.237609  0.377436 -1.387024  ... -0.108300  0.005274   
4       0.095921  0.592941 -0.270533  0.817739  ... -0.009431  0.798278   
...          ...       ...       ...       ...  ...       ...       ...   
284802 -2.606837 -4.918215  7.305334  1.914428  ...  0.213454  0.111864   
284803  1.058415  0.024330  0.294869  0.584800  ...  0.214205  0.924384   
284804  3.031260 -0.296827  0.708417  0.432454  ...  0.232045  0.578229   
284805  0.623708 -0.686180  0.679145  0.392087  ...  0.265245  0.800049   
284806 -0.649617  1.577006 -0.414650  0.486180  ...  0.261057  0.643078   

             V23       V24       V25       V26       V27       V28  Amount  \
0      -0.110474  0.066928  0.128539 -0.189115  0.133558 -0.021053  149.62   
1       0.101288 -0.339846  0.167170  0.125895 -0.008983  0.014724    2.69   
2       0.909412 -0.689281 -0.327642 -0.139097 -0.055353 -0.059752  378.66   
3      -0.190321 -1.175575  0.647376 -0.221929  0.062723  0.061458  123.50   
4      -0.137458  0.141267 -0.206010  0.502292  0.219422  0.215153   69.99   
...          ...       ...       ...       ...       ...       ...     ...   
284802  1.014480 -0.509348  1.436807  0.250034  0.943651  0.823731    0.77   
284803  0.012463 -1.016226 -0.606624 -0.395255  0.068472 -0.053527   24.79   
284804 -0.037501  0.640134  0.265745 -0.087371  0.004455 -0.026561   67.88   
284805 -0.163298  0.123205 -0.569159  0.546668  0.108821  0.104533   10.00   
284806  0.376777  0.008797 -0.473649 -0.818267 -0.002415  0.013649  217.00   

        Class  
0           0  
1           0  
2           0  
3           0  
4           0  
...       ...  
284802      0  
284803      0  
284804      0  
284805      0  
284806      0  

[284807 rows x 31 columns]
```

In [63]:
```python
data.describe()
```

Out[63]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	...	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05
mean	94813.859575	1.168375e-15	3.416908e-16	-1.379537e-15	2.074095e-15	9.604066e-16	1.487313e-15	-5.556467e-16	1.213481e-16	-2.406331e-15	...	1.654067e-16	-3.568593e-16	2.578648e-16	4.473266e-15
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00	1.332271e+00	1.237094e+00	1.194353e+00	1.098632e+00	...	7.345240e-01	7.257016e-01	6.244603e-01	6.056471e-01
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-2.616051e+01	-4.355724e+01	-7.321672e+01	-1.343407e+01	...	-3.483038e+01	-1.093314e+01	-4.480774e+01	-2.836627e+00
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01	-7.682956e-01	-5.540759e-01	-2.086297e-01	-6.430976e-01	...	-2.283949e-01	-5.423504e-01	-1.618463e-01	-3.545861e-01
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02	-2.741871e-01	4.010308e-02	2.235804e-02	-5.142873e-02	...	-2.945017e-02	6.781943e-03	-1.119293e-02	4.097606e-02
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01	3.985649e-01	5.704361e-01	3.273459e-01	5.971390e-01	...	1.863772e-01	5.285536e-01	1.476421e-01	4.395266e-01
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01	7.330163e+01	1.205895e+02	2.000721e+01	1.559499e+01	...	2.720284e+01	1.050309e+01	2.252841e+01	4.584549e+00

8 rows × 31 columns

In [33]:
```python
data.dtypes
```

Out[33]:
```
Time      float64
V1        float64
V2        float64
V3        float64
V4        float64
V5        float64
V6        float64
V7        float64
V8        float64
V9        float64
V10       float64
V11       float64
V12       float64
V13       float64
V14       float64
V15       float64
V16       float64
V17       float64
V18       float64
V19       float64
V20       float64
V21       float64
V22       float64
V23       float64
V24       float64
V25       float64
V26       float64
V27       float64
V28       float64
Amount    float64
Class       int64
dtype: object
```

In [64]:
```python
data.nunique()
```

Out[64]:
```
Time      124592
V1        275663
V2        275663
V3        275663
V4        275663
V5        275663
V6        275663
V7        275663
V8        275663
V9        275663
V10       275663
V11       275663
V12       275663
V13       275663
V14       275663
V15       275663
V16       275663
V17       275663
V18       275663
V19       275663
V20       275663
V21       275663
V22       275663
V23       275663
V24       275663
V25       275663
V26       275663
V27       275663
V28       275663
Amount     32767
Class          2
dtype: int64
```

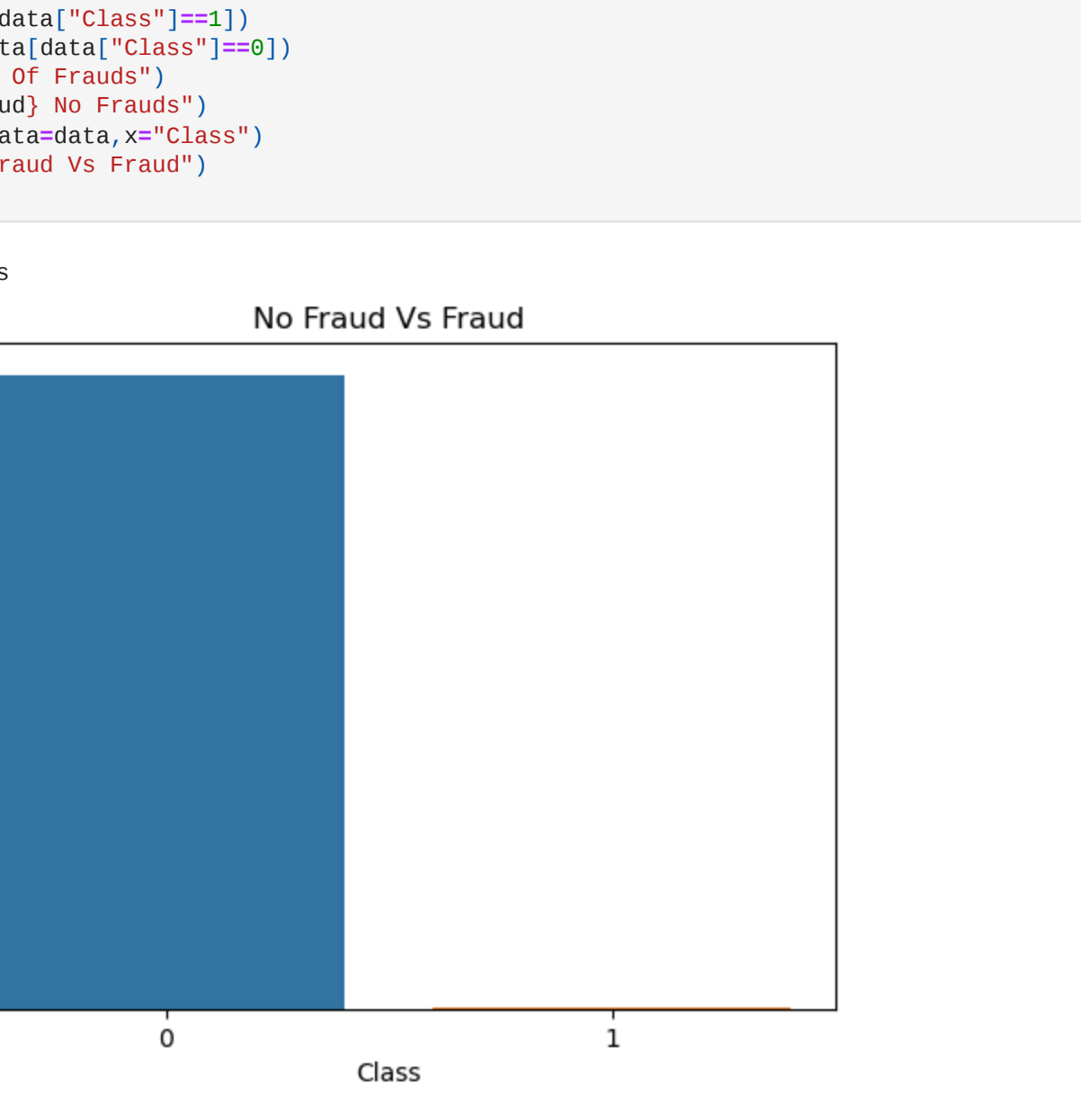
In [36]:
```python
data[["Time","Amount","Class"]].describe().T.style.background_gradient(subset="mean",axis=0,cmap="bwr",vmin=.5,vmax=1,low=.4,high=.8)\
                        .background_gradient(subset=["std","50%"],axis=0,cmap="bwr")\
                        .background_gradient(subset="max",axis=0,cmap="Blues_r",low=.2,high=.9)
```

Out[36]:

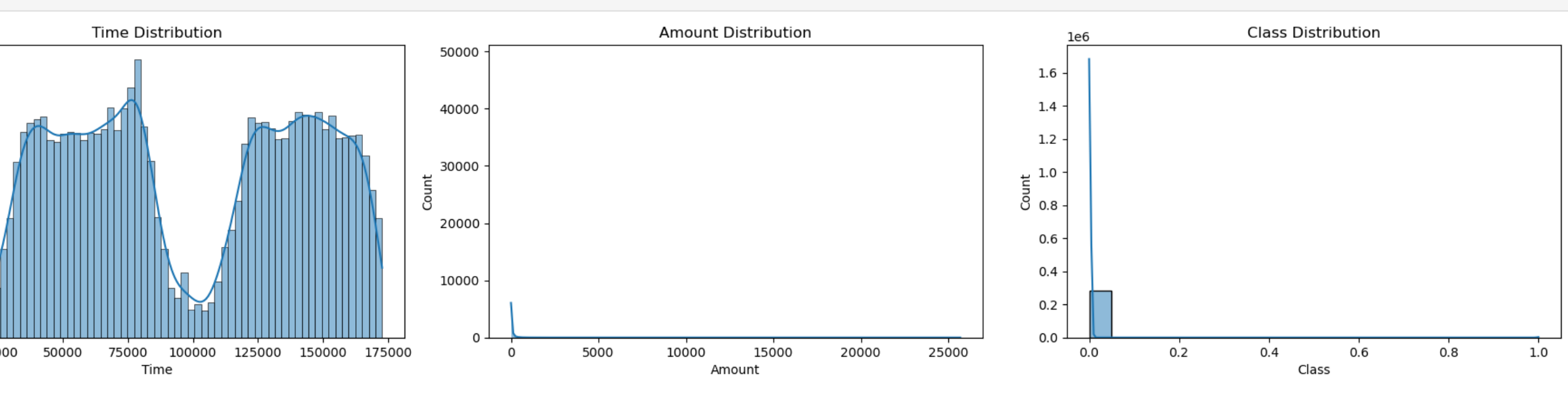
	count	mean	std	min	25%	50%	75%	max
Time	284807.000000	94813.859575	47488.145955	0.000000	54201.500000	84692.000000	139320.500000	172792.000000
Amount	284807.000000	88.349619	250.120109	0.000000	5.600000	22.000000	77.165000	25691.160000
Class	284807.000000	0.001727	0.041527	0.000000	0.000000	0.000000	0.000000	1.000000

In [37]:
```python
fraud=len(data[data["Class"]==1])
no_fraud=len(data[data["Class"]==0])
print(f"{fraud} Of Frauds")
print(f"{no_fraud} No Frauds")
sns.countplot(data=data,x="Class")
plt.title("No Fraud Vs Fraud")
plt.show()
```

```
492 Of Frauds
284315 No Frauds
```

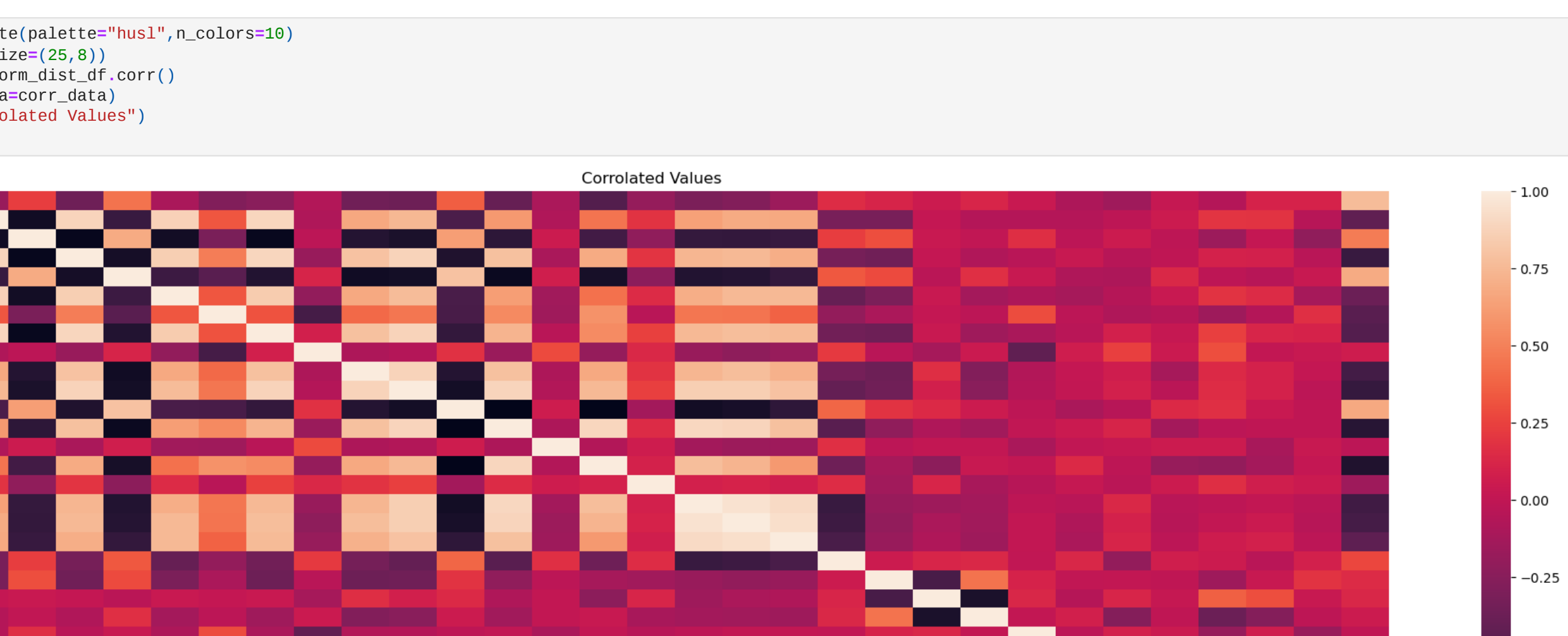


In [38]:
```python
fig,axs=plt.subplots(1,3,figsize=(19,4))
sns.histplot(data["Time"],ax=axs[0],kde=True)
axs[0].set_title("Time Distribution")

sns.histplot(data["Amount"],ax=axs[1],kde=True)
axs[1].set_title("Amount Distribution")

sns.histplot(data["Class"],ax=axs[2],kde=True)
axs[2].set_title("Class Distribution")
plt.tight_layout(pad=.5)
plt.show()
```

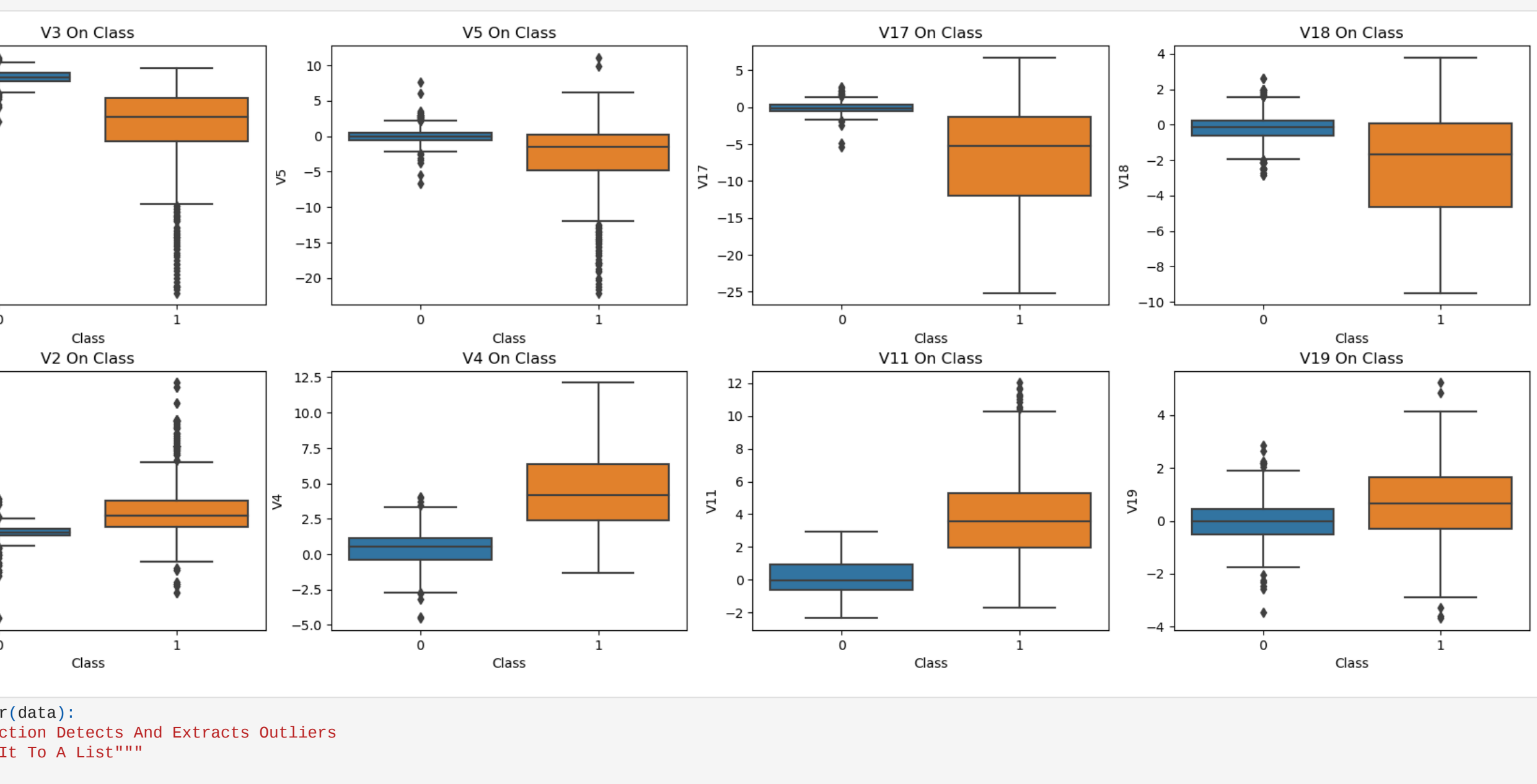


In [39]:
```python
# Do A random Sampling
fraud_df=data.loc[data["Class"]==0][:492]
no_fraud_df=data.loc[data["Class"]==1]
new_norm_dist_df=pd.concat([fraud_df,no_fraud_df])
sns.countplot(data=new_norm_dist_df,x="Class")
plt.title("No Fraud VS Fraud Balanced")
plt.show()
```

In [40]:
```python
sns.color_palette(palette="husl",n_colors=10)
plt.figure(figsize=(25,8))
corr_data=new_norm_dist_df.corr()
sns.heatmap(data=corr_data)
plt.title("Corrolated Values")
plt.show()
```



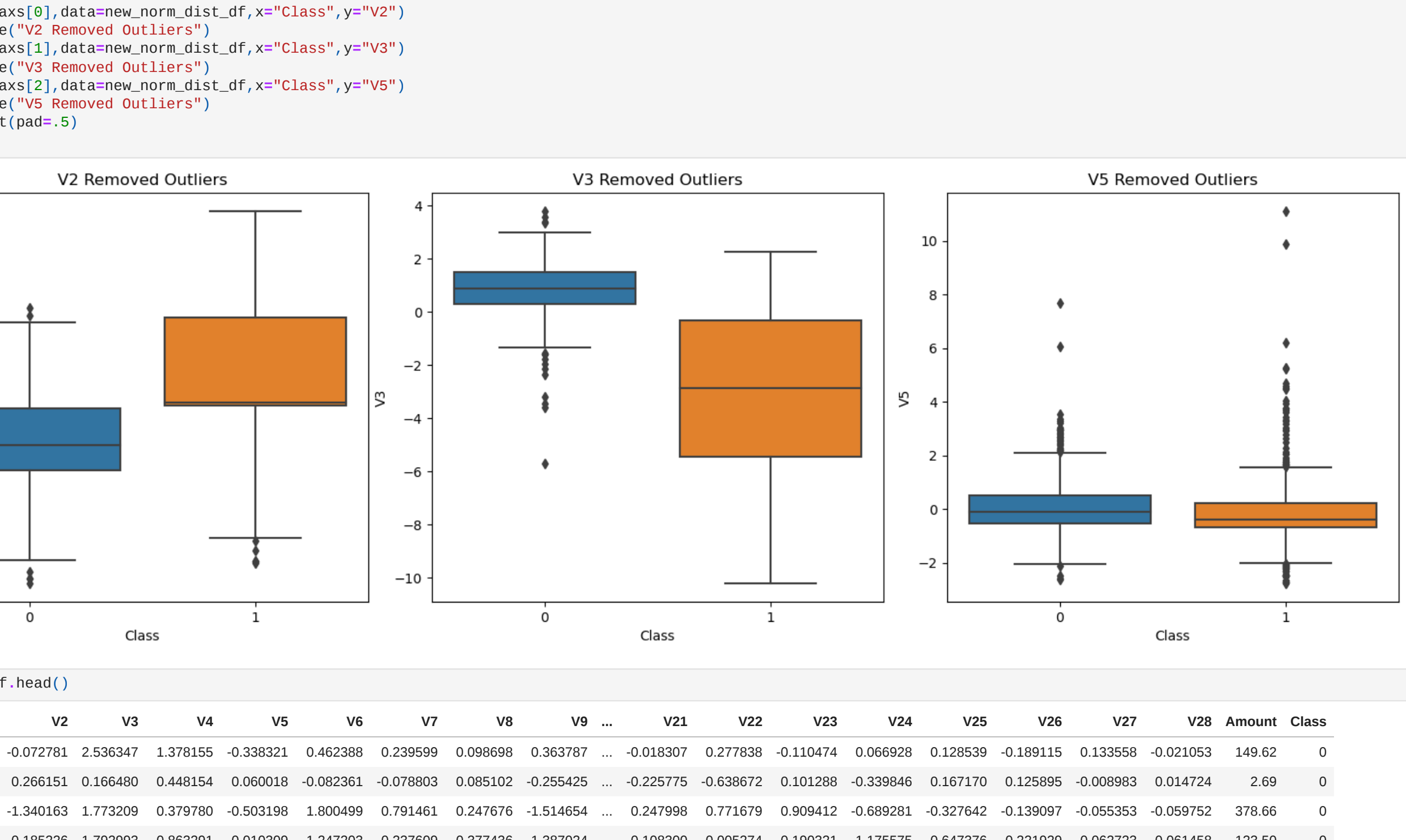
In [41]:
```python
negative_corr_cols=["V3","V5","V17","V18"]
positive_corr_cols=["V2","V4","V11","V19"]

fig,axs=plt.subplots(nrows=2,ncols=4,figsize=(18,7))
sns.boxplot(ax=axs[0,0],data=new_norm_dist_df,x="Class",y=negative_corr_cols[0])
axs[0,0].set_title("V3 On Class")

sns.boxplot(ax=axs[0,1],data=new_norm_dist_df,x="Class",y=negative_corr_cols[1])
axs[0,1].set_title("V5 On Class")

sns.boxplot(ax=axs[0,2],data=new_norm_dist_df,x="Class",y=negative_corr_cols[2])
axs[0,2].set_title("V17 On Class")

sns.boxplot(ax=axs[0,3],data=new_norm_dist_df,x="Class",y=negative_corr_cols[3])
axs[0,3].set_title("V18 On Class")

sns.boxplot(ax=axs[1,0],data=new_norm_dist_df,x="Class",y=positive_corr_cols[0])
axs[1,0].set_title("V2 On Class")

sns.boxplot(ax=axs[0,3],data=new_norm_dist_df,x="Class",y=negative_corr_cols[3])
axs[0,3].set_title("V18 On Class")

sns.boxplot(ax=axs[1,0],data=new_norm_dist_df,x="Class",y=positive_corr_cols[0])
axs[1,0].set_title("V2 On Class")

sns.boxplot(ax=axs[1,1],data=new_norm_dist_df,x="Class",y=positive_corr_cols[1])
axs[1,1].set_title("V4 On Class")

sns.boxplot(ax=axs[1,2],data=new_norm_dist_df,x="Class",y=positive_corr_cols[2])
axs[1,2].set_title("V11 On Class")

sns.boxplot(ax=axs[1,3],data=new_norm_dist_df,x="Class",y=positive_corr_cols[3])
axs[1,3].set_title("V19 On Class")

plt.tight_layout(pad=.5)
plt.show()
```



In [44]:
```python
def outliers_iqr(data):
    """This Function Detects And Extracts Outliers
    And Append It To A List"""

    outliers=[]
    const_iqr=1.5
    data=sorted(data)

    #Get The Percentiles 25% And 75%
    Q_1=np.percentile(data,25)
    Q_3=np.percentile(data,75)

    #Calculate IQR
    IQR=Q_3-Q_1
    lower_bound=Q_1-(IQR*const_iqr)
    upper_bound=Q_3+(IQR*const_iqr)

    #Iterate Over Data And Look For Outliers
    for _,d in enumerate(data):
        if(d<lower_bound or d>upper_bound):
            outliers.append(d)

    return outliers
```

In [56]:
```python
#The Percentiles of V2
V2_Q1,V2_Q3=np.percentile(new_norm_dist_df["V2"],25),\
            np.percentile(new_norm_dist_df["V2"],75)
#The Percentiles of V5
V5_Q1,V5_Q3=np.percentile(new_norm_dist_df["V5"],25)\
            ,np.percentile(new_norm_dist_df["V5"],75)
#The Percentiles of V3
V3_Q1,V3_Q3=np.percentile(new_norm_dist_df["V3"],25),\
            np.percentile(new_norm_dist_df["V3"],75)

#Calculate V2 And Replace Outliers With Median On V2 Column
IQR_V2=V2_Q3-V2_Q1
V2_lower=V2_Q1-(IQR_V2*1.5)
V2_upper=V2_Q3+(IQR_V2*1.5)
new_norm_dist_df["V2"]=np.where((new_norm_dist_df["V2"]>V2_upper)|(new_norm_dist_df["V2"]<V2_lower)\
                        ,np.median(new_norm_dist_df["V2"]),new_norm_dist_df["V2"])

#Calculate V5 And Replace Outliers With Median On V5 Column
IQR_V5=V5_Q3-V5_Q1
V2_lower=V2_Q1-(IQR_V2*1.5)
V2_upper=V2_Q3+(IQR_V2*1.5)
new_norm_dist_df["V2"]=np.where((new_norm_dist_df["V2"]>V2_upper)|(new_norm_dist_df["V2"]<V2_lower)\
                        ,np.median(new_norm_dist_df["V2"]),new_norm_dist_df["V2"])

#Calculate V5 And Replace Outliers With Median On V5 Column
IQR_V5=V5_Q3-V5_Q1
V5_lower=V5_Q1-(IQR_V5*1.5)
V5_upper=V5_Q3+(IQR_V5*1.5)
new_norm_dist_df["V5"]=np.where(new_norm_dist_df["V5"]<V5_lower\
                        ,np.median(new_norm_dist_df["V5"]),new_norm_dist_df["V5"])

#Calculate V3 And Replace Outliers With Median On V3 Column
IQR_V3=V3_Q3-V3_Q1
V3_lower=V3_Q1-(IQR_V3*1.5)
#V3_upper=V3_Q3+(IQR_V3*1.5)
new_norm_dist_df["V3"]=np.where(new_norm_dist_df["V3"]<V3_lower\
                        ,np.median(new_norm_dist_df["V3"]),new_norm_dist_df["V3"])
```

In [46]:
```python
fig,axs=plt.subplots(1,3,figsize=(16,5))

sns.boxplot(ax=axs[0],data=new_norm_dist_df,x="Class",y="V2")
axs[0].set_title("V2 Removed Outliers")
sns.boxplot(ax=axs[1],data=new_norm_dist_df,x="Class",y="V3")
axs[1].set_title("V3 Removed Outliers")
sns.boxplot(ax=axs[2],data=new_norm_dist_df,x="Class",y="V5")
axs[2].set_title("V5 Removed Outliers")
plt.tight_layout(pad=.5)
plt.show()
```



In [47]:
```python
new_norm_dist_df.head()
```

Out[47]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	149.62	0
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724	2.69	0
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	378.66	0
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.061458	123.50	0
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.215153	69.99	0

5 rows × 31 columns

In [48]:
```python
X=new_norm_dist_df.drop(columns="Class",axis=1).values
y=new_norm_dist_df["Class"].values
```

In [49]:
```python
models={
    "SVM":SVC(),
    "DecisionTree":DecisionTreeClassifier(random_state=42)
}
```

In [51]:
```python
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=.2,random_state=42)
```

In [52]:
```python
svm_parameters={
    "C":[0.1,1,10,100]
}
xgb_parameters={
    "max_depth":[3,5,7,9],
    "learning_rate":[0.1,0.01,0.001,0.0001]
}
```

In [53]:
```python
%%time
# Training Using Fine Tuned SVM
svm_grid=GridSearchCV(estimator=models["SVM"],param_grid=svm_parameters,cv=5)
svm_grid.fit(X_train,y_train)
print(f"The Best Cost Parameter Is{svm_grid.best_params_} And The Best Score Is {svm_grid.best_score_}")
```

```
The Best Cost Parameter Is{'C': 10} And The Best Score Is 0.9987261146496815
CPU times: total: 219 ms
Wall time: 298 ms
```

In [55]:
```python
%%time
#XGBoost
xgboost_classi=XGBClassifier(n_estimators=100)
xg_grid=GridSearchCV(estimator=xgboost_classi,param_grid=xgb_parameters,cv=5)
xg_grid.fit(X_train,y_train)
print(f"The Best Cost Parameter Is{xg_grid.best_params_} And The Best Score Is {xg_grid.best_score_}")
```

```
The Best Cost Parameter Is{'learning_rate': 0.1, 'max_depth': 3} And The Best Score Is 0.9987261146496815
CPU times: total: 26.7 s
Wall time: 25.3 s
```