In [105]:
```
import pandas as pd
import seaborn as sns
import numpy as np
import sklearn
import scipy
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report,accuracy_score
from sklearn.ensemble import IsolationForest
from sklearn.neighbors import LocalOutlierFactor
from sklearn.svm import OneClassSVM
from pylab import rcParams

from sklearn.model_selection import train_test_split
rcParams['figure.figsize']=14,8
RANDOM_SEED=42
LABELS=["Normal","Fraud"]
```

In [74]:
```
data=pd.read_csv("C:/Users/Rakesh Kumar/Desktop/creditcard.csv")
```

In [75]:
```
data.head()
```

Out[75]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533

5 rows × 31 columns

In [76]:
```
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 [77]:
```
#get the fraud and normal dataset
fraud=data.loc[data['Class']==1]
normal=data.loc[data['Class']==0]
```

In [78]:
```
print(fraud.shape,normal.shape)
```

```
(492, 31) (284315, 31)
```

In [79]:
```
##we need more information from transaction data
#how different are the amount of money used in different transaction classes
fraud.Amount.describe()
```

Out[79]:
```
count     492.000000
mean      122.211321
std       256.683288
min         0.000000
25%         1.000000
50%         9.250000
75%       105.890000
max      2125.870000
Name: Amount, dtype: float64
```

In [80]:
```
fraud.Amount.describe()
```

Out[80]:
```
count     492.000000
mean      122.211321
std       256.683288
min         0.000000
25%         1.000000
50%         9.250000
75%       105.890000
max      2125.870000
Name: Amount, dtype: float64
```

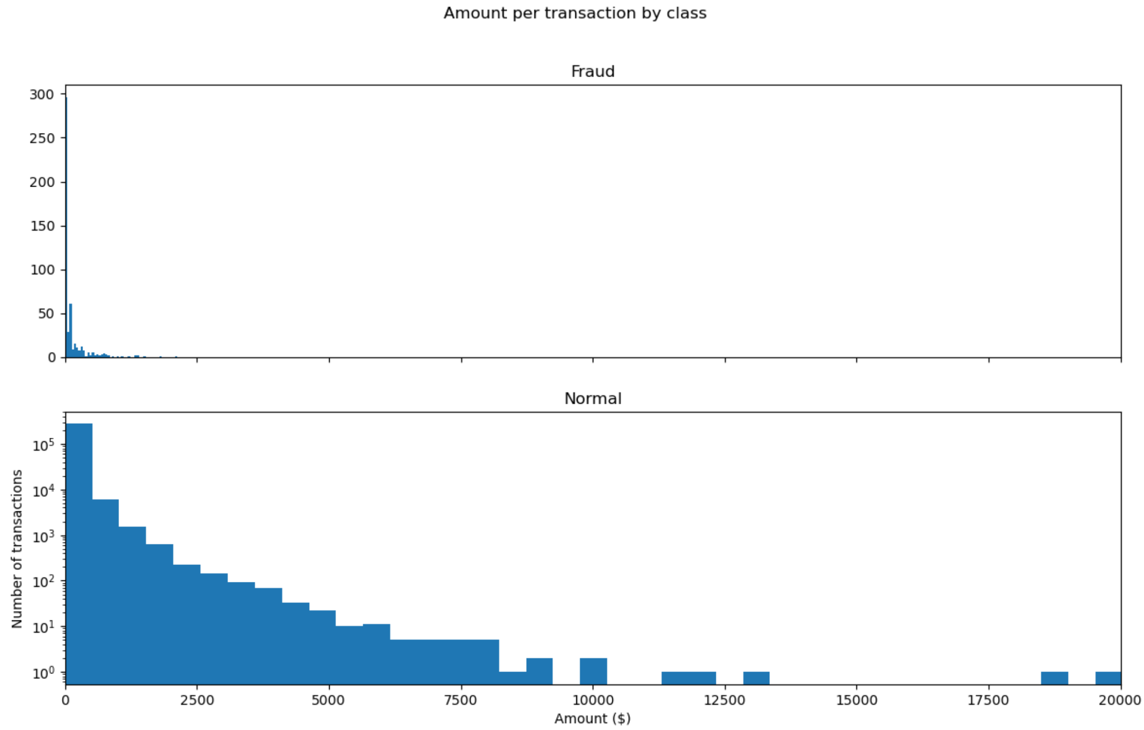
In [81]:
```
fraud
```

Out[81]:

	Time	V1	V2	V3	V4	V5	V6	V7
541	406.0	-2.312227	1.951992	-1.609851	3.997906	-0.522188	-1.426545	-2.537387
623	472.0	-3.043541	-3.157307	1.088463	2.288644	1.359805	-1.064823	0.325574
4920	4462.0	-2.303350	1.759247	-0.359745	2.330243	-0.821628	-0.075788	0.562320
6108	6986.0	-4.397974	1.358367	-2.592844	2.679787	-1.128131	-1.706536	-3.496197
6329	7519.0	1.234235	3.019740	-4.304597	4.732795	3.624201	-1.357746	1.713445
...	...	...	...	...	...	...	...	...
279863	169142.0	-1.927883	1.125653	-4.518331	1.749293	-1.566487	-2.010494	-0.882850
280143	169347.0	1.378559	1.289381	-5.004247	1.411850	0.442581	-1.326536	-1.413170
280149	169351.0	-0.676143	1.126366	-2.213700	0.468308	-1.120541	-0.003346	-2.234739
281144	169966.0	-3.113832	0.585864	-5.399730	1.817092	-0.840618	-2.943548	-2.208002
281674	170348.0	1.991976	0.158476	-2.583441	0.408670	1.151147	-0.096695	0.223050

492 rows × 31 columns

In [82]:
```
f,(ax1,ax2)=plt.subplots(2,1,sharex=True)
f.suptitle("Amount per transaction by class")
bins=50
ax1.hist(fraud.Amount,bins=bins)
ax1.set_title('Fraud')
ax2.hist(normal.Amount,bins=bins)
ax2.set_title('Normal')
plt.xlabel('Amount ($)')
plt.ylabel("Number of transactions")
plt.xlim(0,20000)
plt.yscale('log')
plt.show();
```



In [83]:
```
len(fraud)
```

Out[83]: 492

In [84]:
```
fraud.sum()
```

Out[84]:
```
Time      3.972743e+07
V1       -2.347799e+03
V2        1.782899e+03
V3       -3.460374e+03
V4        2.234678e+03
V5       -1.550403e+03
V6       -6.876865e+02
V7       -2.739816e+03
V8        2.807529e+02
V9       -1.269912e+03
V10      -2.793026e+03
V11       1.869685e+03
V12      -3.079621e+03
V13      -5.379224e+01
V14      -3.430088e+03
V15      -4.572094e+01
V16      -2.036853e+03
V17      -3.279592e+03
V18      -1.105184e+03
V19       3.348844e+02
V20       1.831811e+02
V21       3.510855e+02
V22       6.912050e+00
V23      -1.983152e+01
V24      -5.172411e+01
V25       2.039285e+01
V26       2.541088e+01
V27       8.392280e+01
V28       3.722831e+01
Amount    6.012797e+04
Class     4.920000e+02
dtype: float64
```

In [85]:
```
len(normal)
```

Out[85]:
```
284315
```

In [86]:
```
X=data.iloc[:,:-1]
y=data['Class']
```

In [87]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.35)
```

In [88]:
```
clf=linear_model.LogisticRegression(C=1e5)
```

In [89]:
```
clf.fit(X_train,y_train)
```

```
C:\Users\Rakesh Kumar\anaconda3\anaconda\lib\site-packages\sklearn\linear_
model\_logistic.py:814: ConvergenceWarning: lbfgs failed to converge (stat
us=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://sc
ikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-reg
ression (https://scikit-learn.org/stable/modules/linear_model.html#logisti
c-regression)
  n_iter_i = _check_optimize_result(
```

Out[89]:
```
LogisticRegression(C=100000.0)
```

In [90]:
```
y_pred=np.array(clf.predict(X_test))
y=np.array(y_test)
```

In [91]:
```
from sklearn.metrics import confusion_matrix,classification_report,accuracy_
```

In [92]:
```
print(confusion_matrix(y,y_pred))
```

```
[[99444    48]
 [   63   128]]
```

In [93]:
```
print(accuracy_score(y,y_pred))
```

```
0.9988864701102494
```

In [94]:
```
print(classification_report(y,y_pred))
```

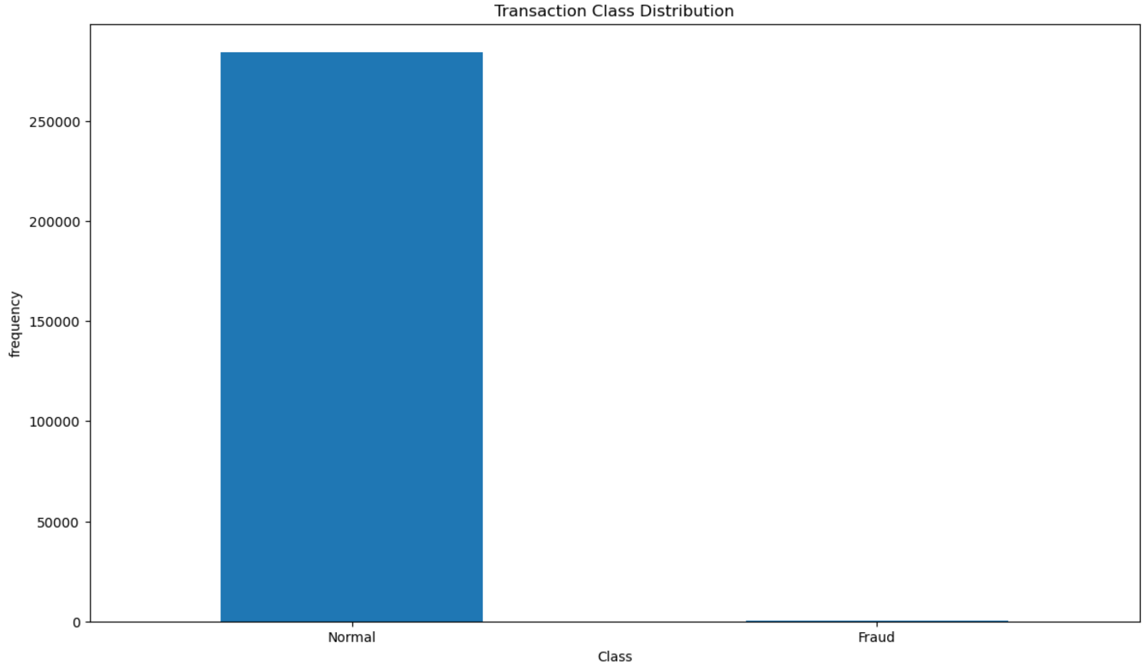
```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     99492
           1       0.73      0.67      0.70       191

    accuracy                           1.00     99683
   macro avg       0.86      0.83      0.85     99683
weighted avg       1.00      1.00      1.00     99683
```

In [95]:
```
#EXPLORATORY DATA ANALYSIS
data.isnull().values.any()
```

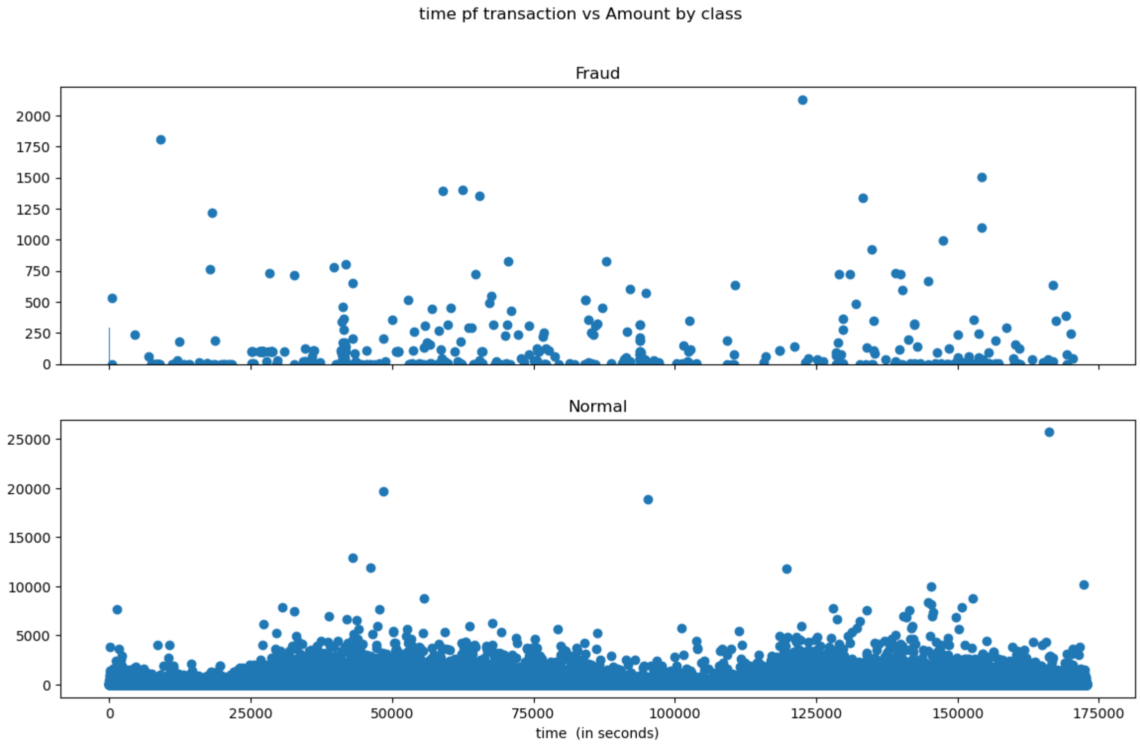
Out[95]:
```
False
```

In [96]:
```
count_classes=pd.value_counts(data['Class'],sort=True)
count_classes.plot(kind='bar',rot=0)
plt.title("Transaction Class Distribution")
plt.xticks(range(2),LABELS)
plt.xlabel("Class")
plt.ylabel("frequency")
```

Out[96]:
```
Text(0, 0.5, 'frequency')
```



In [97]:
```
# we will check do fraudulent transactions occur more often during certain t
f, (ax1,ax2)=plt.subplots(2,1,sharex=True)

f.suptitle("time pf transaction vs Amount by class")
ax1.scatter(fraud.Time,fraud.Amount)
ax1.hist(fraud.Amount,bins=bins)
ax1.set_title('Fraud')
ax2.scatter(normal.Time,normal.Amount)
ax2.set_title('Normal')
plt.xlabel('time  (in seconds)')


plt.show();
```



In [98]:
```
data1=data.sample(frac=0.1,random_state=1)
data1.shape
```

Out[98]: (28481, 31)

In [99]:
```
data.shape
```

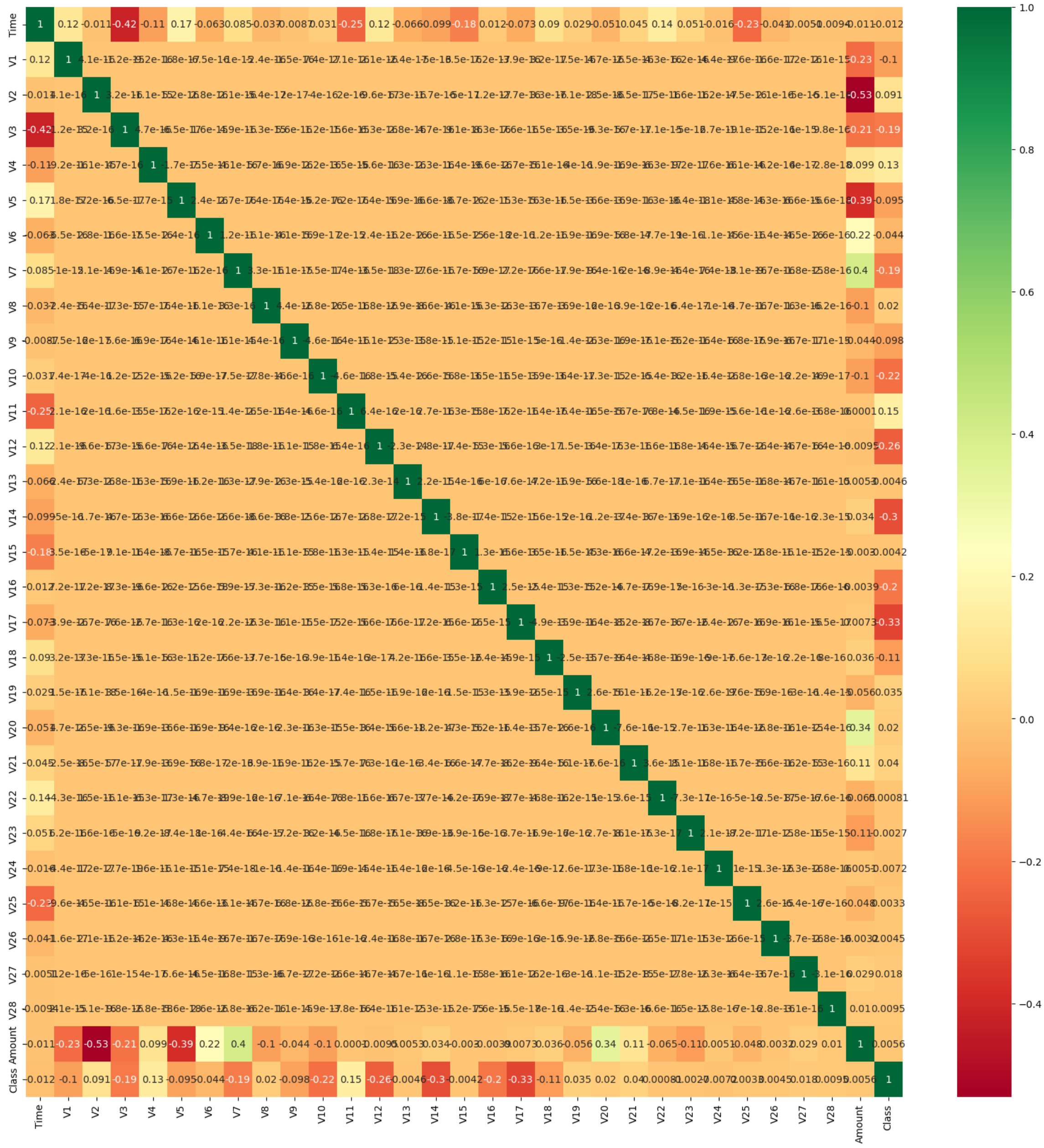
Out[99]: (284807, 31)

In [100]:
```
#Determine the number of fraud and valid transactions in the dataset
Fraud=data1[data1['Class']==1]
Valid=data1[data1['Class']==0]
outlier_fraction=len(Fraud)/float(len(Valid))
```

In [101]:
```
print(outlier_fraction)
print("Fraud Cases:{}".format(len(Fraud)))
print("Valid Cases:{}".format(len(Valid)))
```

```
0.0017234102419808666
Fraud Cases:49
Valid Cases:28432
```

In [102]:
```
##correlation
import seaborn as sns
cormat=data1.corr()
top_corr_features=cormat.index
plt.figure(figsize=(20,20))
g=sns.heatmap(data[top_corr_features].corr(),annot=True,cmap="RdYlGn")
```



In [103]:
```
#create dependent and independent features
columns=data.columns.tolist()
columns=[c for c in columns if c not in ["Class"]]
target="Class"
state=np.random.RandomState(42)
X=data1[columns]
Y=data1[target]
X_outliers=state.uniform(low=0,high=1,size=(X.shape[0],X.shape[1]))
print(X.shape)
print(Y.shape)
```

```
(28481, 30)
(28481,)
```

In [116]:
```
from sklearn.svm import OneClassSVM
from sklearn.ensemble import IsolationForest
from sklearn.neighbors import LocalOutlierFactor

classifiers = {
    "Isolation Forest": IsolationForest(n_estimators=100, max_samples=len(X
                                        contamination=outlier_fraction, verb
    "Local Outlier Factor": LocalOutlierFactor(n_neighbors=20, algorithm='au
                                               contamination=outlier_fractio
    "Support Vector Machine": OneClassSVM(kernel='rbf', degree=3, gamma=0.1
}
```

In [117]:
```
type(classifiers)
```

Out[117]:
```
dict
```

In [*]:
```
n_outliers=len(Fraud)
for i,(clf_name,clf) in enumerate(classifiers.items()):
    if clf_name=="Local Outlier Factor":
        y_pred==clf.fit_predict(X)
        scores_prediction=clf.negative_outlier_factor_
    elif clf_name=="Support Vector Machine":
        clf.fit(X)
        y_pred=clf.predict(X)
    else:
        clf.fit(X)
        scores_prediction=clf.decision_function(X)
        y_pred=clf.predict(X)

y_pred[y_pred==1]=0
y_pred[y_pred==-1]=1
n_errors=(y_pred!=Y).sum()

print("{}:{}".format(clf_name,n_errors))
print("Accuracy Score :")
print(accuracy_score(Y,y_pred))
print("Classification Report :")
print(classification_report(Y,y_pred))
```

In [ ]:

In [ ]:

In [ ]: