Context

- It is important that credit card companies are able to recognize fraudulent credit card transactions so that customers are not charged for items that they did not purchase.

Content

- The dataset contains transactions made by credit cards in September 2013 by European cardholders.
- This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions.
- It contains only numerical input variables which are the result of a PCA transformation. Unfortunately, due to confidentiality issues, we cannot provide the original features and more background information about the data. Features V1, V2, … V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount'. Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature 'Amount' is the transaction Amount, this feature can be used for example-dependant cost-sensitive learning. Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise.
- Given the class imbalance ratio, we recommend measuring the accuracy using the Area Under the Precision-Recall Curve (AUPRC). Confusion matrix accuracy is not meaningful for unbalanced classification.

- Note: The above points are consider from the kaggle

Importing the Required Libraries

In [52]:
```
import pandas as pd
import numpy as np
%matplotlib inline
import seaborn as sns
import matplotlib.pyplot as plt


from sklearn.ensemble import RandomForestClassifier

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import KFold, cross_val_score
from sklearn.metrics import confusion_matrix,precision_recall_curve,f1_score,accu


from sklearn.model_selection import train_test_split

from imblearn.over_sampling import SMOTE
```

Loading the dataset

In [2]:
```
df= pd.read_csv("creditcard.csv")
```

In [3]:
```
""" lets find out the shape of the dataset """
```

Out[3]:
```
' lets find out the shape of the dataset '
```

In [4]:
```
print("The shape of the dataset is ",df.shape)
```

```
The shape of the dataset is  (284807, 31)
```

In [5]:
```
df
```

Out[5]:

	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	0.0
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.0
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.2
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.3
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.2
...	...	...	...	...	...	...	...	...	
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.3
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.2
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.7
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.6
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.4

284807 rows × 31 columns

In [6]:
```
pd.set_option('display.max_columns', 50)
```

In [7]:
```
df
```

Out[7]:

	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	0.0
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.0
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.2
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.3
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.2
...	...	...	...	...	...	...	...	...	
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.3
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.2
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.7
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.6
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.4

284807 rows × 31 columns

In [8]:
```
df.columns
```

Out[8]:
```
Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10',
       'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20',
       'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount',
       'Class'],
      dtype='object')
```

- Here we can observe that
- features are v1-v28 and time , amount , class

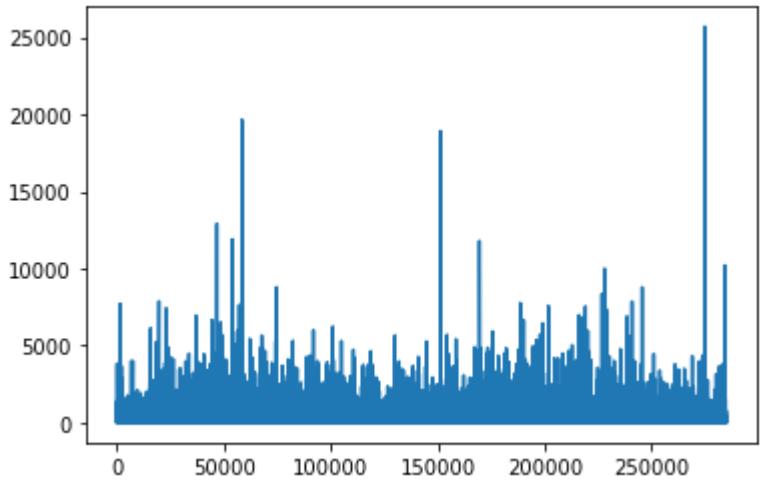
In [9]:
```
df.describe()
```

Out[9]:

	Time	V1	V2	V3	V4	V5
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05
mean	94813.859575	3.918649e-15	5.682686e-16	-8.761736e-15	2.811118e-15	-1.552103e-15
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01

In [10]:
```
df["Amount"].plot()
```

Out[10]: <AxesSubplot:>



In [11]:
```
df.isnull().sum().max()
```

Out[11]: 0

- no null values

In [13]:
```
df.corr()
```

Out[13]:

	Time	V1	V2	V3	V4	V5	
Time	1.000000	1.173963e-01	-1.059333e-02	-4.196182e-01	-1.052602e-01	1.730721e-01	-6.30
V1	0.117396	1.000000e+00	4.135835e-16	-1.227819e-15	-9.215150e-16	1.812612e-17	-6.50
V2	-0.010593	4.135835e-16	1.000000e+00	3.243764e-16	-1.121065e-15	5.157519e-16	2.7873
V3	-0.419618	-1.227819e-15	3.243764e-16	1.000000e+00	4.711293e-16	-6.539009e-17	1.6276
V4	-0.105260	-9.215150e-16	-1.121065e-15	4.711293e-16	1.000000e+00	-1.719944e-15	-7.49
V5	0.173072	1.812612e-17	5.157519e-16	-6.539009e-17	-1.719944e-15	1.000000e+00	2.4083
V6	-0.063016	-6.506567e-16	2.787346e-16	1.627627e-15	-7.491959e-16	2.408382e-16	1.0000
V7	0.084714	-1.005191e-15	2.055934e-16	4.895305e-16	-4.104503e-16	2.715541e-16	1.1916
V8	-0.036949	-2.433822e-16	-5.377041e-17	-1.268779e-15	5.697192e-16	7.437229e-16	-1.10
V9	-0.008660	-1.513678e-16	1.978488e-17	5.568367e-16	6.923247e-16	7.391702e-16	4.1312
V10	0.030617	7.388135e-17	-3.991394e-16	1.156587e-15	2.232685e-16	-5.202306e-16	5.9322
V11	-0.247689	2.125498e-16	1.975426e-16	1.576830e-15	3.459380e-16	7.203963e-16	1.9805
V12	0.124348	2.053457e-16	-9.568710e-17	6.310231e-16	-5.625518e-16	7.412552e-16	2.3754
V13	-0.065902	-2.425603e-17	6.295388e-16	2.807652e-16	1.303306e-16	5.886991e-16	-1.2111
V14	-0.098757	-5.020280e-16	-1.730566e-16	4.739859e-16	2.282280e-16	6.565143e-16	2.6213
V15	-0.183453	3.547782e-16	-4.995814e-17	9.068793e-16	1.377649e-16	-8.720275e-16	-1.5311
V16	0.011903	7.212815e-17	1.177316e-17	8.299445e-16	-9.614528e-16	2.246261e-15	2.6236
V17	-0.073297	-3.879840e-16	-2.685296e-16	7.614712e-16	-2.699612e-16	1.281914e-16	2.0156
V18	0.090438	3.230206e-17	3.284605e-16	1.509897e-16	-5.103644e-16	5.308590e-16	1.2238
V19	0.028975	1.502024e-16	-7.118719e-18	3.463522e-16	-3.980557e-16	-1.450421e-16	-1.86
V20	-0.050866	4.654551e-16	2.506675e-16	-9.316409e-16	-1.857247e-16	-3.554057e-16	-1.85
V21	0.044736	-2.457409e-16	-8.480447e-17	5.706192e-17	-1.949553e-16	-3.920976e-16	5.8333

	Time	V1	V2	V3	V4	V5	
V22	0.144059	-4.290944e-16	1.526333e-16	-1.133902e-15	-6.276051e-17	1.253751e-16	-4.70
V23	0.051142	6.168652e-16	1.634231e-16	-4.983035e-16	9.164206e-17	-8.428683e-18	1.0467
V24	-0.016182	-4.425156e-17	1.247925e-17	2.686834e-19	1.584638e-16	-1.149255e-15	-1.07
V25	-0.233083	-9.605737e-16	-4.478846e-16	-1.104734e-15	6.070716e-16	4.808532e-16	4.5628
V26	-0.041407	-1.581290e-17	2.057310e-16	-1.238062e-16	-4.247268e-16	4.319541e-16	-1.35
V27	-0.005135	1.198124e-16	-4.966953e-16	1.045747e-15	3.977061e-17	6.590482e-16	-4.45
V28	-0.009413	2.083082e-15	-5.093836e-16	9.775546e-16	-2.761403e-18	-5.613951e-18	2.5947
Amount	-0.010596	-2.277087e-01	-5.314089e-01	-2.108805e-01	9.873167e-02	-3.863563e-01	2.1598
Class	-0.012323	-1.013473e-01	9.128865e-02	-1.929608e-01	1.334475e-01	-9.497430e-02	-4.36

In [14]:
```
## lets try some seaborn tools for quick visulisation
```

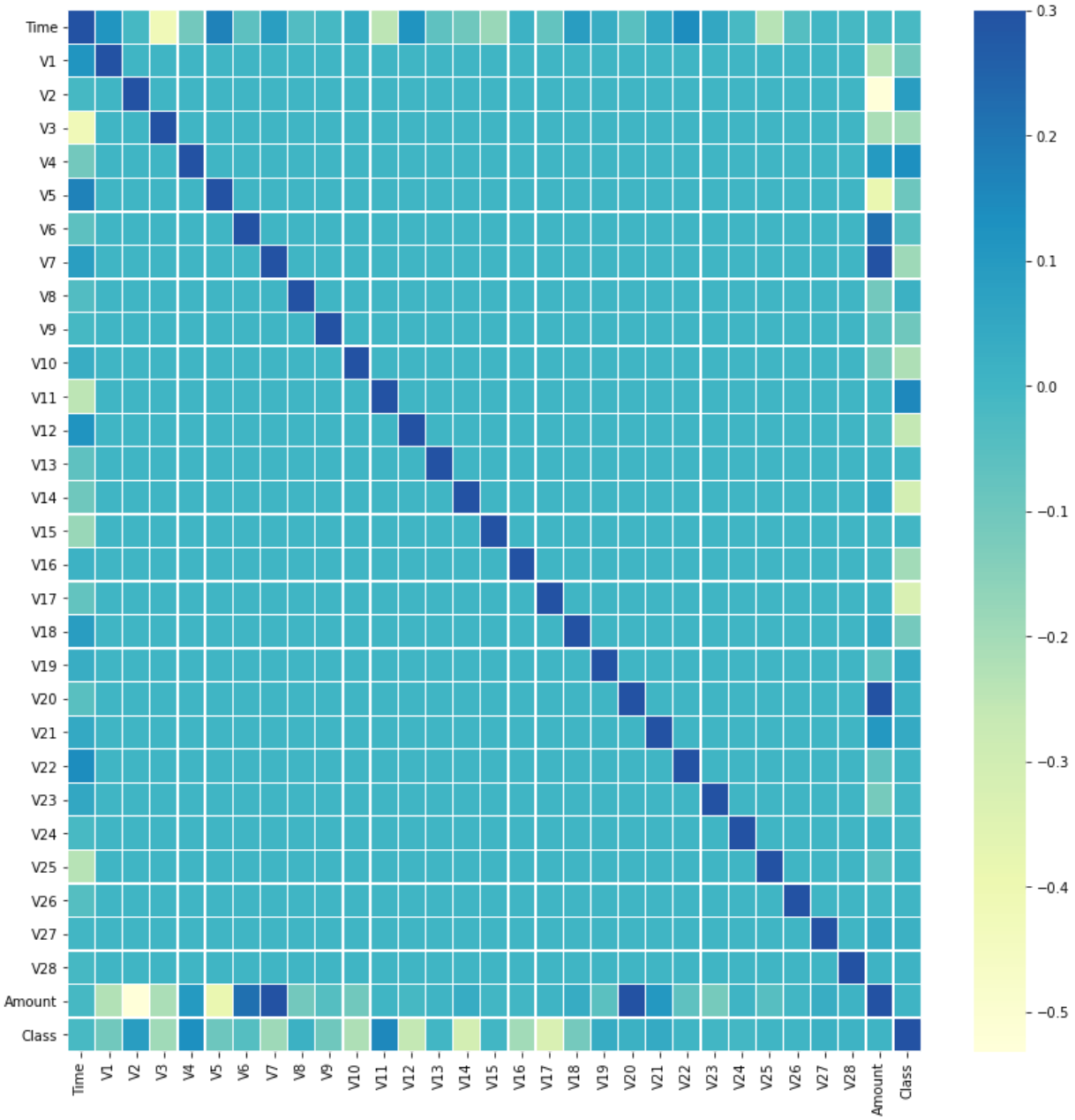
In [15]:
```
corr = df.corr()

# Generate a mask for the upper triangle


# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(14, 14))


# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr, cmap="YlGnBu", vmax=.3, center=0,linewidths=.3 )
```

Out[15]: <AxesSubplot:>



- since we are having large no of columns it is difficult to get some conclusion what are the fr=eature will be important
- For time being let consider all the variables and go ahead

Lets find the distibution of the target class

In [16]:
```
print("Non Frauds transcations", round(df['Class'].value_counts()[0]/len(df) * 10
print("Frauds transcations", round(df['Class'].value_counts()[1]/len(df) * 100,2)
```

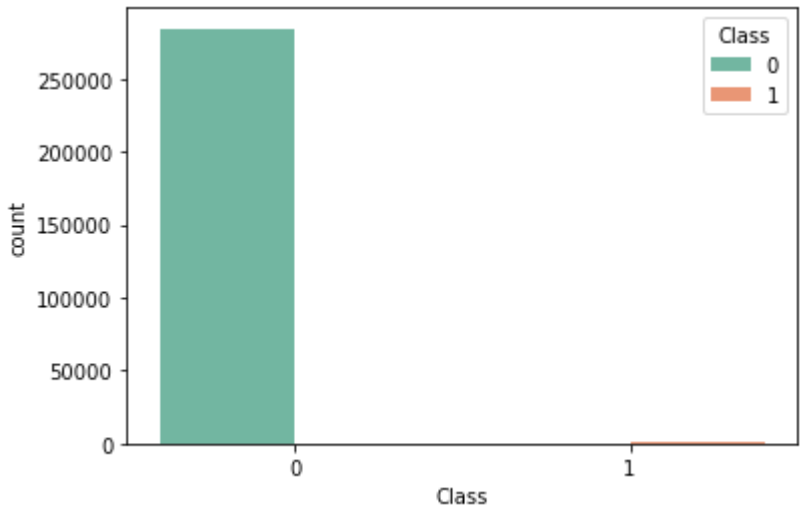
```
Non Frauds transcations 99.83 % of the dataset.
Frauds transcations 0.17 % of the dataset.
```

In [75]:
```
print("Non Frauds transcations", df['Class'].value_counts()[0], "% of the dataset
print("Non Frauds transcations", df['Class'].value_counts()[1], "% of the dataset
```

```
Non Frauds transcations 284315 % of the dataset.
Non Frauds transcations 492 % of the dataset.
```

- As previously metioned about the dataset it is true . it is highly imbalanced dataset

In [17]:
```
ax = sns.countplot(x='Class',  data=df, palette="Set2",hue="Class")
```



- here we will get two possible way to handle the imbalance data set
- 1st if we simply apply the ml model over here will get higher accuracy and the model is baised to non fraud transaction .
- 2nd we need to handle the imbalance dataset like using the upsampling ,smote and boostig techineques

Lets try 1st method

- Before starting it lets split the data set to x and y(target)

In [26]:
```
x = df.iloc[:, df.columns != 'Class']
y = df.iloc[:, df.columns == 'Class']
```

- let split the data set into train and test with 75:25 split ratio

In [31]:
```
x_tr, x_te, y_tr, y_te = train_test_split(x,y,test_size = 0.25, random_state = 32

print("Number transactions train dataset: ", len(x_tr))
print("Number transactions test dataset: ", len(x_te))
print("Total number of transactions: ", len(x))
```

```
Number transactions train dataset:  213605
Number transactions test dataset:  71202
Total number of transactions:  284807
```

In [42]:
```
#intilizeing the clasifier
rfc = RandomForestClassifier()

# fit the data into the classifer
rfc.fit(x_tr, y_tr)

# predict the model
rfc_predict = rfc.predict(x_te)# check performance
```

```
<ipython-input-42-90fe5de24f17>:6: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y to (n_sample
s,), for example using ravel().
  rfc.fit(x_tr, y_tr)
```

* ROC-AUC Score

In [43]:
```
print('ROC - AUC score:',roc_auc_score(y_te, rfc_predict))
```

```
ROCAUC score: 0.8856579968397817
```

* Accuracy Score

In [44]:
```
print('Accuracy score:',accuracy_score(y_te, rfc_predict))
```

```
Accuracy score: 0.9994382180275835
```

* F1 Score

In [45]:
```
print('F1 score:',f1_score(y_te, rfc_predict))
```

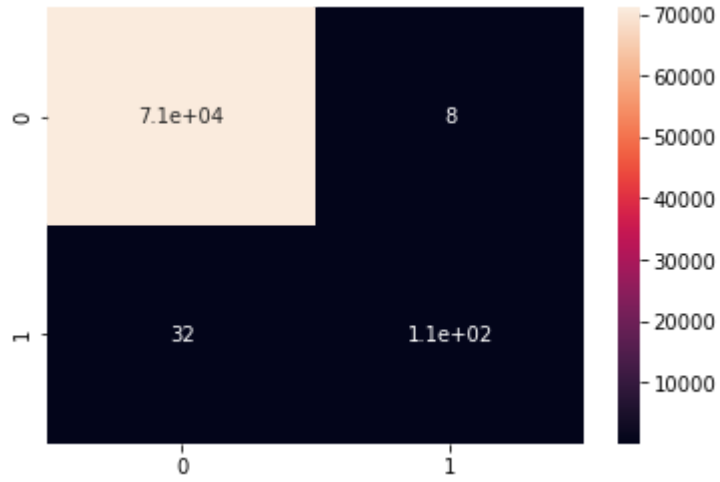
```
F1 score: 0.8437500000000001
```

* Confusion Matrix

In [64]:
```
rfcc = confusion_matrix(y_te, rfc_predict)
```

In [71]:
```
sns.heatmap(rfcc,annot=True)
```

Out[71]: <AxesSubplot:>



In [ ]:

In [32]:

Now lets try SMOTE method and check the difference

- Initilizing the smote values

In [55]:
```
sm = SMOTE(random_state=27)
X_train, y_train = sm.fit_resample(x_tr, y_tr)
```

- Applying the ml model over the smote values

In [58]:
```
smote = LogisticRegression(solver='liblinear').fit(X_train, y_train)
smote_pred = smote.predict(x_te)
```

```
C:\Users\ADMIN\anaconda3\lib\site-packages\sklearn\utils\validation.py:63: Data
ConversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(*args, **kwargs)
```

- Accuracy Score

In [60]:
```
accuracy_score(y_te, smote_pred)
```

Out[60]:
```
0.9833291199685402
```

- F1-Score

In [61]:
```
f1_score(y_te, smote_pred)
```

Out[61]:
```
0.17282229965156795
```

- Recall

In [62]:
```
recall_score(y_te, smote_pred)
```

Out[62]:
```
0.8857142857142857
```

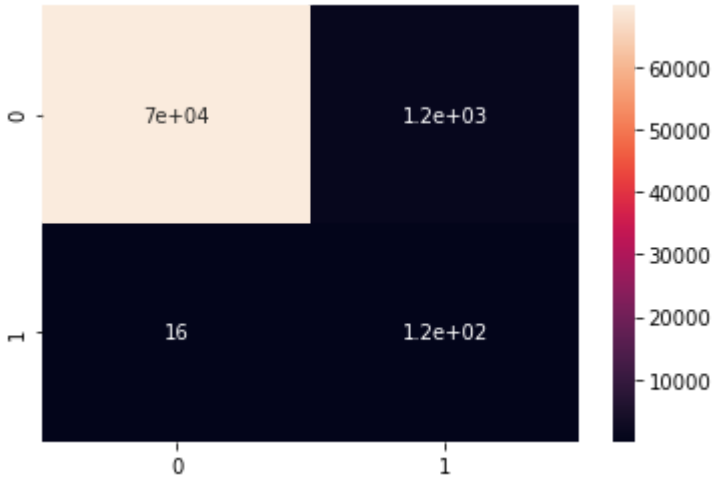
- Confussion matrix

In [66]:
```
confusion_matrix(y_te,smote_pred)
```

Out[66]:
```
array([[69891,  1171],
       [   16,   124]], dtype=int64)
```

In [69]:
```
sns.heatmap(confusion_matrix(y_te,smote_pred), annot=True)
```

Out[69]:
```
<AxesSubplot:>
```



Conclusion

- when we used the 1st method we get better accuracy but the accuracy is more or less equal to the % of non fraud transision i.e the model is baised
- when we use the 2nd method we had overcome the baised method and Model is perforfing better

Things i was willing to try

- comparing the models like Synthetic Minority Over-sampling Technique(SMOTE) AND Modified synthetic minority oversampling technique (MSMOTE)
- And also hyperparameter tuning of the model for better results that will help in better results in the real world.

* Regards

* Sankeerthan Reddy