In [69]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

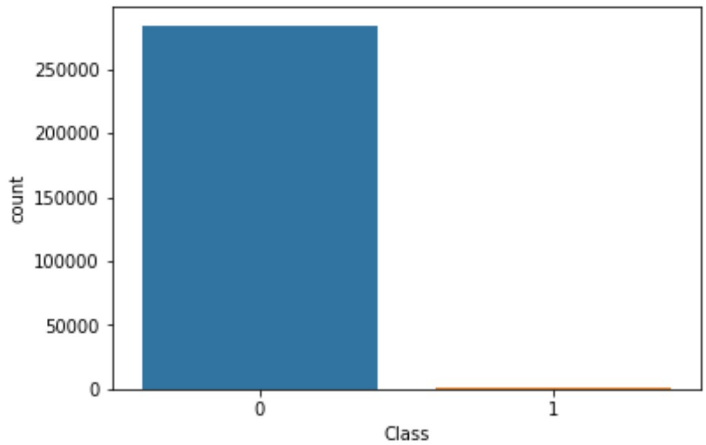
In [70]:
```
data = pd.read_csv("creditcard.csv")
```

In [71]:
```
print(data.head())
print(data.describe())
print(data.shape)
```

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

         V8        V9  ...       V21       V22       V23       V24       V25  \
0  0.098698  0.363787  ... -0.018307  0.277838 -0.110474  0.066928  0.128539
1  0.085102 -0.255425  ... -0.225775 -0.638672  0.101288 -0.339846  0.167170
2  0.247676 -1.514654  ...  0.247998  0.771679  0.909412 -0.689281 -0.327642
3  0.377436 -1.387024  ... -0.108300  0.005274 -0.190321 -1.175575  0.647376
4 -0.270533  0.817739  ... -0.009431  0.798278 -0.137458  0.141267 -0.206010

        V26       V27       V28  Amount  Class
0 -0.189115  0.133558 -0.021053  149.62      0
1  0.125895 -0.008983  0.014724    2.69      0
2 -0.139097 -0.055353 -0.059752  378.66      0
3 -0.221929  0.062723  0.061458  123.50      0
4  0.502292  0.219422  0.215153   69.99      0

[5 rows x 31 columns]
                Time            V1            V2            V3            V4  \
count  284807.000000  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean    94813.859575  1.165980e-15  3.416908e-16 -1.373150e-15  2.086869e-15
std     47488.145955  1.958696e+00  1.651309e+00  1.516255e+00  1.415869e+00
min         0.000000 -5.640751e+01 -7.271573e+01 -4.832559e+01 -5.683171e+00
25%     54201.500000 -9.203734e-01 -5.985499e-01 -8.903648e-01 -8.486401e-01
50%     84692.000000  1.810880e-02  6.548556e-02  1.798463e-01 -1.984653e-02
75%    139320.500000  1.315642e+00  8.037239e-01  1.027196e+00  7.433413e-01
max    172792.000000  2.454930e+00  2.205773e+01  9.382558e+00  1.687534e+01

                 V5            V6            V7            V8            V9  \
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean   9.604066e-16  1.490107e-15 -5.556467e-16  1.177556e-16 -2.406455e-15
std    1.380247e+00  1.332271e+00  1.237094e+00  1.194353e+00  1.098632e+00
min   -1.137433e+02 -2.616051e+01 -4.355724e+01 -7.321672e+01 -1.343407e+01
25%   -6.915971e-01 -7.682956e-01 -5.540759e-01 -2.086297e-01 -6.430976e-01
50%   -5.433583e-02 -2.741871e-01  4.010308e-02  2.235804e-02 -5.142873e-02
75%    6.119264e-01  3.985649e-01  5.704361e-01  3.273459e-01  5.971390e-01
max    3.480167e+01  7.330163e+01  1.205895e+02  2.000721e+01  1.559499e+01

       ...           V21           V22           V23           V24  \
count  ...  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean   ...  1.656562e-16 -3.444850e-16  2.578648e-16  4.471968e-15
std    ...  7.345240e-01  7.257016e-01  6.244603e-01  6.056471e-01
min    ... -3.483038e+01 -1.093314e+01 -4.480774e+01 -2.836627e+00
25%    ... -2.283949e-01 -5.423504e-01 -1.618463e-01 -3.545861e-01
50%    ... -2.945017e-02  6.781943e-03 -1.119293e-02  4.097606e-02
75%    ...  1.863772e-01  5.285536e-01  1.476421e-01  4.395266e-01
max    ...  2.720284e+01  1.050309e+01  2.252841e+01  4.584549e+00

                V25           V26           V27           V28         Amount  \
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  284807.000000
mean   5.340915e-16  1.687098e-15 -3.666453e-16 -1.220404e-16      88.349619
std    5.212781e-01  4.822270e-01  4.036325e-01  3.300833e-01     250.120109
min   -1.029540e+01 -2.604551e+00 -2.256568e+01 -1.543008e+01       0.000000
25%   -3.171451e-01 -3.269839e-01 -7.083953e-02 -5.295979e-02       5.600000
50%    1.659350e-02 -5.213911e-02  1.342146e-03  1.124383e-02      22.000000
75%    3.507156e-01  2.409522e-01  9.104512e-02  7.827995e-02      77.165000
max    7.519589e+00  3.517346e+00  3.161220e+01  3.384781e+01   25691.160000

               Class
count  284807.000000
```

Header Column Class represents the state of transaction (1 = fraudulent, 0 = valid)

In [72]:
```
sns.countplot(data['Class'])
plt.show()
```



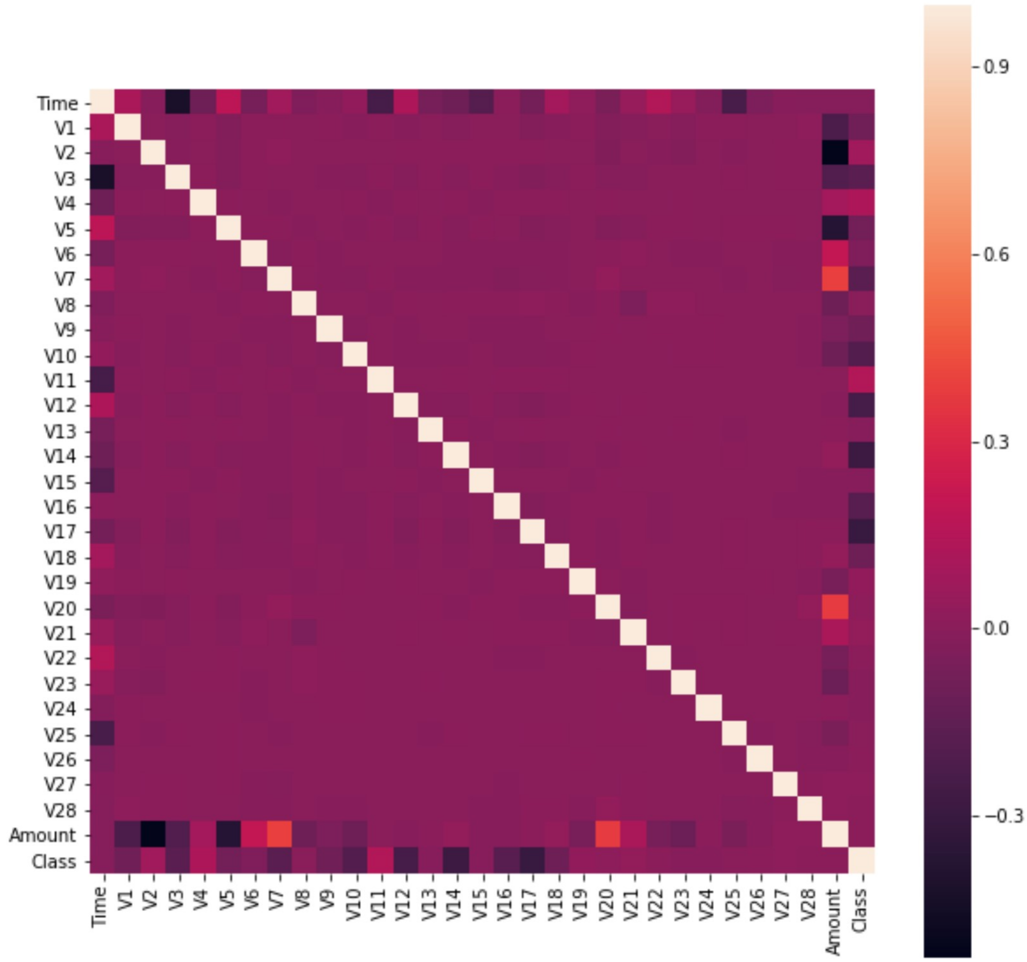
The average for the Class Column head is 0.001727 meaning that a very low amount of transactions in this dataset are considered fraudulent. This could mean using an outlier focused classifier or model would be theoretically effective.

In [73]:
```
data = data.sample(frac = 0.50, random_state = 1)
print(data.shape)
```

```
(142404, 31)
```

A correlation matrix might show us some features to focus on and perhaps increase accuracy and improve speed of classification

In [74]:
```
correlation_matrix = data.corr()
fig = plt.figure(figsize = (10,10))
sns.heatmap(correlation_matrix,square = True)
plt.show()
```



This shows that columns V10,V12,V14,V17 are not that correlated with the state of the transaction

In [75]:
```
data = data.drop(['V10','V12','V14','V17'],1)
```

In [76]:
```
fraud = data[data['Class']==1]
valid = data[data['Class']==0]

percent_fraud = len(fraud)/len(valid)
print(percent_fraud)
```

```
0.0015966014193575613
```

In [77]:
```
columns = data.columns.tolist()

target = "Class"

columns = [c for c in columns if c not in [target]]
print(columns)

X = data[columns]
Y = data[target]

from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size = .5)
print(Y_train.shape,Y_test.shape)

print(X.shape,Y.shape)
```

```
['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V11', 'V13', 'V15
', 'V16', 'V18', 'V19', 'V20', 'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', '
V28', 'Amount']
(71202,) (71202,)
(142404, 26) (142404,)
```

Preprocessing Complete!

In [78]:
```
from sklearn.metrics import classification_report, accuracy_score
from sklearn.ensemble import IsolationForest
```

Lets try an Isolation Forest Classifier as this could be effective when dealing with outlier cases

In [64]:
```
import warnings
warnings.filterwarnings("ignore")
clf = IsolationForest(max_samples=len(X_train))
clf.fit(X_train)
y_pred = clf.predict(X_test)
y_pred[y_pred==1] = 0
y_pred[y_pred==-1] = 1
```

In [65]:
```
print(accuracy_score(y_pred,Y_test))
```

```
0.9022920704474593
```

This is not bad but perhaps tweaking the contamination parameter, which represents the proportion of outliers in the dataset, will improve the accuracy of the model. Assigning it to the percentage of fraudulent transactions would be appropriate in this case as this represents the proportion of anomalies.

In [66]:
```
clf = IsolationForest(contamination = percent_fraud, max_samples = len(X_train))
clf.fit(X_train)
y_pred = clf.predict(X_test)
y_pred[y_pred==1] = 0
y_pred[y_pred==-1] = 1
```

In [67]:
```
print(accuracy_score(y_pred,Y_test))
```

```
0.9975281593213674
```

In [68]:
```
from sklearn.metrics import classification_report
print(classification_report(y_pred,Y_test))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     71091
           1       0.20      0.19      0.19       111

   micro avg       1.00      1.00      1.00     71202
   macro avg       0.60      0.59      0.60     71202
weighted avg       1.00      1.00      1.00     71202
```